Advanced Research Methods - E7004

Day 2 - Inferential Statistics

Dr. Fabio Veronesi

20 March 2018

Summary

- Introduction
- Installing and importing packages
- Basics of inferential statistics
- Power Analysis
- Experimental designs
- t-test
- Sample size calculation

Introduction

After a general introduction to the R language, this lecture will start describing some important statistical techniques that are useful in Agriculture for running successful experiments.

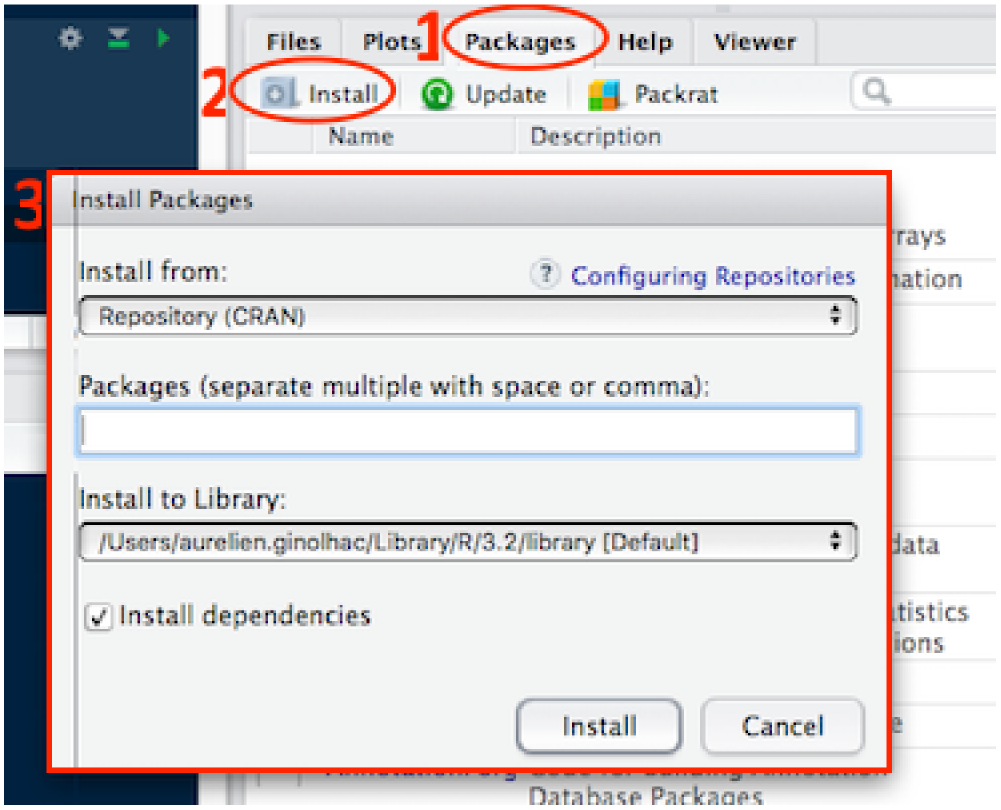
Installing and importing packages

The power of R comes from the fact that it is open-source, meaning that anyone can contribute by creating functions that researchers can freely use for their work. These functions are included in packages, which are add-on that allow us to use methods that are not normally included in the R version we download.

To install these packages we have two ways. We can use the function `install.packages`:

```r
install.packages("agricolae")
install.packages("pwr")
```

The second option is using the menu packages located on the lower right panel in R Studio:



Source: lsru.github.io

Once the package has been installed we can simply load it with the function `library`:

```r
library(agricolae)
library(pwr)
```

