Using permutation and random independent sampling

Contents

Permutation tests: principles

The idea behind random permutation is the following: if two variables are really correlated, then mixing up the data in one of the variables (*permuting* the variable) should break this relationship. If they're not really correlated, then permuting a variable should not have a big effect on the statistic. Put another way, measuring the statistic when one variable is permuted gives us an idea of the baseline level of chance.

R has a function called `sample` which is useful for permutation. It takes one manditory argument: the vector of items to choose from, and an optional argument of how many items to choose (default is as many as in the original vector) and whether to allow choosing the same item twice (default is no). It then returns a random selection of the items in the vector.

With the default settings, it *permutes* the vector

```r
x <- c(1,2,3,4,5,6)
sample(x)
```

```
## [1] 1 6 4 3 5 2
```

Note that, because this is a random process, if we do it again, we get a different answer:

```r
sample(x)
```

```
## [1] 1 2 3 5 6 4
```

Let’s try a simple test. If we pick two sets of random numbers, they should not be correlated:

```r
x <- rnorm(100)
y <- rnorm(100)

cor.test(x,y)
```

```
##
##  Pearson's product-moment correlation
##
## data:  x and y
## t = 0.63809, df = 98, p-value = 0.5249
## alternative hypothesis: true correlation is not equal to 0
## 95 percent confidence interval:
##  -0.1337854  0.2574879
## sample estimates:
##        cor
## 0.06432298
```

They’re not significantly correlated - good! But we can also test this by permuting one of the variables:

```r
y.permuted <- sample(y)

cor.test(x,y.permuted)
```

```
##
##  Pearson's product-moment correlation
##
## data:  x and y.permuted
## t = -0.85863, df = 98, p-value = 0.3926
## alternative hypothesis: true correlation is not equal to 0
## 95 percent confidence interval:
##  -0.2781087  0.1119067
## sample estimates:
##         cor
## -0.08641079
```

We get a different result, but it’s still not significant. What happens if we do this many times? To do this, let’s put the code above into a new custom function. We’ll have it return just the correlation coefficient rather than the whole test.

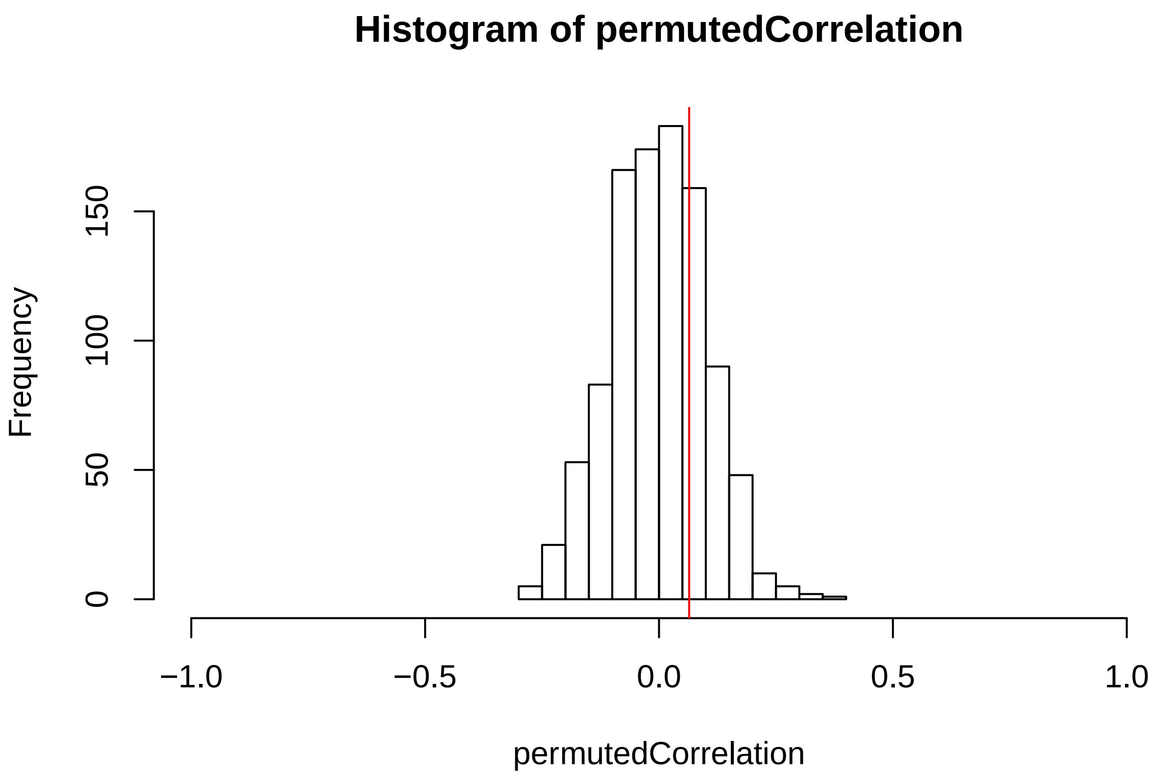
```r
permutationTest <- function(x,y){
  y.permuted <- sample(y)
  r = cor(x,y.permuted)
  return(r)
}
```

Then we’ll call this custom function 100 times using the function `replicate`:

```r
permutedCorrelation <- replicate(1000, permutationTest(x,y))
```

Now we can look at the results using a histogram. We'll also add a red line to mark the position of the real correlation coefficient value, so we can compare them. (Note that we're adjusting the limits of the x axis to show more of the plot).

```r
realCorrelation <- cor(x,y)

hist(permutedCorrelation, xlim=c(-1, 1))
abline(v=realCorrelation, col=2)
```



This looks like a normal distribution centered around zero. In a few cases, the correlation is quite high, but it's quite rate. That's what we expected! This is a representation of the likelihood of the two variables being correlated by chance.

The red line is the real correlation coefficiet - and it sits in the middle of this distribution. This means that we're quite likely to see the real correlation value, even by chance.

