Colin's How To… *R*

Good job you got some data!

Now to think about analysing it the right way to see if it shows anything…

Why *R* & not *Excel*/*SPSS*? 'Cause the skills are invaluable in **many growing** sectors today! & it's **FREE!**

In this Guide, I will be giving you scripts to use *R* effectively to manipulate data, and to perform (& understand the read-outs!) of statistical tests. I would **HIGHLY** recommend using *RStudio* (which runs *R* through its User-Interface) rather than using *R* directly. Download *R* first, and then *RStudio*, but after this you only need to load *RStudio* and use this.

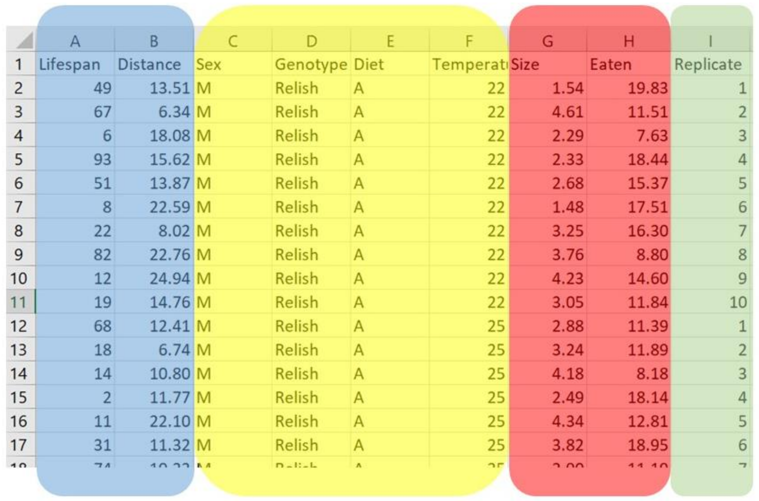
Actual scripts are either given in the boxes below or coloured in **blue** to separate them from notes:

```
R scripts in here
```

Read-Outs from *R* are highlighted in **grey.**

Any notes beginning with a ***#*** will be ignored if written into *R* (it sees ***#*** as a 'stop reading' sign).

The data used for this Guide can be downloaded here. You can download it, load it into *R*, and go through the various data manipulation / stats tests using the code provided. Here's a screenshot:



	A	B	C	D	E	F	G	H	I
1	Lifespan	Distance	Sex	Genotype	Diet	Temperat	Size	Eaten	Replicate
2	49	13.51	M	Relish	A	22	1.54	19.83	1
3	67	6.34	M	Relish	A	22	4.61	11.51	2
4	6	18.08	M	Relish	A	22	2.29	7.63	3
5	93	15.62	M	Relish	A	22	2.33	18.44	4
6	51	13.87	M	Relish	A	22	2.68	15.37	5
7	8	22.59	M	Relish	A	22	1.48	17.51	6
8	22	8.02	M	Relish	A	22	3.25	16.30	7
9	82	22.76	M	Relish	A	22	3.76	8.80	8
10	12	24.94	M	Relish	A	22	4.23	14.60	9
11	19	14.76	M	Relish	A	22	3.05	11.84	10
12	68	12.41	M	Relish	A	25	2.88	11.39	1
13	18	6.74	M	Relish	A	25	3.24	11.89	2
14	14	10.80	M	Relish	A	25	4.18	8.18	3
15	2	11.77	M	Relish	A	25	2.49	18.14	4
16	11	22.10	M	Relish	A	25	4.34	12.81	5
17	31	11.32	M	Relish	A	25	3.82	18.95	6

Our Response (i.e. Dependent) Variables are in blue; Categorical Independent Variables in yellow; Continuous Independent Variables in red; Random Effect in green

NOTE: This document describes how to use *R* to run particular tests, how to read outputs, and (to a degree) how to plot data. If you're still deciding which statistical test if appropriate for you, read my ***Colin's How To Stats*** guide which might be more helpful

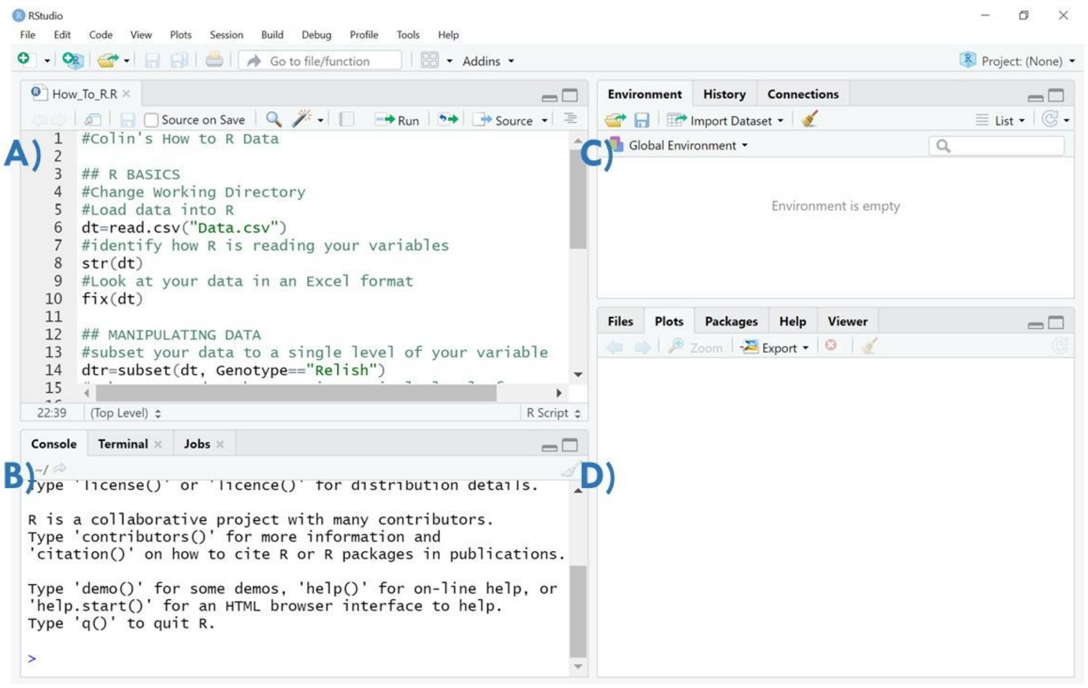
Contents

Getting Started with R

***RStudio* Basics**

As stated at the beginning of this guide, I would **HIGHLY** recommend using *RStudio* (which runs *R* through its User-Interface) rather than using *R* directly. Download *R* first, and then *RStudio*, but after this you only need to load *RStudio* and use this.

When you load *RStudio* you're presented with this Window:



```
#Colin's How to R Data

## R BASICS
#Change Working Directory
#Load data into R
dt=read.csv("Data.csv")
#identify how R is reading your variables
str(dt)
#Look at your data in an Excel format
fix(dt)

## MANIPULATING DATA
#subset your data to a single level of your variable
dtr=subset(dt, Genotype=="Relish")
```

```
Type 'license()' or 'licence()' for distribution details.

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

>
```

A) This window contains previously saved Scripts (.R files) which you can write *R* code and notes in for your own records. For every data set I analyse, I have an attaching .R Script file which contains the code I ran for the analysis, and the results I got in case I need to go back and do the analysis again when I find a mistake (which is WAAAAAAAAY too often)

B) This window is the *R* console itself. This is where you run your code, & where the magic happens

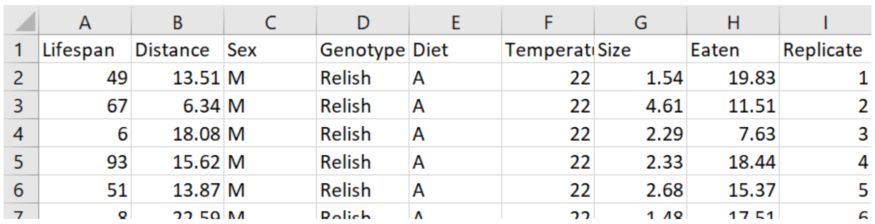
C) This is the *Global Enviornment*, and it's a record of all the different data sets, and models which you've run in your *R* session which is a REALLY handy reminder. So if I load a data set into *R*, it will appear in that window, as will any modifications that I make to that data set (e.g. if I subset the data, or remove columns etc.)

