Class: 13rd April 2018

**Topics**

1. Descriptive Statistics
2. Frequencies and Crosstabs
3. Correlations
4. t-tests
5. Nonparametric Tests of Group Differences
6. Multiple (Linear) Regression
7. Regression Diagnostics
8. ANOVA/MANOVA
9. (M)ANOVA Assumption
10. Resampling Statistics
11. Power analysis
12. Using with() and by()

**Descriptive Statistics**

Descriptive statistics are used to describe the basic features of the data in a study. They provide simple summaries about the sample and the measures. Together with simple graphics analysis, they form the basis of virtually every quantitative analysis of data.

One method of obtaining descriptive statistics is to use the sapply() function with a specified summary statistic.

# get means for variables in data frame mydata  
# excluding missing values   
sapply(mtcars, mean)

mpg cyl disp hp drat wt qsec vs am gear carb

20.090625 6.187500 230.721875 146.687500 3.596563 3.217250 17.848750 0.437500 0.406250 3.687500 2.812500

**Frequencies and Crosstabs**

Generating Frequency Tables

R provides many methods for creating frequency and contingency tables. Three are described below. In the following examples, assume that A, B, and C represent categorical variables.

### **table**

You can generate frequency tables using the **table( )** function, tables of proportions using the **prop.table( )** function, and marginal frequencies using **margin.table( )**.

> attach(mtcars)