We can work out a statistic to tell us how strong the relationship is. The z score measures how many standard deviations away from the mean of permuted values the true value is. We can also work out a p-value by counting the proportion of permutations that resulted in a stronger correlation coefficient than the real one.

Note that below, I'm not concerned with the direction of the correlation, so I'm just looking at the *absolute* size of the value (how far it is from zero), rather than positive/negative. In other words, it's a 2-tailed test. The function `abs` turns a vector of numbers into a vector of absolute values.

```r
# number of standard deviations above the mean
z.score <- (realCorrelation-mean(permutedCorrelation)) / sd(permutedCorrelation)
z.score
```

```
## [1] 0.6537095
```

```
# proportion of stronger results in permutation
p.value <- sum(abs(permutedCorrelation) >= abs(realCorrelation)) / length(permutedCorrelation)
p.value
```

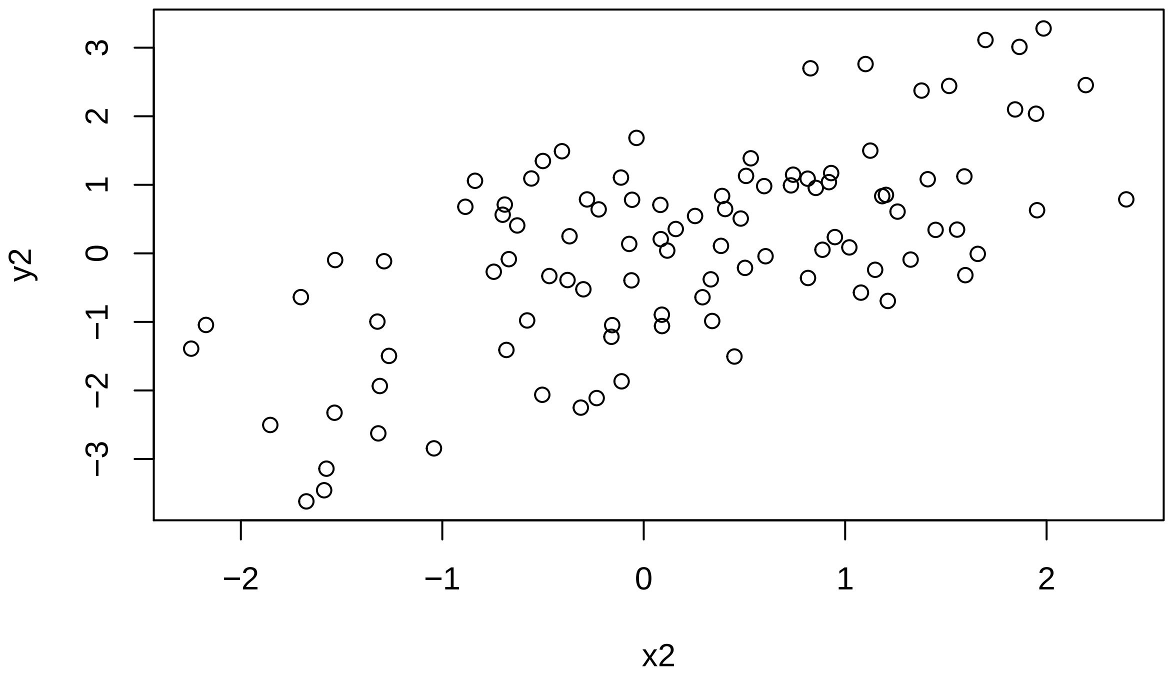
```
## [1] 0.55
```

That is, the strength of the correlation is very weak (z = 0.6537095, p = 0.55).

Correlated variables

Let's do another test - this time with two variables that *are* correlated. We'll do this by picking random numbers for *x2*, then copying them, slightly altered for *y2*:

```
x2 <- rnorm(100)
y2 <- jitter(x2, amount=2)
plot(x2,y2)
```



```
cor.test(x2,y2)
```

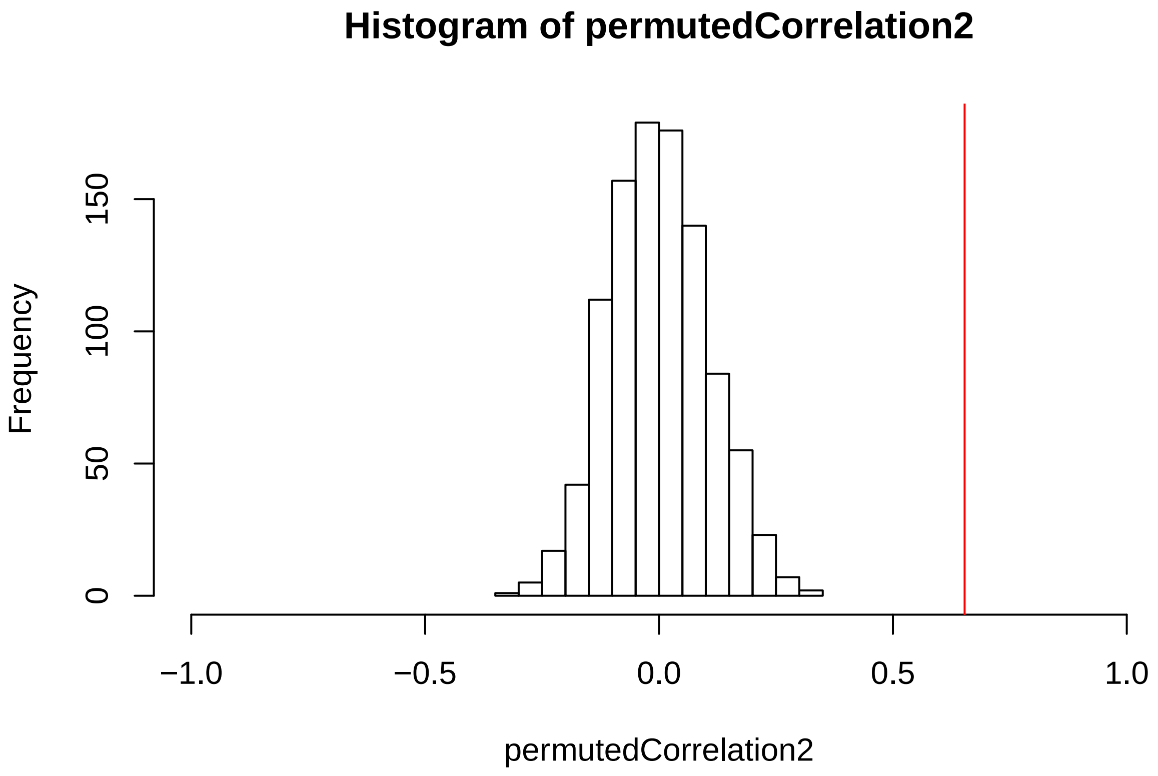
```
##
##  Pearson's product-moment correlation
##
## data:  x2 and y2
## t = 8.5426, df = 98, p-value = 1.741e-13
## alternative hypothesis: true correlation is not equal to 0
## 95 percent confidence interval:
##  0.5241605 0.7530952
## sample estimates:
##       cor
## 0.6533166
```