D) This window will show any plots I make which I can export as images or copy into *Clipboard*. Pretty handy!

Configuring your Data

First configure your data in such a way so it's readable in *R*.

If using an excel file, put it into this style of format and save as a .csv file!



	A	B	C	D	E	F	G	H	I
1	Lifespan	Distance	Sex	Genotype	Diet	Temperati	Size	Eaten	Replicate
2	49	13.51	M	Relish	A	22	1.54	19.83	1
3	67	6.34	M	Relish	A	22	4.61	11.51	2
4	6	18.08	M	Relish	A	22	2.29	7.63	3
5	93	15.62	M	Relish	A	22	2.33	18.44	4
6	51	13.87	M	Relish	A	22	2.68	15.37	5
7	8	22.59	M	Relish	A	22	1.48	17.51	6

NOTE: Any variables termed as ‘1, 2, 3’ etc. will be treated as numerical data by default (e.g. if you’ve recorded Replicate as 1, 2, etc., instead of 1st, 2nd, etc.), so best to use letters. You can change this in *R* (see **Setting up your Variables**) if you’d prefer.

Loading your Data into *R*

First load up *RStudio* (this also loads up *R*, & runs *R* inside *RStudio*, make sure you have both downloaded!) and change the directory to the file where your data is.

[Do this in ‘*Session*’→‘*Set Working Directory*’ and follow the file path to where your data is]

Then to read the file you want:

```
dt=read.csv(“Data.csv”)
```

- ***dt*** is the name you’re giving the data set you’re loading into *R*
- ***read.csv*** is a function in *R* that tells *R* you want it to read a specific *.csv* file containing data, and load it into the programme.
- ***(“Data.csv”)*** is the name of the file you want to load into *R* (you have to include the .csv for this to work).
- **=** tells *R* that everytime you refer to *dt* in the future, you mean the *Data.csv* file that you’ve loaded into it

You can get a summary (1st 5 lines) by writing (just to check you’re looking at the right thing):

```
head(dt)
```

Or you can view your data in a table format like in *Excel* but in *R* by:

```
fix(dt)
```

Setting up Variables Properly

Please note that any variables termed as ‘1, 2, 3’ etc. will be treated as numerical data by default (e.g. if you’ve recorded *Replicate* as 1, 2, etc. instead of 1st, 2nd, etc.), so best to use letters. You can check how *R* is reading your variables using:

```
str(dt)
```

You can change this in *R* by using the *as.factor* function:

```
dt$Replicate=as.factor
```

- ***dt*** here refers to the data set your using
- ***Replicate*** is the variable you're wanting to make a factor (instead of numeric),
- ***$*** tells *R* your looking for the *Replicate* variable, in the *dt* data set

A good video describing the process is here.

Subsetting your Data

It's really useful to subset your data to look at changes in means etc., say by *Genotype* for example:

```
dtr=subset(dt, Genotype=="Relish")
```

NOTE: It's important to name this subsetted data set something **DIFFERENT** from your original, full data set, otherwise you subsetted data set will override you original one!

You can also exclude a level of your variable using the following formula:

```
dtdu=subset(dt, Genotype!="Relish")
```

To choose 2 or more specific categories you can use:

```
dtdr=subset(dt, Genotype=="Relish"|Genotype=="Dif")
```

Understanding your Data

To get some characteristics of your data, you can use the *summary* function:

```
summary(dt)
```

To look at more specific aspects of your variables (means, medians, max, min, etc.) you can use the *tapply* function:

```
tapply(dt$Lifespan, dt$Genotype, mean)
```

- ***tapply*** is the function *R* uses to give you information about your data
- ***dt$Lifespan, dt$Genotype, mean)*** tells *R* that you want to get the *mean* values of the *Lifespan* variable, for each of the *Genotypes* from the dataset *dt*. **NB:** It's important that these are given in this order!

you can even use it to look at multiple independent variables at the same time:

```
tapply(dt$Lifespan, list(dt$Genotype,dt$Sex), mean))
```

My friend (the amazing Dr Weihao Zhong) has written a formula to measure the Standard Error (SE) as well through *tapply*:

```
se=function(x) sqrt(var(x,na.rm=T)/length(x)-length(which(is.na(x))))
tapply(dt$Lifespan, list(dt$Genotype,dt$Sex), se))
```

General Stats

Normality

To figure out what stats test is appropriate for your data, you first must identify whether your data follows a normal distribution or not.

You can do this through a statistical test, *Shapiro-Wilks*: (in this example we're testing the distribution of the response variable *Distance* in the data set *dt*)

```
shapiro.test(dt$Distance)
```

- ***shapiro.test*** is the name of the function which runs the Shapiro-Wilks normality test
- ***(dt$Distance)*** tells *R* you want to test the distribution of the continuous variable *Distance* in the data set you've called *dt*

  ***R* Read-out**

  W = 0.95572, p-value = 6.644e-14

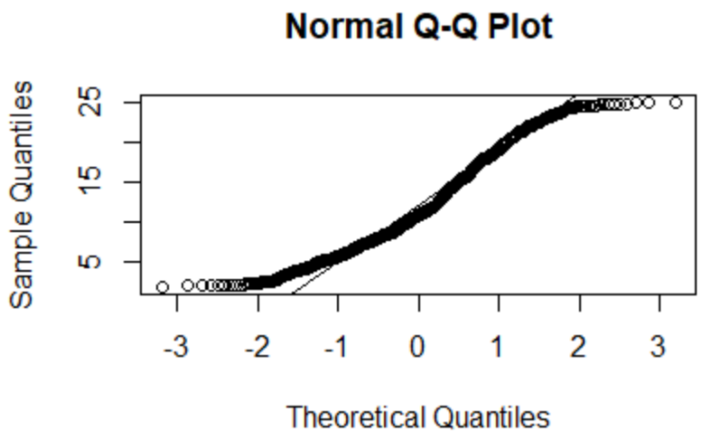
The variable is not normal as the *p* value is <0.05 (in this case *p* almost equals 0!)

This means I should use **Non-Parametric** tests which make fewer assumptions about the data distribution, but are weaker tests. Or I could transform it so it fits a normal distribution, but I won't.

You can also look at the distribution of your data using **Quantile-Quantile (Q-Q) plots**:

```
with(dt, qqnorm(Distance) + qqline(Distance))
```

- ***with(dt,*** tells *R* you want to use the test on the data set you've called *dt*
- ***qqnorm(Distance)*** plots the sample and theoretical quantiles
- ***qqline(Distance)*** draws a line through your points
- ***(Distance)*** tells *R* you want to test the distribution of the continuous variable *Distance*



Although the quantiles follow the projected line pretty well, it follows a sigmodal pattern, deviating from the line significantly, so it's not normally distributed (hence I should use non-parametric tests).

To learn more about how to interpret your qqplots, and determine whether your data is normally distributed, see this useful webpage from the University of Virginia.

Binomial Tests

Probably the simplest Stats test to do is the *binomial* test.

You use this test to see whether your ratios between 2 (i.e. binomial) levels, of your response (dependent) variable, are different from an expected ratio.

A great example is looking at sex ratio.

Say you were over-expressing a gene during development which you think is important in male-specific development. Say you get 17 female flies from a total of 96. You can load your data into *R* and do a *binomial* test:

```
binom.test(17, 96, p=0.5)
```

- ***binom.test*** tells *R* I want to perform a Binomial test
- ***17, 96,*** informs *R* that I got 17 'successes' out of a total of 96 'plays'
- ***p=0.5*** tells *R* that the Null Hypothesis would expect a 50% chance of male & female

R Read-out

- Exact binomial test
- data: 17 and 83
- number of successes = 17, number of trials = 96, p-value = 9.942e-11
- alternative hypothesis: true probability of success is not equal to 0.5
- 95 percent confidence interval:
- 0.1240805 0.3075558
- sample estimates:
- probability of success   0.2048193

p value is ~0 so overexpressing the gene is affecting sex ratio significantly

Proportionality Tests

If you want to check difference in proportions between 2 populations (so with no expected ratio), you can use the *proportionality* test which is a version of a *chi-squared* test.