> testtable <- table(mpg,wt)//mpg is raw, wt is column

> testtable

wt

mpg 1.513 1.615 1.835 1.935 2.14 2.2 2.32 2.465 2.62 2.77 2.78 2.875 3.15 3.17

10.4 0 0 0 0 0 0 0 0 0 0 0 0 0 0

13.3 0 0 0 0 0 0 0 0 0 0 0 0 0 0

14.3 0 0 0 0 0 0 0 0 0 0 0 0 0 0

14.7 0 0 0 0 0 0 0 0 0 0 0 0 0 0

15 0 0 0 0 0 0 0 0 0 0 0 0 0 0

15.2 0 0 0 0 0 0 0 0 0 0 0 0 0 0

15.5 0 0 0 0 0 0 0 0 0 0 0 0 0 0

15.8 0 0 0 0 0 0 0 0 0 0 0 0 0 1

16.4 0 0 0 0 0 0 0 0 0 0 0 0 0 0

17.3 0 0 0 0 0 0 0 0 0 0 0 0 0 0

17.8 0 0 0 0 0 0 0 0 0 0 0 0 0 0

18.1 0 0 0 0 0 0 0 0 0 0 0 0 0 0

18.7 0 0 0 0 0 0 0 0 0 0 0 0 0 0

19.2 0 0 0 0 0 0 0 0 0 0 0 0 0 0

19.7 0 0 0 0 0 0 0 0 0 1 0 0 0 0

21 0 0 0 0 0 0 0 0 1 0 0 1 0 0

21.4 0 0 0 0 0 0 0 0 0 0 1 0 0 0

21.5 0 0 0 0 0 0 0 1 0 0 0 0 0 0

22.8 0 0 0 0 0 0 1 0 0 0 0 0 1 0

24.4 0 0 0 0 0 0 0 0 0 0 0 0 0 0

26 0 0 0 0 1 0 0 0 0 0 0 0 0 0

27.3 0 0 0 1 0 0 0 0 0 0 0 0 0 0

30.4 1 1 0 0 0 0 0 0 0 0 0 0 0 0

32.4 0 0 0 0 0 1 0 0 0 0 0 0 0 0

33.9 0 0 1 0 0 0 0 0 0 0 0 0 0 0

wt

mpg 3.19 3.215 3.435 3.44 3.46 3.52 3.57 3.73 3.78 3.84 3.845 4.07 5.25 5.345

10.4 0 0 0 0 0 0 0 0 0 0 0 0 1 0

13.3 0 0 0 0 0 0 0 0 0 1 0 0 0 0

14.3 0 0 0 0 0 0 1 0 0 0 0 0 0 0

14.7 0 0 0 0 0 0 0 0 0 0 0 0 0 1

15 0 0 0 0 0 0 1 0 0 0 0 0 0 0

15.2 0 0 1 0 0 0 0 0 1 0 0 0 0 0

15.5 0 0 0 0 0 1 0 0 0 0 0 0 0 0

15.8 0 0 0 0 0 0 0 0 0 0 0 0 0 0

16.4 0 0 0 0 0 0 0 0 0 0 0 1 0 0

17.3 0 0 0 0 0 0 0 1 0 0 0 0 0 0

17.8 0 0 0 1 0 0 0 0 0 0 0 0 0 0

18.1 0 0 0 0 1 0 0 0 0 0 0 0 0 0

18.7 0 0 0 1 0 0 0 0 0 0 0 0 0 0

19.2 0 0 0 1 0 0 0 0 0 0 1 0 0 0

19.7 0 0 0 0 0 0 0 0 0 0 0 0 0 0

21 0 0 0 0 0 0 0 0 0 0 0 0 0 0

21.4 0 1 0 0 0 0 0 0 0 0 0 0 0 0

21.5 0 0 0 0 0 0 0 0 0 0 0 0 0 0

22.8 0 0 0 0 0 0 0 0 0 0 0 0 0 0

24.4 1 0 0 0 0 0 0 0 0 0 0 0 0 0

26 0 0 0 0 0 0 0 0 0 0 0 0 0 0

27.3 0 0 0 0 0 0 0 0 0 0 0 0 0 0

30.4 0 0 0 0 0 0 0 0 0 0 0 0 0 0

32.4 0 0 0 0 0 0 0 0 0 0 0 0 0 0

33.9 0 0 0 0 0 0 0 0 0 0 0 0 0 0

wt

mpg 5.424

10.4 1

13.3 0

14.3 0

14.7 0

15 0

15.2 0

15.5 0

15.8 0

16.4 0

17.3 0

17.8 0

18.1 0

18.7 0

19.2 0

19.7 0

21 0

21.4 0

21.5 0

22.8 0

24.4 0

26 0

27.3 0

30.4 0

32.4 0

33.9 0

margin.table(testtable, 1) # A frequencies (summed over B)   
margin.table(testtable, 2) # B frequencies (summed over A)  
  
prop.table(testtable) # cell percentages  
prop.table(testtable, 1) # row percentages   
prop.table(testtable, 2) # column percentages

**table()** can also generate multidimensional tables based on 3 or more categorical variables. In this case, use the **ftable()** function to print the results more attractively.

# 3-Way Frequency Table   
mytable <- table(mpg, wt, cyl)   
ftable(testtable)

### **Crosstable**

The **CrossTable( )** function in the gmodels package produces crosstabulations modeled after PROC FREQ in **SAS** or CROSSTABS in SPSS. It has a wealth of options.

# Correlations

You can use the **cor()** function to produce correlations and the **cov()** function to produces covariances.

A simplified format is **cor(x, use=, method= )** where

|  |  |
| --- | --- |
| Option | Description |
| x | Matrix or data frame |
| use | Specifies the handling of missing data. Options are all.obs(assumes no missing data - missing data will produce an error), complete.obs (listwise deletion), and pairwise.complete.obs(pairwise deletion) |
| method | Specifies the type of correlation. Options are pearson, spearman or kendall. |

# Correlations with significance levels  
library(Hmisc)  
rcorr(x, type="pearson") # type can be pearson or spearman  
  
#mtcars is a data frame   
rcorr(as.matrix(mtcars))

You can use the format **cor(X,Y)**or **rcorr(X,Y)** to generate correlations between the columns of X and the columns of Y. This similar to the VAR and WITH commands in **SAS** PROC CORR.