Next we'll work out the real correlation coefficient, and then the permutation test.

```
realCorrelation2 <- cor(x2,y2)
permutedCorrelation2 <- replicate(1000, permutationTest(x2,y2))
```

Now we can look at the results using a histogram, but we'll also add a red line to mark the position of the real correlation coefficient value, so we can compare them. (Note that we're adjusting the limits of the x axis to show more of the plot).

```
hist(permutedCorrelation2, xlim=c(-1, 1))
abline(v=realCorrelation2, col=2)
```



The real value is clearly outside the distribution of permuted values, which suggests that the two variables are correlated to a greater extent than we'd expect by chance. Here are the statistics:

```
z.score2 <- (realCorrelation2-mean(permutedCorrelation2)) / sd(permutedCorrelation2)
z.score2
```

```
## [1] 6.195499
```

```
# proportion of stronger results in permutation
p.value2 <- sum(abs(permutedCorrelation2) >= abs(realCorrelation2)) / length(permutedCorrelation2)
p.value2
```

```
## [1] 0
```

The p-value is zero, but this is not really a good description of the test. What we found was that, out of 1000 permutations, none were as strong as the real value. So, we could say that the p-value is lower than 1/1000, i.e. we would report "$p < 0.001$". The greater the number of permutations, the lower this p value can become.

Summary

We ran some permutation tests above. The basic steps were:

- Measure the **real** strength of the relationship between our variables
- Permute one of the variables to try to break this relationship
- Measure the strength of the relationship between the first variable and the permuted variable
- Do the permutation many times, and get a distribution of permuted strengths
- Compare the real strength of the relationship with the distribution of permuted strengths

Look again at the results for the uncorrelated variables:

- Pearson test t = 0.6380864, p = 0.5249057
- Permutation test z = 0.6537095, p = 0.55

Note that the measures are very similar. In fact, this is because the correlation test works on essentially the same principle - calculating the likelihood of observing a correlation as strong as the real one by chance. We used 1000 permutations for the test above. As the number of permutations increases, the similarity between the two tests should converge. In this case the similarity is also due to the fact that the variables were normally distributed, which is one of the assumptions of the Pearson correlation test. **The advantage of the permutation test is that it does not require that the variables are normally distributed**.

Another strength of the permutation test is that it can be modified to look at different baselines, for example permuting only within a linguistic area or family. We'll try this in the next sections . . .

An example with real data: testing differences between groups

In this tutorial, we'll investigate a correlation in our data. The hypothesis is the following:

> VSO languages have longer names than SOV languages. For example, the languages "Ik" and "So" are both VSO languages, while "Xaasongaxango" is a SOV language.

It's a stupid hypothesis, but will help demonstrate some principles.

First we load and merge our data, then filter out all except languages with VOS or SOV.

```r
# set working directory
setwd("~/Documents/Teaching/JenaSpringSchool/org/spring-school/IntroToR")
# load data
d <- read.csv("data/WALS_WordOrder.csv", stringsAsFactors = F)
glottoData <- read.csv("data/Glottolog_Data.csv", stringsAsFactors = F)
# merge two data frames
d <- merge(d, glottoData, by.x='glottocode', by.y='glotto.code')

# only VSO or SOV languages
d2 <- d[d$BasicWordOrder %in% c("VSO","SOV"),]

# get rid of NA values
d2 <- d2[complete.cases(d2),]
```

Next we have to convert the *BasicWordOrder* variable to a **factor**. This is a data type which recognises that the items represent discrete categories, rather than continuous numbers or a string of characters.

```
# convert to factors
d2$BasicWordOrder = factor(d2$BasicWordOrder)
head(d2$BasicWordOrder)
```

```
## [1] SOV SOV SOV VSO SOV SOV
## Levels: SOV VSO
```

Next we want our dependent measure - the length of language names:

```
d2$Name.length <- nchar(d2$Name)
```

So, do VSO languages have longer names?

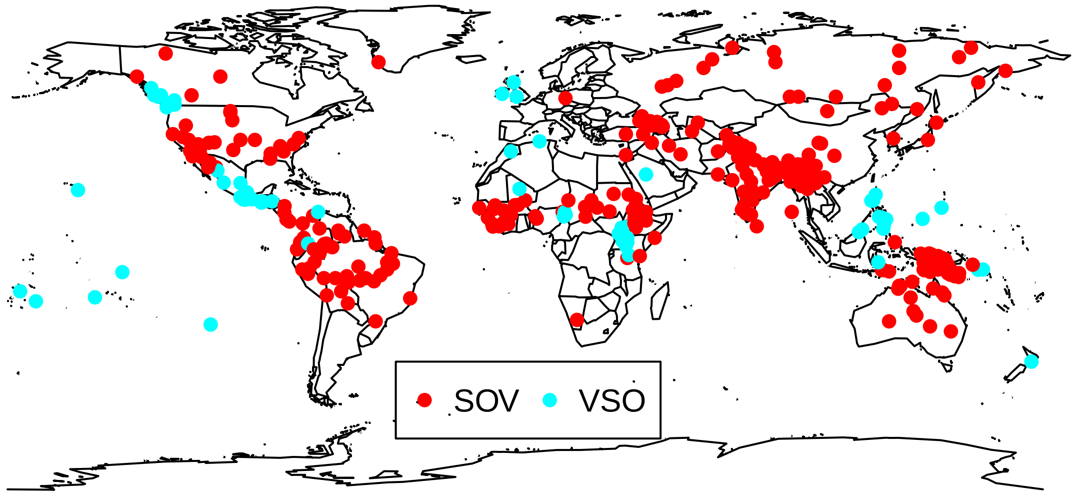
```
t.test(d2$Name.length~d2$BasicWordOrder)
```

```
##
##  Welch Two Sample t-test
##
## data:  d2$Name.length by d2$BasicWordOrder
## t = -3.5424, df = 90.792, p-value = 0.0006283
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  -4.079828 -1.148195
## sample estimates:
## mean in group SOV mean in group VSO
##          7.359673          9.973684
```