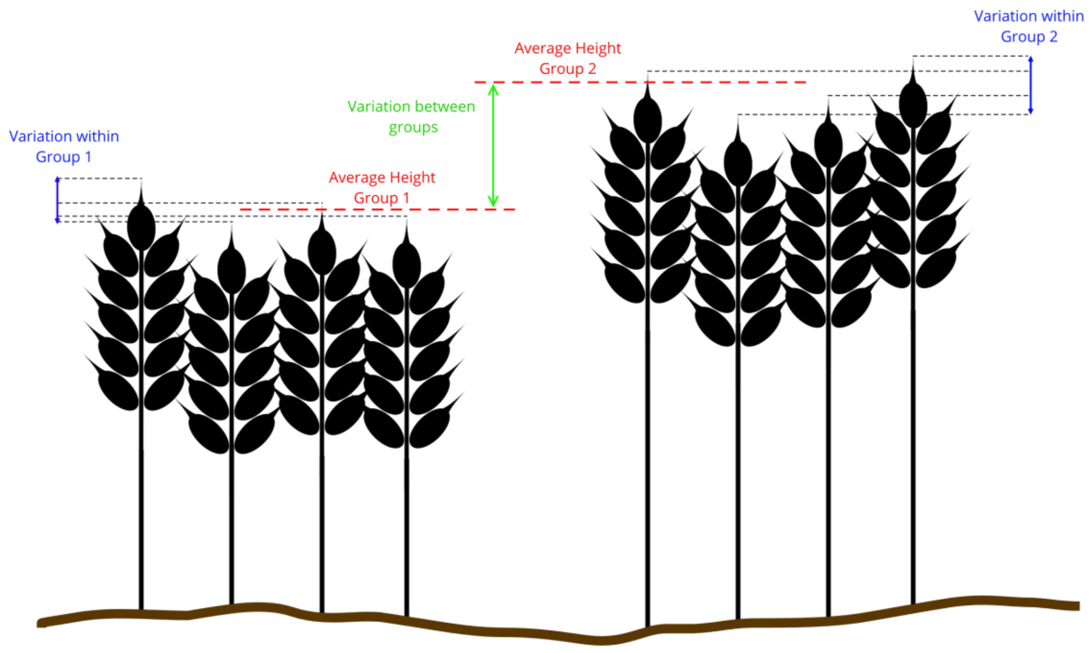
Basics of inferential statistics

In research, we are often interested in measuring the effect of some treatments on a certain variable. For example, we may be interested in understanding how nitrogen affects plant height. If we had unlimited time and resources, we would test different levels of nitrogen in many soil type and climates around the world, and see how these affect plant height. This way we could account for errors derived from different soil types, fertility levels, moisture content ect. However, time and resources are often limited, so this approach is not realistic. To overcome this limitation we need to design small scale experiment that would still allow us to draw conclusion that could be used in other fields (i.e. with other soil types, fertility levels, moisture content ect).

In statistical terms we say that we do not have access to the entire population, i.e. we do not have access to all the plants in the world to test our treatments. Since we are working on relatively small scale experiments, we only have access to relatively small samples.

However, the purpose of statistical analysis is to make important decisions about the general population, i.e. all plants in the world, based only on our relatively small samples.

To better understand these points we can look at the image below:



ANOVA Example

This image depicts a schematic representation of what could be the effect on plant height of two different treatments. As it is often the case in experimental science, even though plants belonging to the same group have received the same exact treatment (e.g. same level of nitrogen), they react differently. Some plants are smaller than others. This is the effect of natural variability, which may depend on local differences in soil conditions, genetic differences or other factors that we cannot account for in our experiment. However, if this natural variability is too high it may "mask" the real effect of treatments. In other words, we would not be able to successfully detect differences between groups, in cases when difference within groups are large.

The only way we have to reduce the impact of this random variation is to make sure we have enough samples to minimise the standard error of the mean (SEM). This is referred to as the power of the experiment. This is where replication becomes important. To reduce the SEM we need to increase the number of samples. However, since time and resources are limited we need to make sure we are not wasting them. So the number of samples need to be the lowest possible to obtain reliable results.

We can better understand this point with a simple simulation:

```
S1 = rnorm(n=3, mean=5, sd=2.5)
S2 = rnorm(n=3, mean=5, sd=2.5)
```

In this example we simulate two samples from a population with normal distribution, mean value equal to 5 and standard deviation of 2.5. Since they come from the same population we would expect them to be similar. However, mean values computed from three random samples may look very different. This is due to the large variation around the mean, which we cannot control as it may be driven by natural variability:

```
mean(S1); mean(S2)

## [1] 3.571634

## [1] 4.911484
```

With the same function we introduced in the previous lecture, we can compute standard error of the mean and confidence intervals:

```
SEM = function(x){sd(x) / sqrt(length(x))}

data.frame(Lower95 = c(mean(S1)-2*SEM(S1), mean(S2)-2*SEM(S2)),
           Mean = c(mean(S1), mean(S2)),
           Upper95 = c(mean(S1)+2*SEM(S1), mean(S2)+2*SEM(S2)))

##    Lower95     Mean  Upper95
## 1 3.039110 3.571634 4.104158
## 2 3.427614 4.911484 6.395355
```

Confidence intervals provide a way to assess the accuracy of the mean value we compute from our sample. In fact, the 95% confidence intervals we calculated (as twice the standard error) provide us a range within which we are 95% certain of finding the true value of the mean, i.e. the population mean. That is why we can use confidence intervals to compare groups.

As you can see, confidence intervals are very wide and they overlap. This indicates very clearly that the two samples may come from the same population. In this case, since we know they come from the same population, our conclusion is correct.