Say if you wanted to assess the sex ratios between 2 populations (i.e. PopA = 17M 83F, PopB = 40M 60F), you have to input them into the test as 'successes' (x) and 'attempts' (n).

So say we take Males as 'successes' and the overall offspring in each population as 'attempts'.

The Males in each population are 17 & 40 respectively, the total offspring is 100 in each population:

```
prop.test(x=c(17,40), n=(100, 100)
```

R Read-out

- 2-sample test for equality of proportions with continuity correction
- X-squared = 11.8758, df = 1, p-value = 0.0005687 # *p* value indicates significant difference
- alternative hypothesis: two.sided
- 95 percent confidence interval:

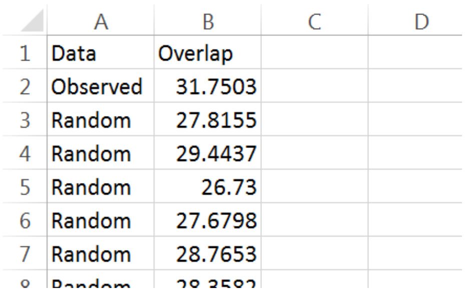
- -0.36099504 -0.09900496
- sample estimates:
- 0.17   0.40

<u>Wilcox Tests</u>

Say you have the overlap of genes from a data-base of interest, to those bound by a transcription factor you're working on. You have a single value (e.g. 31.8%), but you want to know whether this is significant from random genes. Do *Monte-Carlo* analysis (usually >= 100 iterations) with a random set of genes which number the same as in your observed data-set, so you get >= 100 values.

To test the difference between these 100 or so *Monte-Carlo* values and your 1 observational point, you can use a *Wilcox* test.

Have your data as so:



	A	B	C	D
1	Data	Overlap		
2	Observed	31.7503		
3	Random	27.8155		
4	Random	29.4437		
5	Random	26.73		
6	Random	27.6798		
7	Random	28.7653		
8	Random	28.3582		

and use the following script:

```
wilcox.test(dt$Overlap~dt$Data)
```

Single-Variable, Two-Level Tests

Discrete Variable

So say you want to see the difference in a continuous variable (e.g. Lifespan) between a variable which has two-levels (e.g. Sex which has ‘males’, and ‘females). This is pretty simple to do.

For **Normal Data** – You use a *Welch t-test*:

```
t.test(dt$Lifespan~dt$Sex)
```

- ***t.test*** is the function in *R* for the *t-test*
- ***dt$Lifespan*** is the dependent (response) variable we’re measuring in data set *dt*
- ***~dt$Genotype*** is telling *R* that we want to look at the impact of our independent variable *Genotype*, on the distance travelled by the animal (i.e. our dependent variable)

R Read-out

Welch Two Sample t-test

data: Lifespan by Sex

t = 4.0265, df = 698.87, p-value = 6.279e-05# *p* is tiny, so the Sexes significantly differ

mean in group F mean in group M

45.40278 37.51944 # you can see Females (i.e. *F*) live longer

For **Non-Parametric Data** – You use a *Wilcox Rank Sum*:

```
wilcox.test(dt$Lifespan~dt$Sex)
```

R Read-out

Wilcoxon rank sum test with continuity correction

W = 74809, p-value = 0.0003348 # you can see the *p* value (while <0.05) is much higher than in the t-test, and that’s because non-parametric tests are much weaker than parametric

Continuous Variable

Say you want to see if there’s a difference between two numeric variables, e.g. amount eaten and body size.

If you want to know whether there’s a significant relationship between the two variables, and if you’re response variable (i.e. amount Eaten) is normal, then you can use a simple **Pearson’s Correlation**, which can be down using the script below:

```
cor.test(dt$Lifespan, dt$Size, type="pearson")
```

***R* Read-Out**

```
Pearson's product-moment correlation

data:  Lifespan and Size

t = 2.706, df = 718, p-value = 0.006971 # p value <0.05 therefore significant correlation

    cor

0.1004775        # the correlation between Lifespan and Size is 0.1 (i.e. a slight +ve)
```

More information regarding correlations in *R* can be found here.

If your data is not normal, you have to use the non-parametric equivalent which is the **Spearman's Correlation** test:

```
cor.test(dt$Lifespan, dt$Size, type="spearman")
```

To understand how one of the variables **can predict** the other you can run a **Regression**. This can be done by running a simple **Linear Model:**

```
m1=lm(dt$Lifespan~dt$Size)

summary(m1)
```

***R* Read-Out**

```
              Estimate      Std. Error     t value      Pr(>|t|)

(Intercept)   34.8081       2.6485         13.143       < 2e-16 ***

Size          1.7799        0.6578         2.706        0.00697 **

Multiple R-squared:  0.0101,   Adjusted R-squared:  0.008717

F-statistic: 7.323 on 1 and 718 DF,  p-value: 0.006971
```

Here we can see that the *Size* is a significant predictor of *Lifespan* (F = 7.32; *p* = 0.007) but the association between the two variables is pretty weak, as R-squared = 0.01.

This means that *Size* explains only 1% (i.e.0.01) of *Lifespan*, but it does so reliably, and therefore the relationship is significant (i.e. *p* = 0.007)

Visualising Data

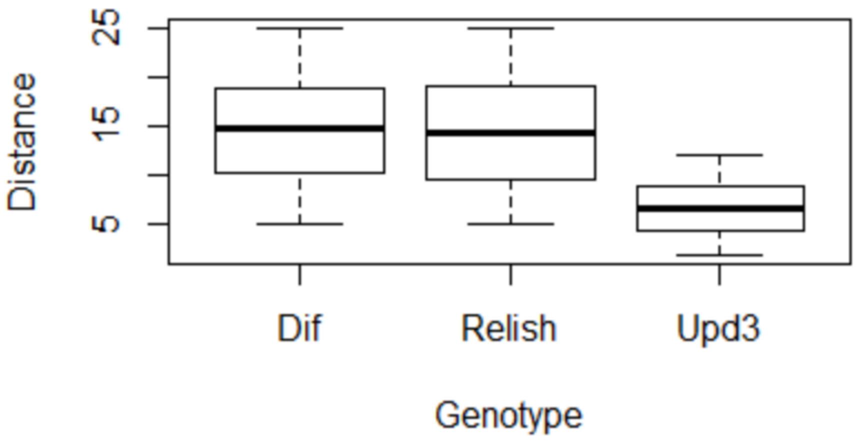
Discrete Variable

You can **visualise the data** pretty easily using the normal plot function:

```
boxplot(dt$Distance~dt$Genotype)
```

You can adjust the y-axis range, or the x & y-axes labels with the following additions:

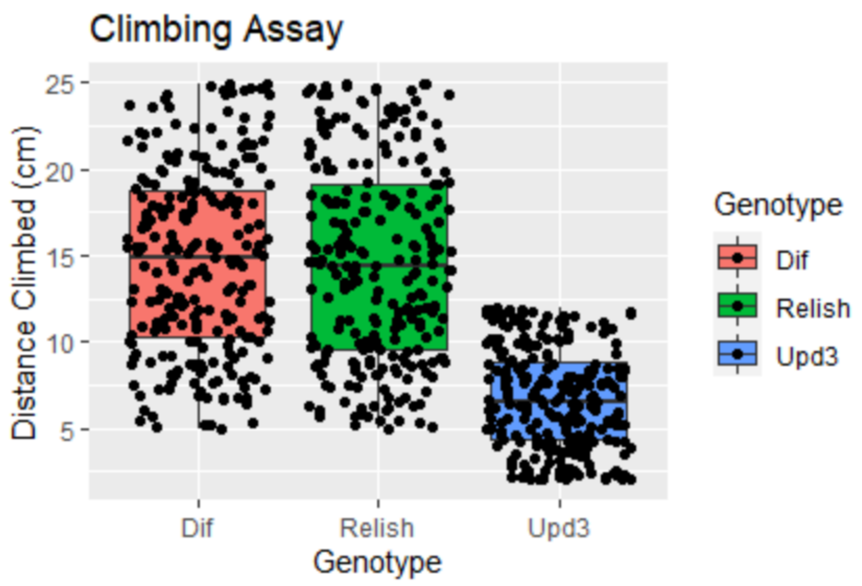
```
boxplot(dt$Distance~dt$Genotype, ylim=c(0,25), xlab=”Genotype”, ylab=”Dista
nce”)
```