The pattern certainly seems strong - but we can't see whether the languages are related. Indeed, when we draw a map of the basic word order data, we notice that there are large geogrpahic areas of similar types:

```
## Warning: package 'maps' was built under R version 3.2.3
```

```
##
##  # maps v3.1: updated 'world': all lakes moved to separate new #
##  # 'lakes' database. Type '?world' or 'news(package="maps")'.  #
```



We can use several methods to try and control for the geographic relatedness of languages.

Permutation tests

Recall that a basic step in the permutation test is measuring the strength of the relationship between two variables. In the section above with two continuous variables, we used the correlation coefficient. In this section, our prediction is that there is a difference in the means of the two groups.

So let's measure the real difference in means between SVO and VSO languages:

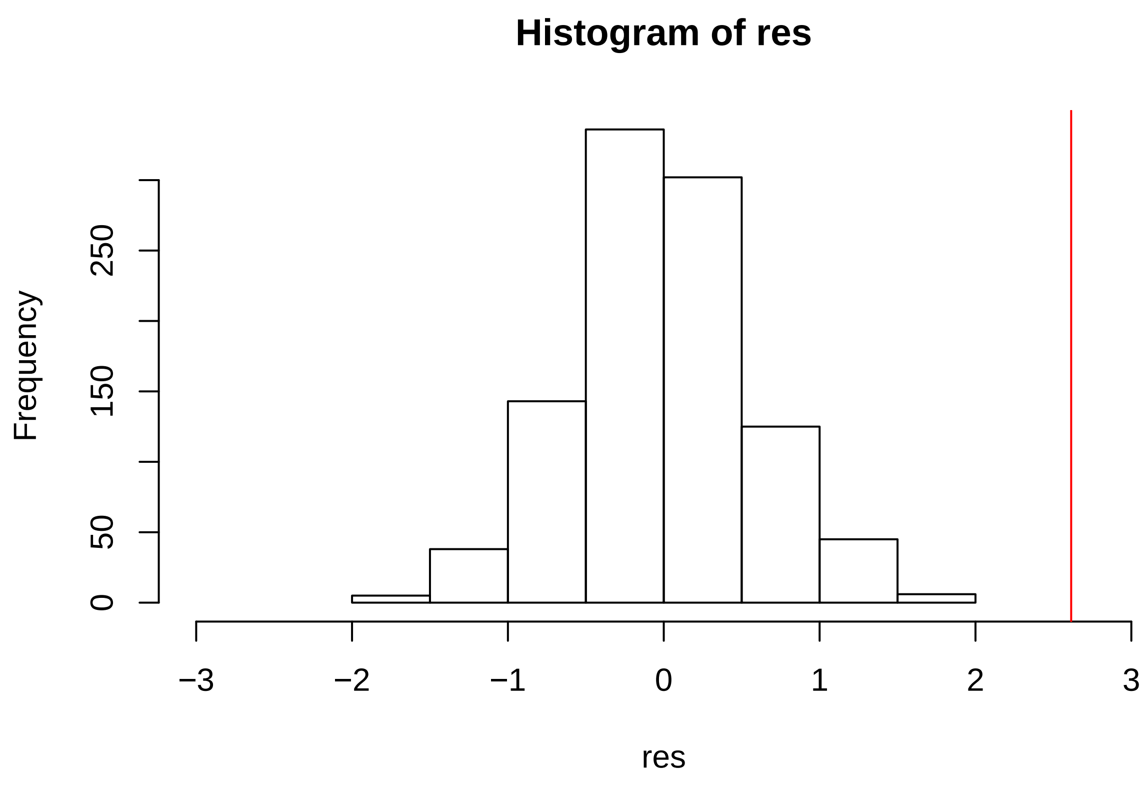
```
# function that takes a vector of numbers and a vector of what group each number belongs to, and works
getDifferenceInMeans <- function(x,group){
  diff(tapply(x, group, mean))
}

realDifferenceInMeans <- getDifferenceInMeans(d2$Name.length, d2$BasicWordOrder)
```

Now, in a similar manner as above, we can make a function that permutes the basic word order between languages:

```
permTest <- function(x,group){
  # permute group
  group <- sample(group)
  # work out difference in means
  permutedDifferenceInMeans <-getDifferenceInMeans(x,group)
  # return the value
  return(permutedDifferenceInMeans)
}
```

And now run this many times, and look at the results:

```
res <- replicate(1000, permTest(d2$Name.length,d2$BasicWordOrder))

hist(res, xlim=c(-3,3))
abline(v= realDifferenceInMeans, col=2)
```



```r
# Z value
(realDifferenceInMeans-mean(res)) / sd(res)
```

```
##      VSO
## 4.499305
```

```r
# P value
sum(res >= realDifferenceInMeans) / length(res)
```

```
## [1] 0
```

So, according to this test, the relationship is still strong. But now let's try controlling for geogrpahic area. To do this, we'll allow data to be permuted only within areas.