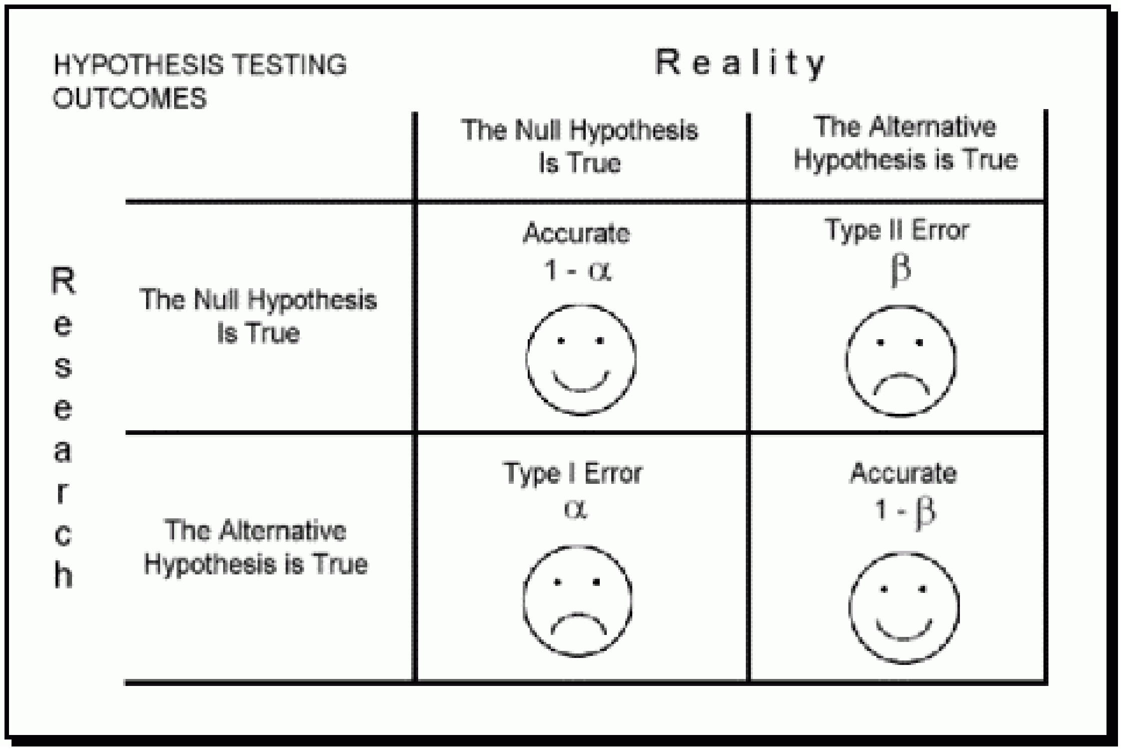
However, let’s look at another sample:

```
S3 = rnorm(n=3, mean=6.25, sd=2.5)

data.frame(Lower95 = c(mean(S1)-2*SEM(S1), mean(S3)-2*SEM(S3)),
           Mean = c(mean(S1), mean(S3)),
           Upper95 = c(mean(S1)+2*SEM(S1), mean(S3)+2*SEM(S3)))

##    Lower95     Mean  Upper95
## 1 3.039110 3.571634 4.104158
## 2 3.119913 4.079033 5.038153
```

Sample 3 is now coming from a population with mean equal to 6.25, which is half a standard deviation higher than the previous population. So we would expect to find differences between sample 1 and sample 3. However, once again the confidence intervals overlap, suggesting no differences. So in this case we are reaching a wrong conclusion, saying the two samples are the same when in fact they are different. In other words we have a false negative conclusion. This is because the sample size is too low and the confidence intervals are therefore very wide.

It may also happen, for example in cases where the distribution has high standard deviation compared to the mean, that even though samples come from the same population, confidence intervals barely overlap or not overlap at all. This is possible, and will make us erroneously conclude the the two samples are different. This is known as a false positive.

In statistical terms, inferential statistics tests a null hypothesis, which assumes our treatments have no effect (i.e. samples come from the same population), against an alternative hypothesis, in which we try to demonstrate the effectiveness of our treatments (i.e. samples come from two distinct populations). The result is a probability (*p-value*), which allows us to either accept or reject the null hypothesis. However, to make sure we are not obtaining false positive or false negative outcomes we need to consider two quantities:



HYPOTHESIS TESTING OUTCOMES

		Reality	
		The Null Hypothesis Is True	The Alternative Hypothesis is True
Research	The Null Hypothesis Is True	Accurate $1 - \alpha$ ☺	Type II Error β ☹
	The Alternative Hypothesis is True	Type I Error α ☹	Accurate $1 - \beta$ ☺

Source: datasciencedojo.com

The probability of incurring in a type I error (false positive) is indicated by α (or significance level) and usually takes a value of 5%; this means that we are happy to consider a scenario where we have 5% chances of rejecting the null hypothesis when it is actually true. If we are not happy with this, we can further decrease this probability by decreasing the value of α (for example to 1%). On the contrary the probability of incurring in a type II error (false negative) is expressed by β, which usually takes a value of 20%

(meaning a power of 80%). This means we are happy to work assuming that we have a 20% chance of accepting the null hypothesis when it is actually false. If our experiments are not designed properly we cannot be sure whether we actually incurred in one of these two errors. In other words, if we run a bad experiment and we obtain an insignificant *p-value* it may be that we incurred in a type II error, meaning that in reality our treatment works but its effect cannot be detected by the experiment. However, it may also be that we obtained a significant *p-value* but we incurred in a type I error, and if we repeat the experiment we will find different results.