Or if you want something nicer, load the ***ggplot2*** package (‘*Packages*’ → ‘*Install Package*’ [if first time], or ‘*load package*’ if already installed in previous sessions, or load in terminal)

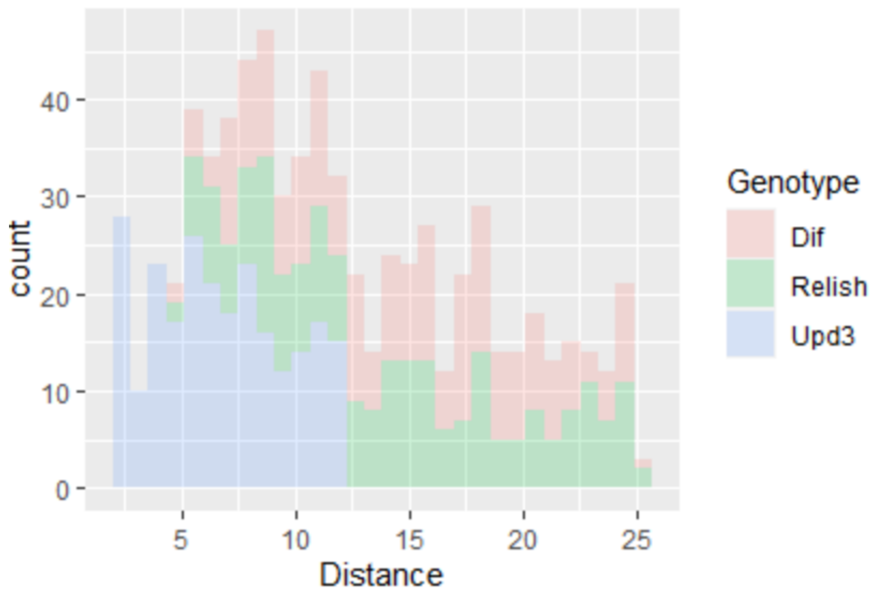
```
library(ggplot2)

qplot(Genotype, Distance, data=dt, geom=c("boxplot", "jitter"),

   fill=Genotype, main="Climbing Assay",

   xlab="Genotype", ylab="Distance Climbed (cm)")
```

- ***qplot*** is the function in *R* for *plotting* within the package *ggplot2*
- ***Genotype*** is telling *R* to plot Genotype on the x-axis
- ***Distance*** is the dependent (response) variable we’re measuring, and want it on the y-axis
- ***data=dt*** is the telling *R* we want to take the data from the data set we’ve called *dt*
- ***geom=c(“boxplot”, “jitter”)*** lets *R* know what kind of visualisation we want (i.e. Boxplot)
- ***fill=Genotype*** tells *R* we want to colour in the data using the number of ‘*levels’* within the variable *Genotype*. Because there’s 2 ‘*levels*’ here (i.e. Relish & Upd3), it gives red & blue
- ***main=”Climbing Assay”*** tells *R* to label the graph ‘*Climbing Assay*’ at the top
- ***xlab*** and ***ylab*** tells *R* what to call the x-axis and y-axis respectively



Alternatively, you can view the response variable between 2 populations as an overlapping Histogram

```
ggplot(dt, aes(x=Distance, fill=Genotype)) + geom_histogram(alpha=0.2, posi
tion="identity")
```

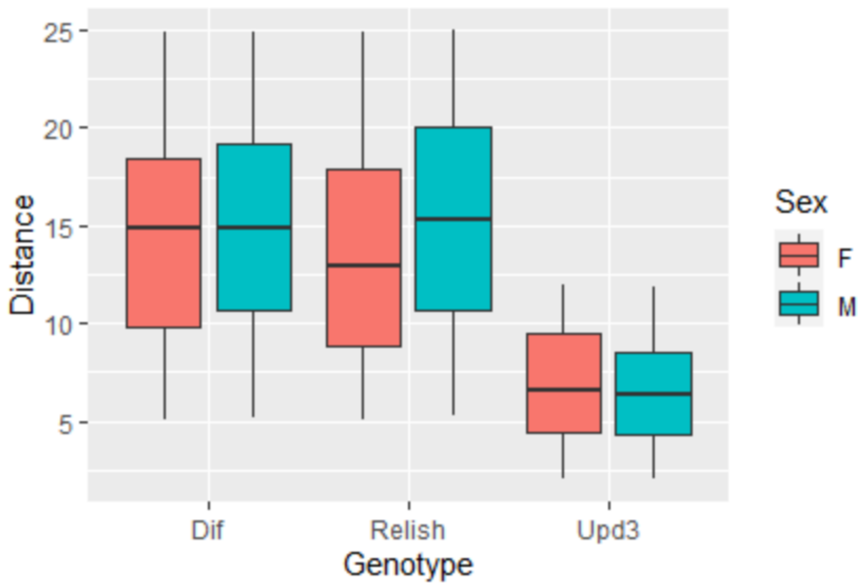


Two-Level Boxplots

Say if you wanted to split a variable (e.g. Genotype) by another (e.g. Sex), and to see if there might be a difference between them, you can make a boxplot like this too!

```
ggplot(aes(y=Distance, x=Genotype, fill=Sex), data=dt) + geom_boxplot(posit
ion=position_dodge(width=.9))
```

- ***geom_boxplot*** lets *R* know what kind of visualisation we want (i.e. Boxplot)
- ***position=position_(width=.9)*** physically separates the *Sex* sections under *Genotype*



25
20
15
10
5
Distance
Dif
Relish
Upd3
Genotype
Sex
F
M

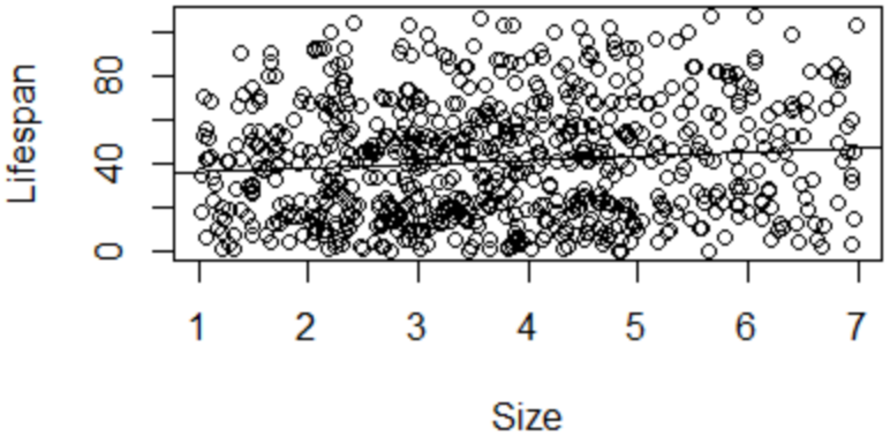
Continuous Variable

To **plot correlations** you can write:

```
plot(dt$Lifespan~dt$Size) + abline(lm(dt$Lifespan~dt$Size))
```

NB: The variable that you put first (i.e. *Lifespan* in this case) will be on the y-axis