Using `tapply` to do structured permutation

This can be done using the function `tapply`, which groups data before applying a function to each group. Let's say we have a vector of letters, and another vector saying whether they're a consonant or a vowel:

```r
myLetters   <-      c('x','z','a','d','e','o')
vowelOrConsonant <- c('c','c','v','c','v','v')
```

We can permute *myLetters* 4 times - note that vowels and consonants are completely mixed up:

```r
sample(myLetters)
```

```
## [1] "x" "o" "e" "a" "z" "d"
```

```r
sample(myLetters)
```

```
## [1] "a" "d" "e" "o" "x" "z"
```

```r
sample(myLetters)
```

```
## [1] "a" "z" "e" "x" "d" "o"
```

```r
sample(myLetters)
```

```
## [1] "x" "z" "d" "a" "o" "e"
```

Next, we can use tapply to first group letters by type, then sample within each group:

```r
tapply(myLetters,vowelOrConsonant, sample)
```

```
## $c
## [1] "z" "x" "d"
##
## $v
## [1] "e" "o" "a"
```

This command returns a list, but we'd like it to return a vector, so we can use the function `unlist`. Let's do it several times and look at the results:

```r
unlist(tapply(myLetters,vowelOrConsonant, sample))
```

```
##  c1  c2  c3  v1  v2  v3
## "d" "z" "x" "a" "o" "e"
```

```r
unlist(tapply(myLetters,vowelOrConsonant, sample))
```

```
##  c1  c2  c3  v1  v2  v3
## "x" "d" "z" "e" "o" "a"
```

```r
unlist(tapply(myLetters,vowelOrConsonant, sample))
```

```
##  c1  c2  c3  v1  v2  v3
## "x" "d" "z" "o" "a" "e"
```

```r
unlist(tapply(myLetters,vowelOrConsonant, sample))
```

```
##  c1  c2  c3  v1  v2  v3
## "d" "x" "z" "e" "o" "a"
```

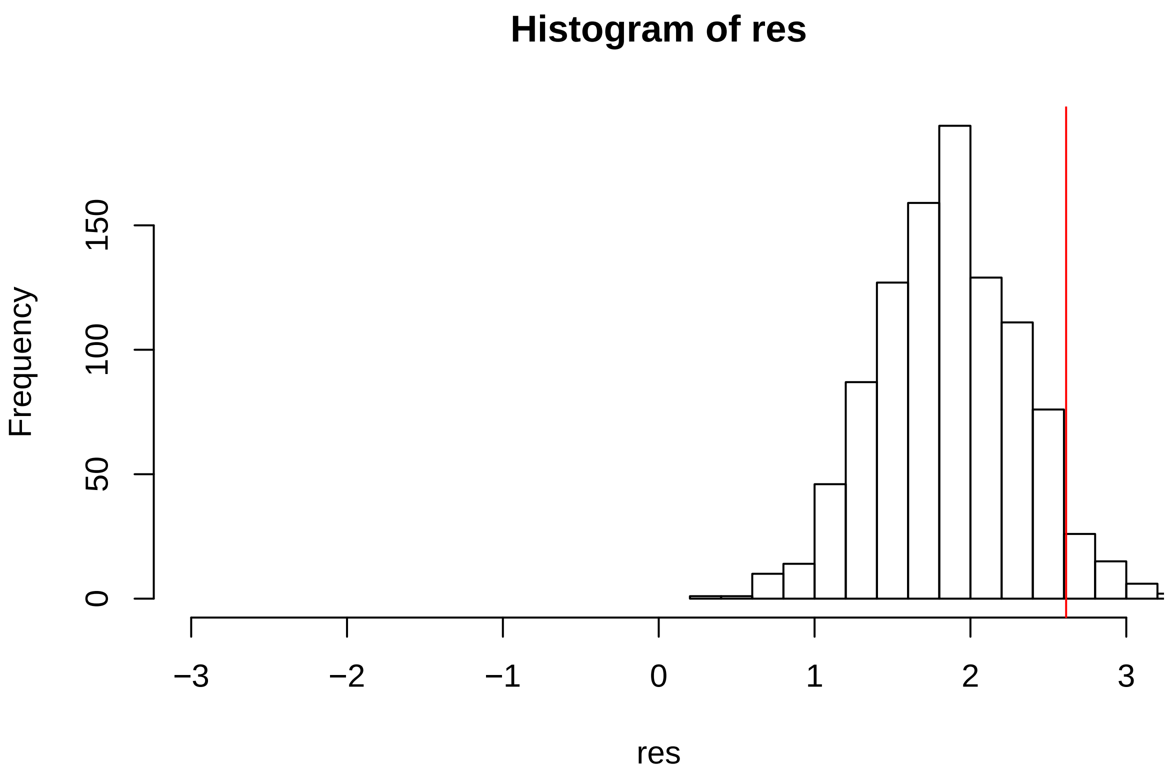
Now note that the vowels can only trade places with anoter vowel. Note also that the resulting vector sorts the data so that the consonants all come first.

Controlling for area

We can now write a function which takes a vector of numbers, a vector indicating which group they belong to and a vector saying which geographic *area* they belong to, and run a structured permutation test.

The function below is similar to the permutation function above, except it uses the tapply trick to permute only within areas. Note also that the numeric vector has to be sorted by area.

```
permuteWithinAreas <- function(x,group,area){
  # permute number
  x <- unlist(tapply(x,area,sample))
  # put group into right order
  group <- unlist(tapply(group,area,sample))
  permutedDifferenceInMeans <-getDifferenceInMeans(x,group)
  return(permutedDifferenceInMeans)
}

res <- replicate(1000, permuteWithinAreas(d2$Name.length, d2$BasicWordOrder,d2$area))

hist(res, xlim=c(-3,3))
abline(v= realDifferenceInMeans, col=2)
```



Note how the distribution has shifted to the right - with the test above, we're much more likely to see higher correlation values. The reason for the difference is that we're using a different baseline to represent random chance.

```
# Z value
(realDifferenceInMeans-mean(res)) / sd(res)
```

```
##      VSO
## 1.634164
```

```r
# P value
sum(res >= realDifferenceInMeans) / length(res)
```

```
## [1] 0.049
```

The result now is marginally significant. This suggests that the length of language name really varies by geographic area, and the original result is due to basic word order also being related to geographic area.