In summary, designing experiments requires us to think very carefully about all the aspects we mentioned above. In particular, there are three basic principles that we need to consider when designing good and robust experiments:

- **Randomization**: this is the most important step in every experiment. Randomizing treatments guarantee that the conclusions we make are unbiased. For example, let's say we are testing nitrogen levels in a field that may be affected by changes in soil fertility (which are unknown to us). If we provide high nitrogen to plants only in one part of the field we may end up concluding that this has an effect on plant height only because the soil in that part of the field was more fertile. If we randomize treatments across the field we effectively account for different soil conditions.
- **Blocking**: this is another tool we can use to minimise the error of our experiment. Let's go back to our previous example of a field experiment. If we know that a particular area of our field has more fertile soils, we can partition our study area and create two blocks. The idea is that we can better control the error that different soil conditions may cause (i.e. we effectively minimise the within groups variation), by effectively design two sub-experiments in one. If we use blocks, we need to randomize treatments within each block, as if they were separate experiments.
- **Replication**: randomization and blocking are two powerful ways to control for natural variation. However, the reality is we can never fully minimise it so we need to find ways to reduce it in our samples. We can achieve this by working with larger samples, thus reducing confidence intervals around the mean and allowing the statistical tests to better detect treatment effects.

Experimental designs

The selection of the design most suitable for the experiment we have in mind is of crucial important. Several factors needs to be considered before we can decide an experimental design. However, there are some designs that are more commonly used that other. For example, in field experiments we often find either complete randomized designs, or complete block designs. While in animal studies the latin square is the most common.

All these designs respect the basic principles outline above, but allow us to make sure the experiment is optimised for our treatment structure and for any practical constrain we may have, e.g. lack of space.

Complete randomized design (CRD)

This is probably the most common design, and it is generally used when conditions are uniform, so we do not need to account for local variations due for example to soil conditions.

The package agricolae offers several functions to cover all the most important experimental designs.

First of all, we need to specify our treatment structure:

```
Treatment = c("A","B","Control")
```

Then we can use the function design.crd to create a complete randomized design:

```
CRD = design.crd(trt=Treatment, r=3)
```

We need to specify two options: trt, for treatment, and r for the number of replicates.

The results is a complex object:

```
CRD

## $parameters
## $parameters$design
## [1] "crd"
##
## $parameters$trt
## [1] "A"       "B"       "Control"
##
## $parameters$r
## [1] 3 3 3
##
## $parameters$serie
## [1] 2
##
## $parameters$seed
## [1] 419958237
##
## $parameters$kinds
## [1] "Super-Duper"
##
## $parameters[[7]]
## [1] TRUE
##
##
## $book
##   plots r Treatment
## 1   101 1         A
## 2   102 1   Control
## 3   103 2         A
## 4   104 1         B
## 5   105 2   Control
## 6   106 3   Control
## 7   107 3         A
## 8   108 2         B
## 9   109 3         B
```

We are interested in the last element of the object `CRD`:

```
CRD$book

##   plots r Treatment
## 1   101 1         A
## 2   102 1   Control
## 3   103 2         A
## 4   104 1         B
## 5   105 2   Control
## 6   106 3   Control
## 7   107 3         A
## 8   108 2         B
## 9   109 3         B
```

This is the table we can use to establish our design in the field, using the plot IDs to make sure we use the right treatment in the right place.

One thing we need to remember in the package `agricolae` is that we can only provide a vector with our treatment structure. However, if our experiment is complex we need some more lines of code to plan it properly.

Let's look at the following lines:

```
Trt1 = c("A","B","C")
Trt2 = c("1","2")
Trt3 = c("+","-")

TRT.tmp = as.vector(sapply(Trt1, function(x){paste0(x,Trt2)}))
TRT = as.vector(sapply(TRT.tmp, function(x){paste0(x,Trt3)}))
TRT.Control = c(TRT, rep("Control", 2))
```

In this example we are planning an experiment with three levels of treatment. This means for example that treatment `A`, can be split into `A1` and `A2`, and these can be further divided into `A1+` and `A1-`. The same is true for `B` and `C`. To complicate things we are also including Control, which needs to be separated from the other treatments.