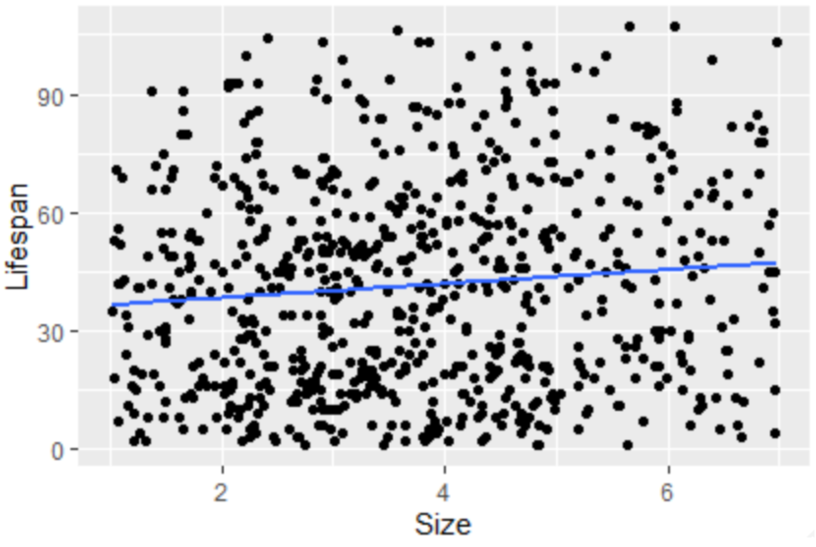
- ***abline(lm(dt$Lifespan~dt$Size)*** adds a line-of-best-fit to the correlation



To make it a bit more customisable and nicer looking you can use other packages, i.e. *ggplot2*

```
install(ggplot2)

ggplot(dt, aes(x=Size, y=Lifespan) + geom_point() + geom_smooth(method=lm,
se=FALSE)
```

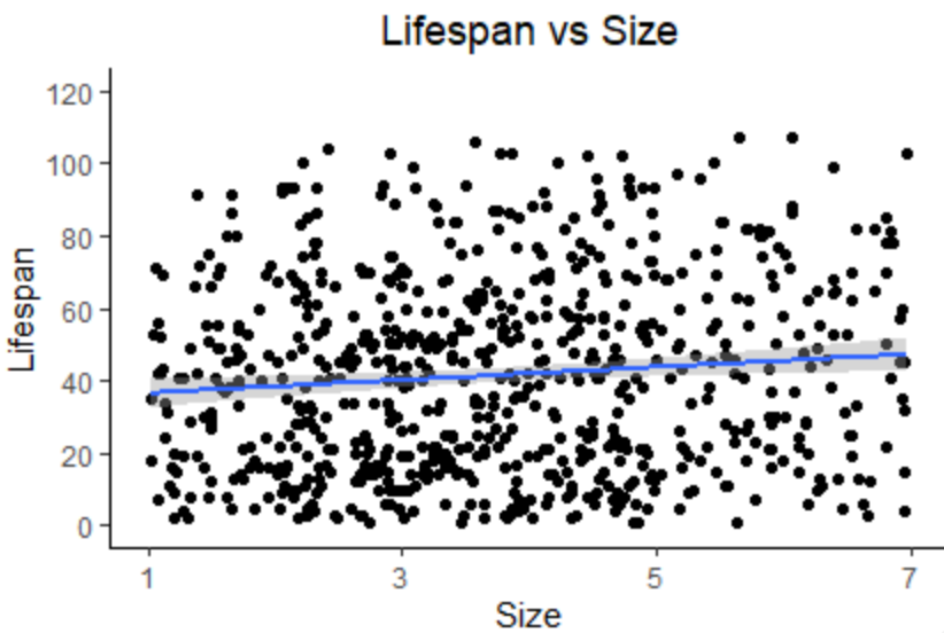
- ***ggplot*** is the function in *R*;
- ***dt*** is name I've given the data set in *R*;
- ***aes(x=Eaten, y=Size)*** is telling *R* to put the *x*-axis as *Eaten*, and the *y*-axis as *Size*;
- ***+ geom_point()*** tells *R* that I want *R* to show me each individual data point;
- ***+ geom_smooth(method=lm, se=FALSE)*** tells *R* to show a line-of-best-fit, ensuring it uses a line, and not to show the standard error in the values (comes up as a shadow around the line, see below)



The good thing with ggplot2 is that you can makes this really customised. For example, using the following script (I know it looks a bit much, but it's OK when broken down):

```
ggplot(dt, aes(x=Size, y=Lifespan)) + geom_point() + ggtitle("Lifespan vs S
ize") + geom_smooth(method=lm, se=TRUE) + scale_x_continuous(name = "Size",
limits = c(1, 7), breaks = seq(1, 7, 2)) + scale_y_continuous(name = "Lifes
pan", limits = c(0, 120), breaks = seq(0, 120, 20)) + theme(plot.title = el
ement_text(hjust = 0.5), panel.background = element_blank(), axis.line = el
ement_line(color="black"))
```

- *ggtitle("Lifespan vs Size")* tells *R* you want to give the graph the title "Lifespan vs Size"
- *scale_x_continuous(name = "Size", limits = c(0, 8), breaks = seq(0, 8, 2))* tells *R* that the *x*-axis should be given the name "Size", it should go from values 0→8, and the points on the scale should go from 0→8 with intervals of 2
- *scale_y_continuous(name = "Lifespan", limits = c(0, 120), breaks = seq(0, 120, 20))* tells *R* that the *y*-axis should be given the name "Lifespan", it should go from values 0→120, and the points on the scale should go from 0→120 with intervals of 20
- *theme(plot.title = element_text(hjust = 0.5)* tells *R* to label the axis at the mid-point
- *panel.background = element_blank()* tells *R* that the grey-panelled background that's default (see figure above) should be removed
- *axis.line = element_line(color="black")* tells *R* to draw a black line on the y axis



Two-Variable / >Two-Level Tests

>Two-Level Tests

Say you want to assess the difference between the Lifespans of flies conditioned to 4 different diets.

If the expression data is **normally distributed**, you can use a **One-Way ANOVA** to assess the effect of each treatment. Ensure you name your ANOVA something, in this example I've called it '*fit*':

```
fit=aov(dt$Lifespan~dt$Diet)

summary(fit)
```

***R* Read-Out**

```
            Df    Sum Sq    Mean Sq    F value    Pr(>F)
Diet         3    1515      504.8      0.716      0.543 # p > 0.05
Residuals  716    505066    705.4
```

If your response variable isn't normally distributed, you can use a **Kruskal-Wallis** test to do the same thing an ANOVA does:

```
kruskal.test(dt$Lifespan~dt$Diet)
```

***R* Read-Out**

```
Kruskal-Wallis chi-squared = 2.4611, df = 3,

p-value = 0.4824
```

So not a significant effect between diets… , but what about **between each Diet**?

For this you need **Post-Hoc** analysis.

For the ANOVA this is pretty straight forward, you use a Tukey's test:

```
TukeyHSD(fit)
```

***R* Read-Out**

```
     diff      lwr        upr         p adj
B-A  0.3444444 -6.864742  7.553631    0.9993342
C-A  3.3444444 -3.864742  10.553631   0.6304511
D-A  2.7111111 -4.498075  9.920297    0.7674907
C-B  3.0000000 -4.209186  10.209186   0.7069972
```

```
D-B  2.3666667 -4.842520         9.575853          0.8327234

D-C -0.6333333 -7.842520         6.575853          0.9959167
```

P adj is the important value here, everything's significant with each other

For non-parametric data, it's a little easier, as you can complete a **Dunn Test** which will complete BOTH the overall Kruskal-Wallis test, AND the Dunn pair-wise comparisons:

```
install.packages("dunn.test")

dunn.test::dunn.test(dt$Lifespan, dt$Diet)
```

ALTERNATIVELY

For Kruskal-Wallis analysis, you use the crazy named **Kruskal-Wallis-Nemenyi** test. You need to load the **PMCMR package** (updated in January 2018 to **PMCMRPlus**):

```
library(PMCMRplus)

kwAllPairsNemenyiTest(Lifespan~Diet, dt)
```

***R* Read-Out**

```
        A       B       C

B       1.00    -       -

C       0.61    0.65    -

D       0.73    0.76    1.00
```

So 'A' & 'C' are most different (p = 0.61), but no Diets are different from one-another significantly

Multi-Variable Tests

Sometimes you want to measure the effect of more than one factor e.g. the lifespan of flies under different diets and temperatures. In this scenario you want to see whether there are independent effects of diet and temperature, but also whether there is an interaction between these variables on longevity (e.g. does Diet affect Lifespan differently at one temperature than it does at another). For this you can use a **Two-Way ANOVA** for normal data when dealing with **categorical factors**.   **NB:** Use '*' to look at factor & interaction, and ':' for just the interaction

```
m1=aov(dt$Lifespan~dt$Diet*dt$Temperature)

summary(m1)
```

***R* Read-Out**

```
                    Df     Sum Sq      Mean Sq     F value  Pr(>F)

Diet                3      1515        505         0.776    0.507

Temperature         2      43583       21791       33.516   1.24e-14 ***

Diet:Temperature    6      1155        193         0.296    0.939

Residuals           708    460328      650
```

From these results, we can see that Temperature has a significant effect on Lifespan, but not Diet, nor the interaction between the two variables.