Independent samples

An extreme way of ensuring that your data is not affected by historical or geographical relations is only to run the test with data that you know are not related. That is, to use a sample of independent data points. For example, we could choose a random language from each geographic area for each basic word order type.

The function `sample` has an optional parameter “size”, which defines the number of random selections to return. We can pass this to `sample` inside the `tapply` function to pick just one item from each group:

```r
# Set up some variables
x <- d2$Name.length
group <- d2$BasicWordOrder
area <- d2$area

# split the numbers by group
g1 = x[group=="SOV"]
g2 = x[group=="VSO"]

# split the areas by group
a1 = area[group=="SOV"]
a2 = area[group=="VSO"]

# pick one number from each area for group 1
indepSample1= tapply(g1,a1, sample, size=1)
# pick one number from each area for group 2
indepSample2= tapply(g2,a2, sample, size=1)

# work out the difference in means
indepDiff = mean(indepSample2) - mean(indepSample1)

indepDiff
```

```
## [1] -1.75
```

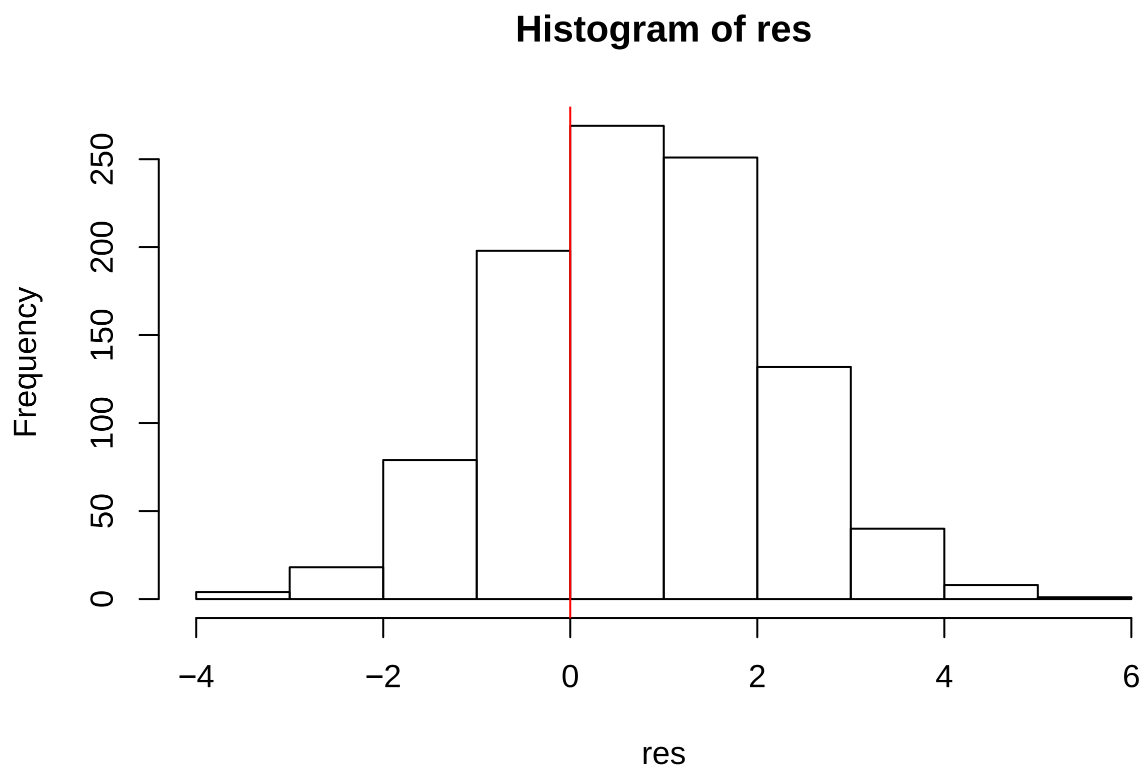
So, this tells us that the difference between SOV and VSO language names from a random independent sample is -1.75 characters. That’s more than zero, but is it significantly more than zero? What would happen if we had chosen a different random selection?

We can answer these questions by running the code above many times. First, we put the code into a custom function, then repliate it many times:

```
indepSample <- function(x,group,area){
  # work out what the two groups are
  groupNames = sort(unique(group))

  # split the x variable into the two groups
  g1 = x[group==groupNames[1]]
  g2 = x[group==groupNames[2]]

  # split the area vairable into the two groups
  a1 = area[group==groupNames[1]]
  a2 = area[group==groupNames[2]]

  # choose independent samples from both groups
  indepSample1= tapply(g1,a1, sample, size=1)
  indepSample2= tapply(g2,a2, sample, size=1)

  # work out difference in means
  indepDiff = mean(indepSample2) - mean(indepSample1)
  return(indepDiff)
}

res = replicate(1000, indepSample(d2$Name.length, d2$BasicWordOrder, d2$area))
```

Let's look at the results, and determine what proportion of independent samples resulted in a difference greater than zero:

```
hist(res)
abline(v= 0, col=2)
```



```r
# P value
p.value <- sum(res >= 0) / length(res)

p.value
```

```
## [1] 0.712
```

The p value is 0.712 - not significant at all. This suggests that the correlation between language name length and basic word order is really just driven by languages being related geographically.

However, it's worth noting that this test is very conservative - it throws away a lot of information and runs tests on a much smaller sample of the data. It's possible that this would lead to reduced statistical power, and an inability to detect relationships that really were there.

Combining permutation and independent samples

In the tests above, we took one language from each area where we could. But this results in an imbalance in the sample sizes - there are more areas with SVO languages than with VSO langauges. Larger samples can include more extreme measures by chance, which could affect our result. We could control for this by only selecting the same number of observations for each group.

In addition, we were assuming that the difference between indpendent samples by chance was zero.