The function `sapply` allows us to iterate through the vector `Tr1` and pasting (with the function`paste`) the elements of vector `Tr2`. Let's see what is the result of the first `sapply` call:

```
TRT.tmp

## [1] "A1" "A2" "B1" "B2" "C1" "C2"
```

Then we do the same process again to include the third level of treatment:

```
TRT

##  [1] "A1+" "A1-" "A2+" "A2-" "B1+" "B1-" "B2+" "B2-" "C1+" "C1-" "C2+"
## [12] "C2-"
```

Since we are also planning to have a control, so we need an additional line where we add two plots where no treatment will be applied:

```
TRT.Control

##  [1] "A1+"     "A1-"     "A2+"     "A2-"     "B1+"     "B1-"     "B2+"    
##  [8] "B2-"     "C1+"     "C1-"     "C2+"     "C2-"     "Control" "Control"
```

The vector `TRT.Control` has now all the elements we need to design our experiment, again with 3 replicates:

```
design.crd(trt=TRT.Control, r=3)$book

##    plots r TRT.Control
## 1    101 1         B1+
## 2    102 1         B2+
## 3    103 1     Control
## 4    104 1         A2-
## 5    105 1         C1+
## 6    106 1         A1-
## 7    107 2         B1+
## 8    108 1         A1+
## 9    109 1         A2+
## 10   110 2     Control
## 11   111 2         A2+
## 12   112 1         B1-
## 13   113 2         A2-
## 14   114 2         A1-
## 15   115 1         C1-
## 16   116 3     Control
## 17   117 2         C1+
## 18   118 1         B2-
## 19   119 1         C2-
## 20   120 2         B2-
## 21   121 2         B2+
## 22   122 3         B2-
## 23   123 1     Control
## 24   124 2     Control
## 25   125 2         C2-
## 26   126 3         A1-
## 27   127 3         C1+
## 28   128 2         A1+
## 29   129 1         C2+
## 30   130 2         C1-
## 31   131 3         A1+
## 32   132 2         C2+
## 33   133 3         C1-
## 34   134 2         B1-
## 35   135 3         B1-
## 36   136 3         B1+
## 37   137 3     Control
## 38   138 3         A2-
## 39   139 3         C2+
## 40   140 3         B2+
## 41   141 3         A2+
## 42   142 3         C2-
```

Complete block design

The exact same approach can be used with all other designs available in the package `agricolae`:

```
CBD = design.rcbd(trt=TRT.Control, r=3)
```

For more complex designs, the function create a sketch that we can print for easier visualisation:

```
print(CBD$sketch)

##      [,1]      [,2] [,3]      [,4] [,5]      [,6]      [,7] [,8] [,9]
## [1,] "A2+"     "B2-" "C1-"     "C2-" "A2-"     "Control" "B1+" "B2+" "B1-"
## [2,] "A2+"     "B2-" "C2-"     "C1-" "Control" "A2-"     "C1+" "A1+" "A1-"
## [3,] "Control" "B2-" "Control" "B1+" "A2-"     "A1+"     "C1+" "B1-" "C1-"
##      [,10] [,11]     [,12] [,13] [,14]
## [1,] "A1-" "C1+"     "C2+" "A1+" "Control"
## [2,] "B1-" "Control" "B1+" "C2+" "B2+"
## [3,] "B2+" "A1-"     "C2+" "C2-" "A2+"
```

Incomplete Block Design

Incomplete block designs are useful when our treatment structure is difficult to organise into a balanced design. For example, let's say we have two treatment levels, and one control:

```
Trt1 = c("A","B","C")
Trt2 = c("1","2")

TRT = as.vector(sapply(Trt1, function(x){paste0(x,Trt2)}))
TRT.Control = c(TRT, rep("Control", 1))
TRT.Control
```

```
## [1] "A1"      "A2"      "B1"      "B2"      "C1"      "C2"      "Control"
```

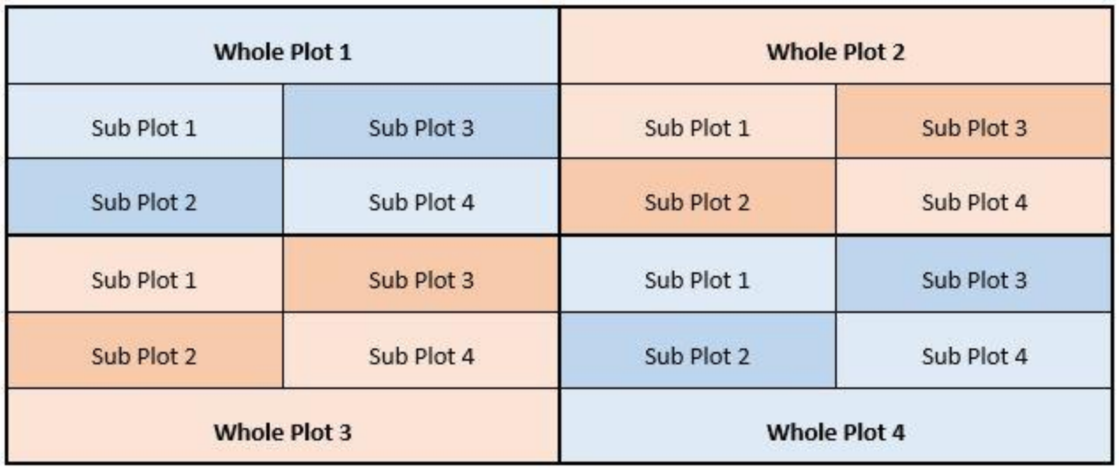
AS you can see, here we have a total of 7 treatment combinations, which makes it difficult to create a balanced design. Clearly we could include an additional Control and make it a set of 8 treatments. However, there may be cases where this is not possible.