# Correlation matrix from mtcars  
# with mpg, cyl, and disp as rows   
# and hp, drat, and wt as columns   
> x <- mtcars[1:3]

> y <- mtcars[4:6]

> cor(x, y)

hp drat wt

mpg -0.7761684 0.6811719 -0.8676594

cyl 0.8324475 -0.6999381 0.7824958

disp 0.7909486 -0.7102139 0.8879799

**Other types of correlations**

# polychoric correlation  
# x is a contingency table of counts

x <- mtcars[1:3]

install.packages(‘polycor’)  
library(polycor)  
polychor(x)   
[1] -0.01036146

# heterogeneous correlations in one matrix   
# pearson (numeric-numeric),   
# polyserial (numeric-ordinal),   
# and polychoric (ordinal-ordinal)  
# x is a data frame with ordered factors   
# and numeric variables

x <- mtcars[1:3]  
library(polycor)  
hetcor(x)

Two-Step Estimates

Correlations/Type of Correlation:

mpg cyl disp

mpg 1 Pearson Pearson

cyl -0.8522 1 Pearson

disp -0.8476 0.902 1

Standard Errors:

mpg cyl

mpg

cyl 0.04919

disp 0.05059 0.03353

n = 32

P-values for Tests of Bivariate Normality:

mpg cyl

mpg

cyl 0.008597

disp 0.5738 0.008302  
  
# partial correlations

install.packages(‘ggm’)  
library(ggm)  
data(mtcars)  
> pcor(c("disp", "cyl", "mpg", "wt", "gear"), var(mtcars))

[1] 0.6801385

# partial corr between disp and cyl controlling for mpg, wt and gear.

**t-tests**

The **t.test()**function produces a variety of t-tests. Unlike most statistical packages, the default assumes unequal variance and applies the Welsh df modification.

# independent 2-group t-test

> t.test(mtcars$mpg, mtcars$cyl) //mpg and cyl are numaric factors

Welch Two Sample t-test

data: mtcars$mpg and mtcars$cyl

t = 12.512, df = 36.402, p-value = 9.508e-15

alternative hypothesis: true difference in means is not equal to 0

95 percent confidence interval:

11.65034 16.15591

sample estimates:

mean of x mean of y

20.09062 6.18750

# paired t-test

> t.test(mtcars$mpg, mtcars$cyl, paired = TRUE)

Paired t-test

data: mtcars$mpg and mtcars$cyl

t = 10.34, df = 31, p-value = 1.435e-11

alternative hypothesis: true difference in means is not equal to 0

95 percent confidence interval:

11.16070 16.64555

sample estimates:

mean of the differences

13.90312

# Nonparametric Tests of Group Differences

R provides functions for carrying out Mann-Whitney U, Wilcoxon Signed Rank, Kruskal Wallis, and Friedman tests.

# independent 2-group Mann-Whitney U Test   
wilcox.test(y~A)   
# where y is numeric and A is A binary factor

# independent 2-group Mann-Whitney U Test  
wilcox.test(y,x) # where y and x are numeric

# dependent 2-group Wilcoxon Signed Rank Test   
wilcox.test(y1,y2,paired=TRUE) # where y1 and y2 are numeric

# Kruskal Wallis Test One Way Anova by Ranks   
kruskal.test(y~A) # where y1 is numeric and A is a factor

# Randomized Block Design - Friedman Test   
friedman.test(y~A|B)  
# where y are the data values, A is a grouping factor  
# and B is a blocking factor

# Multiple (Linear) Regression

R provides comprehensive support for multiple linear regression. The topics below are provided in order of increasing complexity.

# Multiple Linear Regression Example   
> fit <- lm(wt ~ cyl + mpg + disp, data=mtcars)

> fit

Call:

lm(formula = wt ~ cyl + mpg + disp, data = mtcars)

Coefficients:

(Intercept) cyl mpg disp

4.669367 -0.186998 -0.083558 0.005997

> summary(fit)

Call:

lm(formula = wt ~ cyl + mpg + disp, data = mtcars)

Residuals:

Min 1Q Median 3Q Max

-0.78815 -0.29883 -0.08519 0.32692 0.76120

Coefficients:

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

(Intercept) 4.669367 0.891094 5.240 1.44e-05 \*\*\*

cyl -0.186998 0.099183 -1.885 0.069795 .