We could address these shortcomings by combining independent samples and random permutation. The idea is to take indpendent samples, then run a mini permutation test on them.

In the code below, I add a function which takes indpendent samples from each area for each group, but then makes sure both samples are the same size by throwing out data from the larger group. The data I throw out is picked randomly.

In addition, I don't just return the difference between the means of the two samples. When I've picked my two samples, I permute one of them and then return the difference between the *real* difference and the permuted difference.

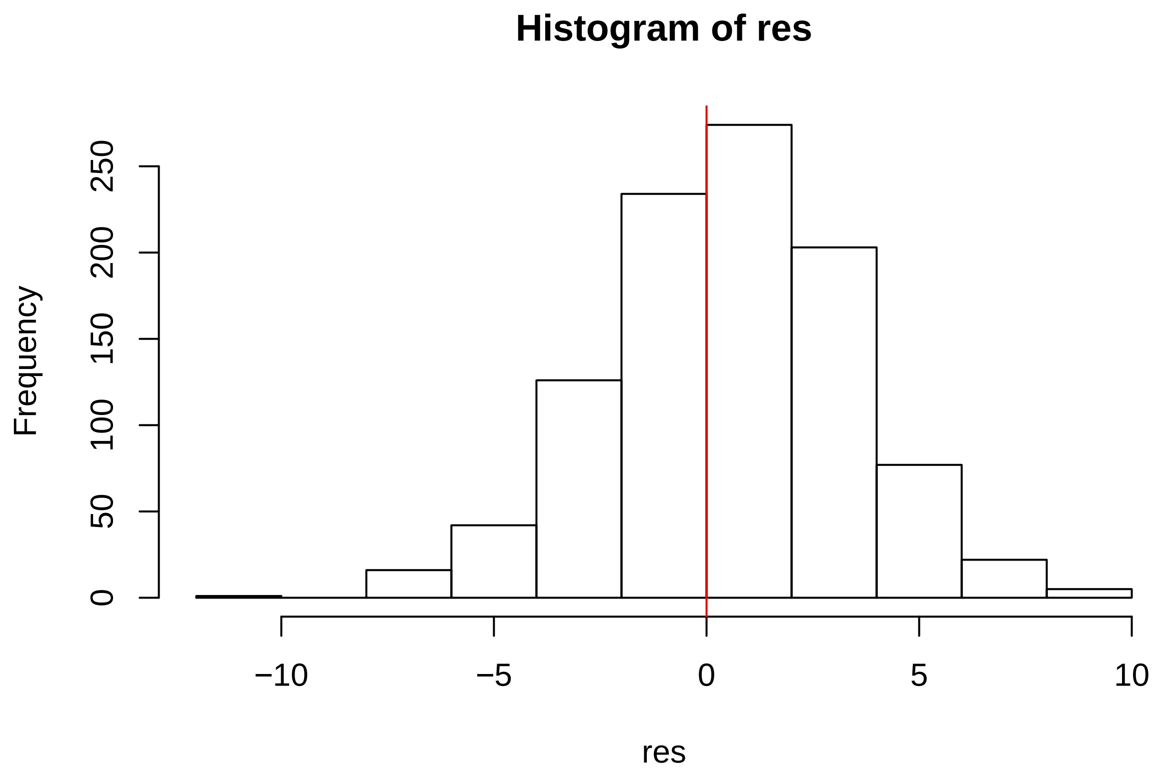
Phew! That's hard to explain in words. Here's the code:

```r
indepSample.equalSizes <- function(x,group,area){
  # work out what the two groups are
  groupNames = unique(group)

  # split the x variable into the two groups
  g1 = x[group==groupNames[1]]
  g2 = x[group==groupNames[2]]

  # split the area vairable into the two groups
  a1 = area[group==groupNames[1]]
  a2 = area[group==groupNames[2]]

  # choose independent samples from both groups
  indepSample1= tapply(g1,a1, sample, size=1)
  indepSample2= tapply(g2,a2, sample, size=1)

  # work out how many samples are in the smallest group
  chooseNum <- min(length(indepSample1),length(indepSample2))
  # randomly pick items from each group, so they have the same size
  indepSample1 <- sample(indepSample1, size=chooseNum)
  indepSample2 <- sample(indepSample2, size=chooseNum)

  # work out the difference in means
  indepDiff = mean(indepSample2) - mean(indepSample1)

  ### Permutation:

  # permute membership between samples by
  # 1) joining them together and permuting
  permutedSamples1and2 <- sample(c(indepSample1,indepSample2))
  # 2) splitting them into 2 new groups
  permSample1 <-permutedSamples1and2[1:chooseNum]
  permSample2 <-permutedSamples1and2[(chooseNum+1):length(permutedSamples1and2)]

  # work out the difference in means for the permuted group
  permDiff = mean(permSample2) - mean(permSample1)

  # return difference between real independent samples and permuted independent samples
  return(indepDiff - permDiff)
}
```

If the function above returns a value greater than zero, that indicates that the relationship is stronger for the real data than for the permuted data.

Now we can run this many times and look at the results.

```
res = replicate(1000, indepSample.equalSizes(d2$Name.length, d2$BasicWordOrder, d2$area))

hist(res)
abline(v= 0, col=2)
```



```r
# P value
sum(res >= 0) / length(res)
```

```
## [1] 0.603
```

The distribution is centered around zero, indicating that the relationship is not strong: there's almost an equal chance of the real relationship to be weaker or stronger than for the permuted data.

This technique is discussed in this supporting information of this paper.

Everett, Blasí & Roberts (2016) Language evolution and climate: the case of desiccation and tone. Journal of Language Evolution Jan 2016, 1 (1) 33-46;

Mixed effects modelling

Another method for controlling for language relatedness is mixed effects modelling. The theory of how mixed effects modelling works is beyond the scope of this tutorial, but below I show how it might be done, to compare with the results above. Note that, in comparison to the indpendent sample method, this uses all the data, and so has greater statistical power. It's also easier to test a range of constraints than the permutation method.