Again, we can use Tuskey's HSD to see the pairwise interactions:

```
TukeyHSD(m1)
```

***R* Read-Out**

```
$Diet

       diff         lwr          upr          p adj

B-A    0.3444444    -6.577009    7.265898     0.9992479

C-A    3.3444444    -3.577009    10.265898    0.5988103

D-A    2.7111111    -4.210343    9.632565     0.7444022

C-B    3.0000000    -3.921454    9.921454     0.6796821

D-B    2.3666667    -4.554787    9.288120     0.8149808

D-C    -0.6333333   -7.554787    6.288120     0.9953930


$Temperature

       diff         lwr          upr          p adj
```

```
25-22   -12.200000     -17.66694      -6.733055      0.0000006
28-22   -18.779167     -24.24611      -13.312222     0.0000000
28-25   -6.579167      -12.04611      -1.112222      0.0134064


$`Diet:Temperature`
                 diff        lwr            upr            p adj
B:22-A:22   1.5666667   -13.698624     16.83195707    1.0000000
C:22-A:22   3.5833333   -11.681957     18.84862374    0.9998022
D:22-A:22   0.9333333   -14.331957     16.19862374    1.0000000
A:25-A:22 -12.7500000   -28.015290     2.51529040     0.2095667
B:25-A:22 -14.4000000   -29.665290     0.86529040     0.0859197
…                         …        …              …              …
```

When you have **non-parametric** distributions, you can't use Kruskal-Wallis as it only works on one-way analysis. Ordinal Logistic Regressions are pretty bad as your response variable has to be categorical with >=2 levels…

So use **Generalised Linear Models (GLM)**!

GLMs are pretty much the same as a **Linear Model** as used above (i.e. in Regression analysis), but can use variables of different types (i.e. not just continuous variables).

However, these models require you to know a little more about your data…

If your data follows a Guassian distribution, you can use a Linear Model, but if it follows any other distribution (e.g. Poisson, binomial, etc. see this site for details), use a GLM, and specify the distribution in the formula (as below). **NB:** Requires the Package lmer4

```
library(lme4)
a=glm(Lifespan~Diet*Temperature, family=poisson, data=dt)
summary(a)
```

***R* Read-Out**

```
                 Estimate         Std. Error        z value          Pr(>|z|)
(Intercept)      3.917342         0.018209          215.133          < 2e-16
DietB            0.030691         0.025556          1.201            0.22977
DietC          0.068860           0.025319          2.720            0.00653
…                …                ….                …                …
```

Comparing the effects of multiple variables

When you have 2 or more variables that affect your response variable (e.g. like Diet & Temperature), you will want to identify the impact of each of these variables independently, as well as whether any interactions exist between them.

You get some indication of the impact of these elements individually from an ANOVA or a GLM (like in the example above, LINK), but a better way is to assess the difference in how Models Fit the data using an ANOVA. If you do this in a step-wise manner, you can identify whether including a variable in your test makes a significant difference to how the model fits the data or not, and thus whether it affects your response variable or not. This is known as a ***Partial F Statistic***. Here's a good video describing the process (Example 1 describes a **Multiple Linear Regression**, while Example 2 describes a **Non-linear Regression** of a single variable).

NB: The **Partial F Statsitic** measure only works (i.e. gives you a significance value) with **Linear Models** (i.e. *lm* or *lmer* models, see below) and not **Generalised Linear Models** (i.e. *glm* or *glmer* models).

```
model1 = lm(dt$Lifespan ~ dt$Diet + dt$Temperature)
model2 = lm(dt$Lifespan ~ dt$Diet)
anova(model1, model2)
```

***R* Read-Out**

```
        Res.Df   RSS            Df       Sum of Sq       F              Pr(>F)
1       714      461484
2       716      505066         -2       -43583          33.715         1.02e-14 ***
```

We can see from the **Residual Sum Squares (RSS)** [this is a value which indicates how far the model deviates from the data points, so a lower number is better], that the 1st model fits better (i.e. the one with temperature included) than the 2nd model (i.e. has a lower RSS value), and we see from the *p* value, that this difference is significant. Therefore, we can say that Temperature has a significant effect on Lifespan

Mixed-Effects Models

The word Mixed here means the inclusion of a ***Random Effect*** in your model, which just means you want to identify whether some variable in your data which you don't intend on measuring or investigating affects your response variable significantly or not.

This could be the age of the individual testes, the location the organisms are in the incubator, or the batch (e.g. Replicate) in which the samples were processed in. Including these as Random Effects in your model allow us to account for inherent lumpiness of data caused by such factors. Like ordinary GLMs, they require the package lme4

If your data follows a **Gaussian distribution**, use a **linear mixed model (*lmer*)**, otherwise, you can use a *glmer*, but be aware that **Partial F Statistics** won't give a *p* value if comparing two *glmer* models.

```
m1=lmer(Lifespan~Diet+Temperature+(1|Replicate), data=dt)
```

```
summary(m1)
```

Again, we use **Partial F Statistics** to identify whether the variable has a significant effect:

```
model1 = lmer(Lifespan~Diet+(1|Replicate), data=dt)
model2 = lmer(Lifespan~Diet, data=dt)
anova(model1, model2)
```

R Read-Out

```
     Df    AIC  BIC      logLik     deviance   Chisq  Chi Df  Pr(>Chisq)
2    4     6707 6725.3   -3349.5    6699
1    5     6709 6731.9   -3349.5    6699       0      1       1
```

The difference between the models here is very small (AIC & BIC are values which indicate how well the model fits [lower is better], and they're very similar between the models), and this is reflected in the p value (p = 1) which shows the **Random Effect**, *Replicate*, does not affect the data.

NB: You can use:

```
(1|Replicate)
```

If you can assume the random effect is the same for every individual sample.

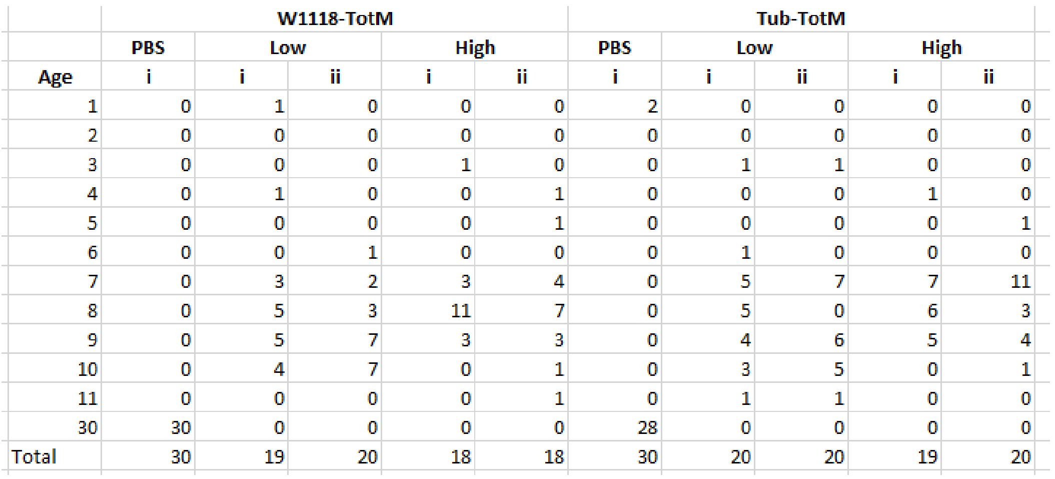
Alternatively, if you have reason to believe the random effect might affect samples differently between the different levels of one of your variables, you can use the formula

```
(Diet|Replicate)
```