mpg -0.083558 0.023905 -3.495 0.001596 \*\*

disp 0.005997 0.001409 4.256 0.000211 \*\*\*

---

Signif. codes: 0 ‘\*\*\*’ 0.001 ‘\*\*’ 0.01 ‘\*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 0.3933 on 28 degrees of freedom

Multiple R-squared: 0.854, Adjusted R-squared: 0.8384

F-statistic: 54.61 on 3 and 28 DF, p-value: 8.027e-12

# Other useful functions   
coefficients(fit) # model coefficients  
confint(fit, level=0.95) # CIs for model parameters   
fitted(fit) # predicted values  
residuals(fit) # residuals  
anova(fit) # anova table   
vcov(fit) # covariance matrix for model parameters   
influence(fit) # regression diagnostics

**Model Fitting**

# Multiple Linear Regression Example   
fit <- lm(y ~ x1 + x2 + x3, data=mydata)  
summary(fit) # show results

# Other useful functions   
coefficients(fit) # model coefficients  
confint(fit, level=0.95) # CIs for model parameters   
fitted(fit) # predicted values  
residuals(fit) # residuals  
anova(fit) # anova table   
vcov(fit) # covariance matrix for model parameters   
influence(fit) # regression diagnostics

Regression Diagnostics

An excellent review of regression diagnostics is provided in John Fox's aptly named Overview of Regression Diagnostics. Dr. Fox's car package provides advanced utilities for regression modeling.

# Assume that we are fitting a multiple linear regression

# on the MTCARS data

library(car)

fit <- lm(mpg~disp+hp+wt+drat, data=mtcars)

This example is for exposition only. We will ignore the fact that this may not be a great way of modeling the this particular set of data!

Outliers

# Assessing Outliers

outlierTest(fit) # Bonferonni p-value for most extreme obs

qqPlot(fit, main="QQ Plot") #qq plot for studentized resid

leveragePlots(fit) # leverage plots

leverage plot click to view

Influential Observations

# Influential Observations

# added variable plots

av.Plots(fit)

# Cook's D plot

# identify D values > 4/(n-k-1)

cutoff <- 4/((nrow(mtcars)-length(fit$coefficients)-2))

plot(fit, which=4, cook.levels=cutoff)

# Influence Plot

influencePlot(fit, id.method="identify", main="Influence Plot", sub="Circle size is proportial to Cook's Distance" )

av plots Cook's D plot influence plot click to view

Non-normality

# Normality of Residuals

# qq plot for studentized resid

qqPlot(fit, main="QQ Plot")

# distribution of studentized residuals

library(MASS)

sresid <- studres(fit)

hist(sresid, freq=FALSE,

main="Distribution of Studentized Residuals")

xfit<-seq(min(sresid),max(sresid),length=40)

yfit<-dnorm(xfit)

lines(xfit, yfit)

qq plot histogram of studentized residuals click to view

Non-constant Error Variance

# Evaluate homoscedasticity

# non-constant error variance test

ncvTest(fit)

# plot studentized residuals vs. fitted values

spreadLevelPlot(fit)

spread vs. levels click to view

Multi-collinearity

# Evaluate Collinearity

vif(fit) # variance inflation factors

sqrt(vif(fit)) > 2 # problem?

Nonlinearity

# Evaluate Nonlinearity

# component + residual plot

crPlots(fit)

# Ceres plots

ceresPlots(fit)

component plus residual plot Ceres plots click to view

Non-independence of Errors

# Test for Autocorrelated Errors

durbinWatsonTest(fit)

Additional Diagnostic Help

The gvlma( ) function in the gvlma package, performs a global validation of linear model assumptions as well separate evaluations of skewness, kurtosis, and heteroscedasticity.

# Global test of model assumptions

library(gvlma)

gvmodel <- gvlma(fit)

summary(gvmodel)

ANOVA

If you have been analyzing ANOVA designs in traditional statistical packages, you are likely to find R's approach less coherent and user-friendly. A good online presentation on ANOVA in R can be found in ANOVA section of the Personality Project. (Note: I have found that these pages render fine in Chrome and Safari browsers, but can appear distorted in iExplorer.)

1. Fit a Model

In the following examples lower case letters are numeric variables and upper case letters are factors.