In such cases we can use the incomplete block design:

```
BIB = design.bib(trt=TRT.Control, r=2, k=7)

##
## Parameters BIB
## ==============
## Lambda     : 2
## treatmeans : 7
## Block size : 7
## Blocks     : 2
## Replication: 2
##
## Efficiency factor 1
##
## <<< Book >>>

print(BIB$sketch)

##      [,1] [,2] [,3] [,4] [,5]      [,6] [,7]
## [1,] "A1" "A2" "C1" "C2" "B2"      "B1" "Control"
## [2,] "C1" "B1" "A1" "C2" "Control" "B2" "A2"
```

Split-Plot Design

In some experiments we may have particular treatments, e.g. fertilizers or irrigation regimes, which can only be applied to relatively large areas. While other treatments, e.g. seed variety, are applied to smaller areas. This creates a sort of nested design, which can be better understood by looking at the image below:



Source: http://blog.minitab.com

As you can see the area is divided into four whole-plots, with the blue areas being for example irrigated differently compared to the red areas. Then, each whole plot is further divided into sub-plots, where another treatment is applied and randomized.

In R we can design split-plots with the following line:

```
SpPl = design.split(Trt1, Trt2, r=3, design=c("crd"))
```

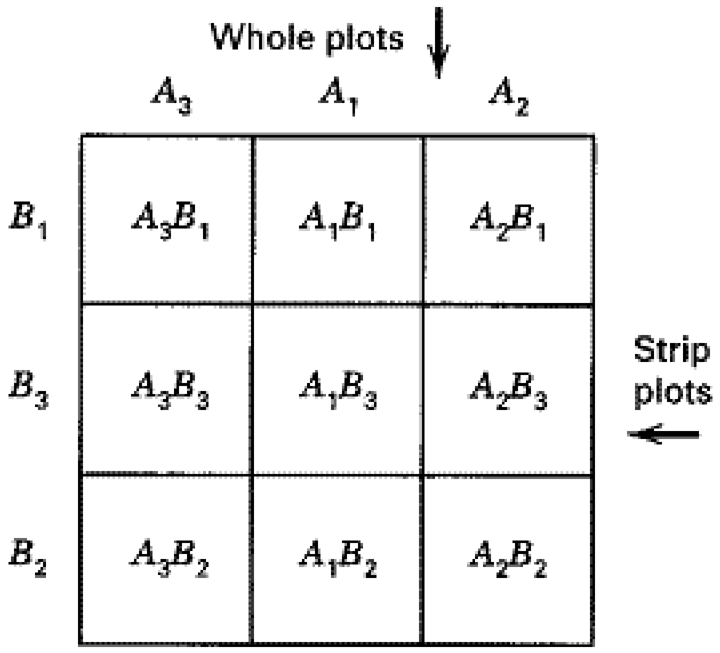
Within the function `design.split` we can include two treatments, the first for whole-plots and the second for sub-plots. Then we can also include the option `design`, in which we can select: `rcbd`, for blocks; `crd`, for complete random and `lsd` for latin square (which we will describe below).

As always, we can look at the design with:

```
SpPl$book

##    plots splots r Trt1 Trt2
## 1    101      1 1    A    1
## 2    101      2 1    A    2
## 3    102      1 2    A    1
## 4    102      2 2    A    2
## 5    103      1 1    B    1
## 6    103      2 1    B    2
## 7    104      1 1    C    1
## 8    104      2 1    C    2
## 9    105      1 2    C    1
## 10   105      2 2    C    2
## 11   106      1 3    C    2
## 12   106      2 3    C    1
## 13   107      1 2    B    1
## 14   107      2 2    B    2
## 15   108      1 3    A    2
## 16   108      2 3    A    1
## 17    109       1 3     B     2
## 18    109       2 3     B     1
```

Strip-Plot Design

This design is somewhat similar to split-plots. The main difference is that whole-plots and sub-plots are arranged in strips. Let's look at the image below:



Source: onlinecourses.science.psu.edu

As you can see we have three vertical strips, once for each treatment (A1, A2 and A3), and three horizontal strips (for treatments B1, B2 and B3).