Collating & Analysing Survival Data

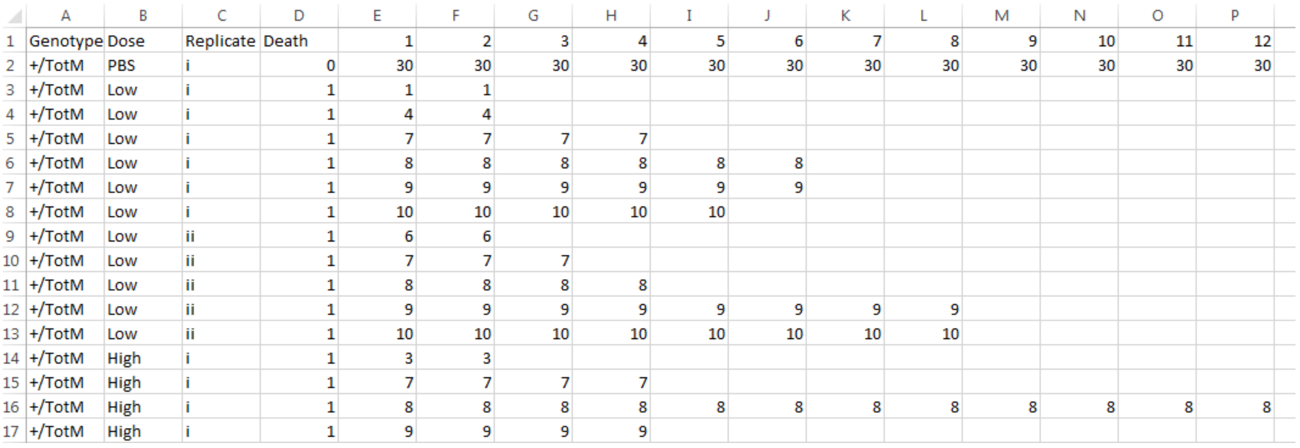
Configuring for Survival Analysis

Say you have data in an Excel file like:



	W1118-TotM					Tub-TotM				
	PBS	Low		High		PBS	Low		High	
Age	i	i	ii	i	ii	i	i	ii	i	ii
1	0	1	0	0	0	2	0	0	0	0
2	0	0	0	0	0	0	0	0	0	0
3	0	0	0	1	0	0	1	1	0	0
4	0	1	0	0	1	0	0	0	1	0
5	0	0	0	0	1	0	0	0	0	1
6	0	0	1	0	0	0	1	0	0	0
7	0	3	2	3	4	0	5	7	7	11
8	0	5	3	11	7	0	5	0	6	3
9	0	5	7	3	3	0	4	6	5	4
10	0	4	7	0	1	0	3	5	0	1
11	0	0	0	0	1	0	1	1	0	0
30	30	0	0	0	0	28	0	0	0	0
Total	30	19	20	18	18	30	20	20	19	20

To get it ready for R you have to configure it as so:



	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
1	Genotype	Dose	Replicate	Death	1	2	3	4	5	6	7	8	9	10	11	12
2	+/TotM	PBS	i	0	30	30	30	30	30	30	30	30	30	30	30	30
3	+/TotM	Low	i	1	1	1										
4	+/TotM	Low	i	1	4	4										
5	+/TotM	Low	i	1	7	7	7	7								
6	+/TotM	Low	i	1	8	8	8	8	8	8						
7	+/TotM	Low	i	1	9	9	9	9	9	9						
8	+/TotM	Low	i	1	10	10	10	10	10							
9	+/TotM	Low	ii	1	6	6										
10	+/TotM	Low	ii	1	7	7	7									
11	+/TotM	Low	ii	1	8	8	8	8								
12	+/TotM	Low	ii	1	9	9	9	9	9	9	9	9				
13	+/TotM	Low	ii	1	10	10	10	10	10	10	10	10				
14	+/TotM	High	i	1	3	3										
15	+/TotM	High	i	1	7	7	7	7								
16	+/TotM	High	i	1	8	8	8	8	8	8	8	8	8	8	8	8
17	+/TotM	High	i	1	9	9	9	9								

You do this in Excel for each replicate, treatment and factor individually. The formula can be found in the '*Survival Transformation Example.xls*' file.

Once you have transformed each of your replicates like above, collate them with the appropriate information. Before you save the file, you have to **DELETE** the first data column (column E in this example) as it is just a list of the days survival was recorded. Once this has been done you can save the file as a .csv called "*Survival Data for R.csv*".

Once in this format you can use a function made by the fantastic **Dr Weihao Zhang** to transpose it in R:

Weihao's Script for Transposing Survival Data

```
raw=read.csv("Survival Data for R.csv")
# make function
make.rm<-function(constant,repeated,data,contrasts) {
 if(!missing(constant) && is.vector(constant)) {
  if(!missing(repeated) && is.vector(repeated)) {
   if(!missing(data)) {
    dd<-dim(data)
    replen<-length(repeated)
    if(missing(contrasts))
     contrasts<-
      ordered(sapply(paste("T",1:length(repeated),sep=""),rep,dd[1]))
    else
     contrasts<-matrix(sapply(contrasts,rep,dd[1]),ncol=dim(contrasts)[2])
    if(length(constant) == 1) cons.col<-rep(data[,constant],replen)
    else cons.col<-lapply(data[,constant],rep,replen)
    new.df<-data.frame(cons.col,
     repdat=as.vector(data.matrix(data[,repeated])),
     contrasts)
    return(new.df)
   }
  }
 }
 cat("Usage: make.rm(constant, repeated, data [, contrasts])\n")
 cat("\tWhere 'constant' is a vector of indices of non-repeated data and\n"
)
 cat("\t'repeated' is a vector of indices of the repeated measures data.\n"
)
}
dt=make.rm(1:4, 5:112, raw) # **CHANGE THESE!!!!!!**
dt.no.NAs=dt[complete.cases(dt),]
write.csv(dt.no.NAs,file = "Transposed survival data.csv", row.names = FALS
E)
```

** Change highlighted numbers depending on how many category (e.g. Genotype, Diet, Death etc.) and data columns (i.e. days lifespan was recorded) you have in your data set.

Delete 'contrasts' column and change 'repdat' to 'Lifespan' in the 'Transposed survival data' file, save it (again as a .csv file) & it's ready for analysis!

Analysing Survival Data

Once you have your data in R, there are a number of ways you can analyse your data

Load data

```
dt=read.csv("Survivals.csv")
```

Load (or install, then load) survival package

```
library(survival)
```

You want to have a general look at the data to see if there are any immediate differences (i.e. mean lifespan with standard errors).

You first have to create a formula for the standard error calculation.