First, we load the `lme4` library:

```r
library(lme4)
```

```
## Warning: package 'lme4' was built under R version 3.2.3
```

```
## Loading required package: Matrix
## Warning: package 'Matrix' was built under R version 3.2.4
```

Next, we make two models predicting the probability of a language having SOV order. One just tries to predict the overall probability, with a seperate probability for each area. The second adds the adposition order as a fixed effect.

If the second model provides a better fit to the data, then that suggests that basic word order does help predict name length.

```
library(lme4)

# Model 0 - null model with random effects for area
m0 = lmer(Name.length  ~ 1 + (1 +BasicWordOrder | area), data=d2)

# Model 1 - model with fixed effect for AdpositionOrder
m1 = lmer(Name.length ~BasicWordOrder + (1 + BasicWordOrder | area), data=d2)

# Has the model fit improved when adding adposition order?
anova(m0,m1)
```

```
## refitting model(s) with ML (instead of REML)
```

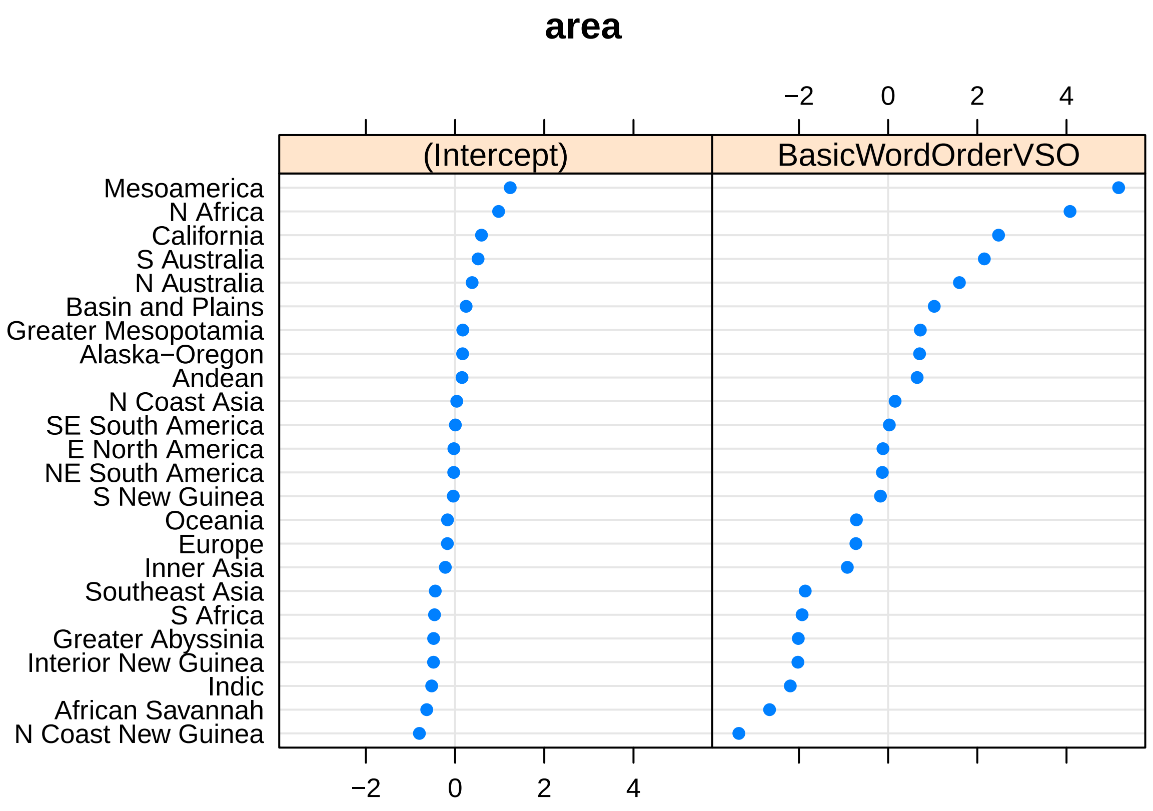
```
## Data: d2
## Models:
## m0: Name.length ~ 1 + (1 + BasicWordOrder | area)
## m1: Name.length ~ BasicWordOrder + (1 + BasicWordOrder | area)
##    Df    AIC    BIC  logLik deviance  Chisq Chi Df Pr(>Chisq)
## m0  5 2591.8 2612.3 -1290.9   2581.8
## m1  6 2591.7 2616.3 -1289.9   2579.7 2.0955      1     0.1477
```

The second model does not improve the fit of the data. This suggests that, if you know what area a language belongs to, then knowing the basic word order won't help you improve your guess about the language name length. That is, language name length and basic word order are not correlated.

Here's a plot of the random effects, which show that in some areas, the relationship between basic word order and name length actually goes in the opposite direction to the hypothesised prediction.

```
library(lattice)
dotplot(ranef(m1))
```

```
## $area
```



Why did we find the correlation in the first place?

If we look at the language names, we note that several have extra information in brackets, such as the geographic location or family, to distinguish them from similar languages.

```
head(d2$Name[grepl("\\(",d2$Name)])
```

```
## [1] "Adyghe (Abzakh)"         "Kanum (Bädi)"
## [3] "Agta (Central)"          "Tarahumara (Central)"
## [5] "Berber (Middle Atlas)"   "Chinantec (Comaltepec)"
```

If we remove the text in brackets:

```
d2$Name.short <- gsub("\\([^\\)]+\\)","",d2$Name)
# remove spaces at end of word
d2$Name.short <- gsub(" $","",d2$Name.short)
d2$Name.short.length = nchar(d2$Name.short)
```

Then recalculate the original t-test:

```
t.test(d2$Name.short.length~d2$BasicWordOrder)
```

```
##
##  Welch Two Sample t-test
##
```

```
## data:  d2$Name.short.length by d2$BasicWordOrder
## t = -1.5692, df = 120.4, p-value = 0.1192
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  -0.8778592  0.1015792
## sample estimates:
## mean in group SOV mean in group VSO
##          6.269755          6.657895
```

We see there's no relationship.