The code for creating strip-plot design sin R is again very simple:

```
Trt1 = c("A1","A2","A3")
Trt2 = c("B1","B2","B3")

StPl = design.strip(Trt1, Trt2, r=3)

StPl$book

##    plots block Trt1 Trt2
## 1    101     1   A2   B2
## 2    102     1   A2   B3
## 3    103     1   A2   B1
## 4    104     1   A3   B2
## 5    105     1   A3   B3
## 6    106     1   A3   B1
## 7     107      1    A1    B2
## 8     108      1    A1    B3
## 9     109      1    A1    B1
## 10    201      2    A2    B2
## 11    202      2    A2    B3
## 12    203      2    A2    B1
## 13    204      2    A3    B2
## 14    205      2    A3    B3
## 15    206      2    A3    B1
## 16    207      2    A1    B2
## 17    208      2    A1    B3
## 18    209      2    A1    B1
## 19    301      3    A3    B2
## 20    302      3    A3    B3
## 21    303      3    A3    B1
## 22    304      3    A2    B2
## 23    305      3    A2    B3
## 24    306      3    A2    B1
## 25    307      3    A1    B2
## 26    308      3    A1    B3
## 27    309      3    A1    B1
```

Latin Square Design

Latin square designs are very popular in animal studies, where we need to limit as much as possible the number of subjects involved. This design allows each subject to receive all treatments, and still allows to reach good experimental power. In `agricolae`, latin square designs can be created as follows:

```
LS = design.lsd(trt=TRT)

LS$sketch
```

```
##      [,1] [,2] [,3] [,4] [,5] [,6]
## [1,] "B2" "B1" "C1" "A2" "A1" "C2"
## [2,] "C2" "C1" "A1" "B2" "B1" "A2"
## [3,] "A1" "C2" "A2" "C1" "B2" "B1"
## [4,] "B1" "A2" "B2" "A1" "C2" "C1"
## [5,] "A2" "A1" "B1" "C2" "C1" "B2"
## [6,] "C1" "B2" "C2" "B1" "A2" "A1"
```

This is a 6x6 latin square.

Please look at the package description to know what other experimental designs are avilable in `agricolae`.

---

t-test

Now that we covered the basics of experimental designs we can start exploring the statistical tests that we can use to detect differences between treatments. The first test we are going to look at is the t-test, which allows us to compare two samples and check whether they are statistically different (i.e. come from two populations). In essence, the t-test does exactly what we described above, takes two samples and computes the probability (*p-value*) that the null hypothesis is true.

Let's see how to perform a t-test in R using the samples we created above:

```r
t.test(S1, S3, alternative="two.sided")
```

```
##
##  Welch Two Sample t-test
##
## data:  S1 and S3
## t = -0.92503, df = 3.1261, p-value = 0.4207
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  -2.213873  1.199075
## sample estimates:
## mean of x mean of y
##  3.571634  4.079033
```

The t-test checks the two samples under the hypothesis that they come from two different distributions. As you can see we have specified the alternative hypothesis, `two.sided`, which means test whether `S1` and `S2` come from two populations. The test returns several outputs, and what we are mostly interested in is the *p-vaue*, which tells whether we can reject the null hypothesis.

The decision of rejecting the null hypothesis is something we can do based on the significance level we want to achieve. Significance level is the α we discussed above, and indicates the probability of rejecting the null hypothesis when it is actually true (false positive). Usually, we consider a statistical test significant only if the *p-value* is below 0.05, meaning that the chances of getting a false positive are 5%.

As mentioned before, after properly accounting for sources of error by creating good experimental designs, the only way we have to ensure the statistical test can detect differences between samples is increasing their size. Larger samples have lower standard error of the mean, and therefore are less variable. However, since we are often constrained by space or time, we need to make sure we do not design experiments that are too complex to run. Therefore the sample size needs to be large enough to get robust confidence intervals, but small enough so that we can run the experiment smoothly.

Sample size

For all the reasons explained above, we have now the issue of finding the optimal number of samples that would allow us to detect a difference between our treatments, or be large enough to avoid any false negative errors. The procedure to compute the optimal sample size is referred to as **Power Analysis**.

Power analysis is based on the concept of **Effect Size** which is defined as the standardized mean differences between groups. In other words, the effect size measures the magnitude of the differences between groups (length of green arrow in ANOVA example image). There are several ways to compute the effect size, but the most common is the following:

$$ES = d = \frac{\bar{y}_B - \bar{y}_A}{SD_{pooled}}$$

Source: *http://r-video-tutorial.blogspot.co.uk*

where ES is the effect size (often referred to as Cohen's d). In this equation, $\bar{y}_A$ is the mean for treatment A, and $\bar{y}_B$ is the mean for treatment B. The denominator is the pooled standard deviation, which is computed as follows:

$$SD_{Pooled} = \sqrt{\frac{(n_B - 1)SD_B^2 + (n_A - 1)SD_A^2}{n_B + n_A - 2}}$$

Source: *http://r-video-tutorial.blogspot.co.uk*

where SD are the standard deviations for treatments B and A, and n is the sample size.