Thankfully the brilliant **Dr Weihao Zhang** also made a function for this called se:

```
se=function(x) sqrt(var(x,na.rm=T)/(length(x)-length(which(is.na(x)))))
```

So subset the data as you wish and work out the means & se for Lifespan:

```
dti=subset(dt, Dose!="PBS")
tapply(dti$Lifespan, list(dti$Genotype, dti$Dose), mean)
se=function(x) sqrt(var(x,na.rm=T)/(length(x)-length(which(is.na(x)))))
tapply(dti$Lifespan, list(dti$Genotype, dti$Dose), se))
```

```
            High        Low
+/TotM      7.680851    8.140000
tub/TotM    7.604167    8.115385
            High        Low
+/TotM      0.2528403   0.292784
tub/TotM    0.1900438   0.280935
```

So not much difference in the means between the Genotypes…

You can test these statistically using a Wilcox test (lifespan data isn't normally distributed)

```
wilcox.test(dtl$Lifespan ~ dtl$Genotype)
W = 863.5, p-value = 0.4037
wilcox.test(dth$Lifespan ~ dth$Genotype)
W = 798.5, p-value = 0.2855
```

Model Analysis

Now because I have infection data in this example, there's not much point in me doing complex analysis on survival curves using model fitting, as the survivals crash, so I will use the Cox Hazard Proportion model. I will use the *dti* data set to see if there's a Genotype effect:

First load/install the *Survival* package which contains the *coxph()* function

```
library("survival")
coxph(Surv(Lifespan, Death)~Genotype,dti)
coxph(formula = Surv(Lifespan, Death) ~ Genotype, data = dti)
```

```
                           coef     exp(coef)      se(coef)      z      p
Genotype      tub/TotM     0.0401     1.04         0.143         0.28   0.78
```

Likelihood ratio test=0.08 on 1 df, p=0.779 n= 197, number of events= 197

Nope, doesn't look like it…

What about a Genotype:Dose interaction?

```
m1<-coxph(Surv(Lifespan, Death)~Genotype*Dose,dti)
m2<-update(m1,~.- Genotype:Dose )
anova(m1,m2)
```

Cox model: response is Surv(Lifespan, Death)

Model 1: ~ Genotype * Dose

Model 2: ~ Genotype + Dose

```
      loglik      Chisq       Df     P(>|Chi|)
1     -841.03
2     -841.74     1.4183      1      0.2337
```

Nope, no interaction either

Plotting Cox HP

Say I wanted to make a graph of the Cox HP results.

Let's go back to our Genotype results:

```
                           coef     exp(coef)      se(coef)      z      p
Genotype      tub/TotM     0.0401     1.04         0.143         0.28   0.78
```

Here the *exp(coef)* is the hazard proportion relative to the first genotype assessed, which in this case is *+/TotM*, the control Genotype. (Subsets are processed in alphabetical order, so the subset you want to relate everything to has to alphabetically come first, thankfully '+' comes before all letters!).

The *se(coef)* is the standard error of the coefficient, although I think if your *exp(coef)* is >1, you have to multiply the *se(coef)* by the *exp(coef)* to get the true standard error.

I would put these values into a new *.csv* file and plot them using the *Plotrix* package:

```
dt=read.csv("Surv Fig.csv")

attach(dt)

library(plotrix) # or install first!

par(mar=c(5, 5, 4, 2))
```

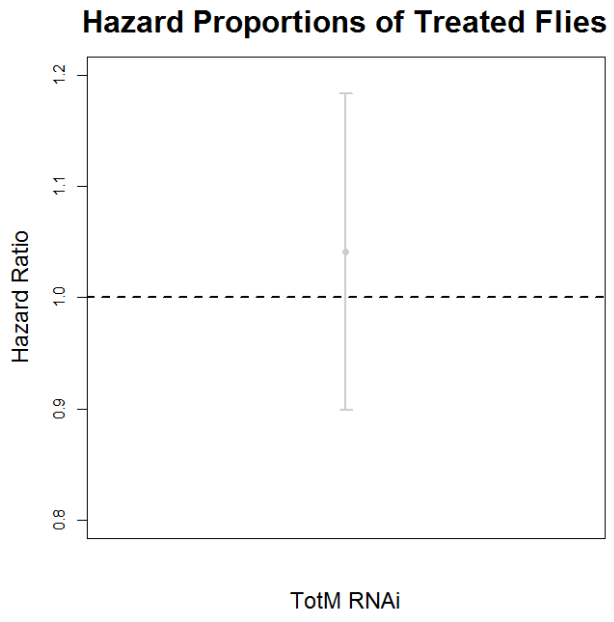
This sets the position of graph in space, best not to touch!

```
plotCI(x=c(1.5), 1.041, uiw=0.142, liw=0.142, col=c('grey80'), lwd=2, pch=1
6, err="y", xaxt="n", xlab="TotM RNAi", ylab="Hazard Ratio", main="Hazard P
roportions of Treated Flies", cex.main=2, cex.lab=1.5, ylim=c(0.8, 1.2), xl
im=c(1,2))
```

Lots of stuff in this line... *uiw* & *liw* are the error limits, *pch* is the point type, etc.

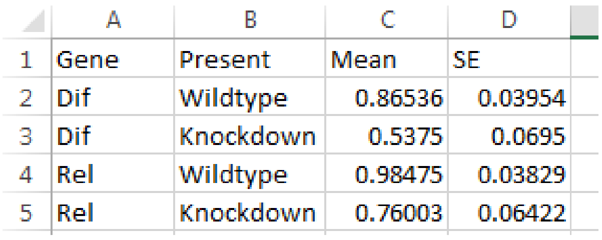
```
abline(h=1, col=1, lty=2, lwd=2)
```

This adds an additional line in, *h=1* means that it's horizontal and is at 1, *col* is the colour, *lty* is line type, and *lwd* is line width



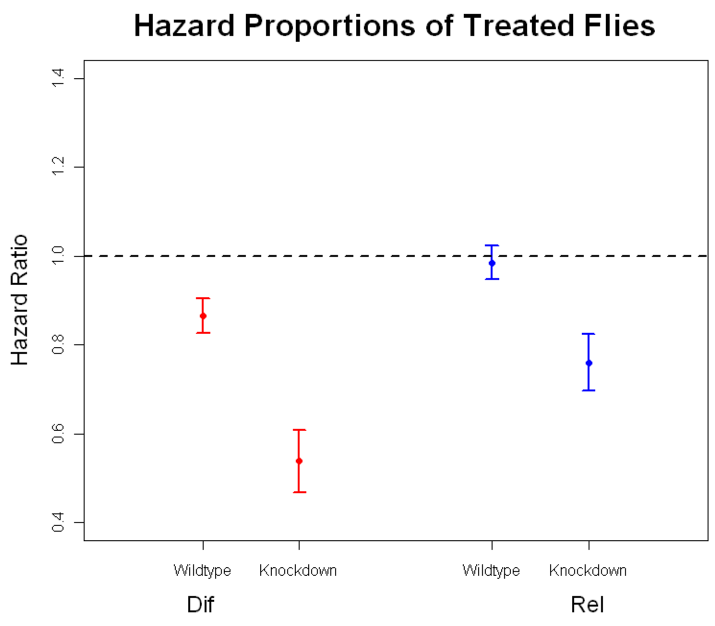
Or if using more data points

Make a *.csv* file of CoxHP data:



	A	B	C	D
1	Gene	Present	Mean	SE
2	Dif	Wildtype	0.86536	0.03954
3	Dif	Knockdown	0.5375	0.0695
4	Rel	Wildtype	0.98475	0.03829
5	Rel	Knockdown	0.76003	0.06422

```
attach(dt)

par(mar=c(5, 5, 4, 2)) #sets position of graph in space

plotCI(x=c(1.5, 2, 3, 3.5), Mean, uiw=SE, liw=SE, col=c('black', 'black', '
grey65', 'grey65'), lwd=2, pch=16, err="y", xaxt="n", xlab="Dif
Rel", ylab="Hazard Ratio", main="Hazard Proportions of Treated Flies", cex.
main=2, cex.lab=1.5, ylim=c(0.4, 1.4), xlim=c(1, 4))

# Lots of stuff in this line... uiw & liw are the error limits, pch is the
point type, etc.

chp<-as.character(dt[[2]]) # stating which column in csv file to put as x a
xis

axis(side=1,at=c(1.5, 2, 3, 3.5), labels=chp, cex.axis=1) # stating space o
f axis, axis labels, cex.axis describes axis font size

abline(h=1, col=1, lty=2, lwd=2) # adds an additional line in, h=1 means th
at it's horizontal and is at 1, col is the colour, lty is line type, and lw
d is line width
```


Hazard Proportions of Treated Flies
Hazard Ratio
Wildtype
Knockdown
Wildtype
Knockdown
Dif
Rel