In R we can easily compute the effect size for `S1` and `S3` as follows:

```
numerator = (mean(S3)-mean(S1))
denominator = sqrt((((length(S3)-1)*sd(S3)^2)+((length(S1)-
1)*sd(S1)^2))/(length(S3)+length(S1)-2))

ES = numerator/denominator
ES

## [1] 0.7552873
```

Cohen's d is very easy to interpret, because it describes differences between groups based on their standard deviation. An effect size ES equal 1, would imply that the means for the two groups are separated by one full standard deviation; while ES equal 0.5 implies the two values are half a standard deviation away. As you remember, samples `S1` and `S3` come from two populations that have mean values separated by half a standard deviation. However, since we are simulating only three values the ES may change dramatically between simulations.

Now that we know the effect size for our sample, we can compute the sample size we would require to have a robust experiment, using the function `pwr.t.test` in the package `pwr`: In this function we need to specify several options:

- Effect size, option `d`: this is what we calculated above so we can just use the object `ES`
- Alpha, option `sig.level`: here we need to specify the significance level we are happy to consider. Typically a value of 0.05 is accepted, meaning a 5% chances of getting false positive results.
- Beta, option `power`: here we need to specify the power we want to achieve. Typically a value of 0.8 is accepted, meaning that 80% chances of not getting false negative outcomes.
- Type and alternative depends on the test, in this case we have two samples and we want to test whether they are different.

```
PW.T = pwr.t.test(d = ES, sig.level = 0.05, power = 0.8, type = "two.sample",
alternative = "two.sided")

PW.T

##
##      Two-sample t test power calculation
##
##               n = 28.51068
##               d = 0.7552873
##       sig.level = 0.05
##           power = 0.8
##     alternative = two.sided
##
## NOTE: n is number in *each* group
```

Again, results vary each time we run the simulation. However, it is clear that we would need a lot more replicates to have a robust experiment. We can use the sample size `n` computed above to run a new simulation. We can extract the sample size from the object `PW.T`, but it nees to be rounded up (example, 24.5 become 25). We can do that with the function ceiling:

```
SampleSize = ceiling(PW.T$n)

SampleSize

## [1] 29
```

Now we can input the sample size into a new simulation and run the t-test again:

```r
S4 = rnorm(n=SampleSize, mean=5, sd=2.5)
S5 = rnorm(n=SampleSize, mean=6.25, sd=2.5)

t.test(S4, S5, alternative="two.sided")
```

```
##
##  Welch Two Sample t-test
##
## data:  S4 and S5
## t = -3.0965, df = 55.628, p-value = 0.003066
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  -3.5180199 -0.7539232
## sample estimates:
## mean of x mean of y
##  4.862332  6.998303
```

Results should now be significant, i.e. *p-value* below 0.05. However, please remember that around 5% of the times results will still not result significant. This is because we set a significant level at 5%.

A Priori Calculation

In the previous example, we computed the effect size based on samples we simulated. However, the large majority of times we do not have access to data we can use to compute the effect size. So the question becomes, how can we compute the number of samples if we do not know the effect size?

We have two ways of approaching this issue. The first is perform a pilot study, intended as a small scale experiment where we test our treatments. With a pilot study we are not really interested in sample size, we just want to look at average differences between groups.

The other way is looking at the literature, where it is suggested to use a medium effect size every time we do not have direct measures. A medium effect size corresponds to `ES = 0.5`.

Conclusions

In this lecture we introduced some very important concepts that form the basis of inferential statistics. We learned how to design robust experiments and compute the required sample size for t-tests.

References

Power Analysis

Please refer to my blog to learn more about power analysis: Power Analysis

For the large majority of the power analysis I perform, I use the free software G*Power developed by the University of Düsseldorf. It is simple to use but very effective!

Additional resources for power analysis are:

- "Statistics Done Wrong" by Alex Reinhart: LINK
- "Power Analysis for Experimental Research" by Bausell & Li
- "Statistical Power Analysis for the Behavioral Sciences" by Cohen

Experimental desings

Design of Experiments (Penn State) Statistical Methods for Bioscience (Wisconsin-Madison)

Homework

1. Using once again the Diet dataset, please perform a t-test that compares the weight loss after 6 weeks between gender. Comment your results.

2. Compute the effect size between genders.

3. Using the `pwr.t.test` try to perform an *a posteriori* power analysis. In this case we know the sample size (n) and we need to compute power, so `power=NULL`.