# One Way Anova (Completely Randomized Design)

fit <- aov(y ~ A, data=mydataframe)

# Randomized Block Design (B is the blocking factor)

fit <- aov(y ~ A + B, data=mydataframe)

# Two Way Factorial Design

fit <- aov(y ~ A + B + A:B, data=mydataframe)

fit <- aov(y ~ A\*B, data=mydataframe) # same thing

# Analysis of Covariance

fit <- aov(y ~ A + x, data=mydataframe)

For within subjects designs, the data frame has to be rearranged so that each measurement on a subject is a separate observation. See R and Analysis of Variance.

# One Within Factor

fit <- aov(y~A+Error(Subject/A),data=mydataframe)

# Two Within Factors W1 W2, Two Between Factors B1 B2

fit <- aov(y~(W1\*W2\*B1\*B2)+Error(Subject/(W1\*W2))+(B1\*B2),

data=mydataframe)

2. Look at Diagnostic Plots

Diagnostic plots provide checks for heteroscedasticity, normality, and influential observerations.

layout(matrix(c(1,2,3,4),2,2)) # optional layout

plot(fit) # diagnostic plots

For details on the evaluation of test requirements, see (M)ANOVA Assumptions.

3. Evaluate Model Effects

WARNING: R provides Type I sequential SS, not the default Type III marginal SS reported by SAS and SPSS. In a nonorthogonal design with more than one term on the right hand side of the equation order will matter (i.e., A+B and B+A will produce different results)! We will need use the drop1( ) function to produce the familiar Type III results. It will compare each term with the full model. Alternatively, we can use anova(fit.model1, fit.model2) to compare nested models directly.

summary(fit) # display Type I ANOVA table

drop1(fit,~.,test="F") # type III SS and F Tests

Nonparametric and resampling alternatives are available.

Multiple Comparisons

You can get Tukey HSD tests using the function below. By default, it calculates post hoc comparisons on each factor in the model. You can specify specific factors as an option. Again, remember that results are based on Type I SS!

# Tukey Honestly Significant Differences

TukeyHSD(fit) # where fit comes from aov()

Visualizing Results

Use box plots and line plots to visualize group differences. There are also two functions specifically designed for visualizing mean differences in ANOVA layouts. interaction.plot( ) in the base stats package produces plots for two-way interactions. plotmeans( ) in the gplots package produces mean plots for single factors, and includes confidence intervals.

# Two-way Interaction Plot

attach(mtcars)

gears <- factor(gears)

cyl <- factor(cyl)

interaction.plot(cyl, gear, mpg, type="b", col=c(1:3),

leg.bty="o", leg.bg="beige", lwd=2, pch=c(18,24,22),

xlab="Number of Cylinders",

ylab="Mean Miles Per Gallon",

main="Interaction Plot")

interaction plot click to view

# Plot Means with Error Bars

library(gplots)

attach(mtcars)

cyl <- factor(cyl)

plotmeans(mpg~cyl,xlab="Number of Cylinders",

ylab="Miles Per Gallon", main="Mean Plot\nwith 95% CI")

mean plot click to view

MANOVA

If there is more than one dependent (outcome) variable, you can test them simultaneously using a multivariate analysis of variance (MANOVA). In the following example, let Y be a matrix whose columns are the dependent variables.

# 2x2 Factorial MANOVA with 3 Dependent Variables.

Y <- cbind(y1,y2,y3)

fit <- manova(Y ~ A\*B)

summary(fit, test="Pillai")

Other test options are "Wilks", "Hotelling-Lawley", and "Roy". Use summary.aov( ) to get univariate statistics. TukeyHSD( ) and plot( ) will not work with a MANOVA fit. Run each dependent variable separately to obtain them. Like ANOVA, MANOVA results in R are based on Type I SS. To obtain Type III SS, vary the order of variables in the model and rerun the analyses. For example, fit y~A\*B for the TypeIII B effect and y~B\*A for the Type III A effect.

Assessing Classical Test Assumptions

In classical parametric procedures we often assume normality and constant variance for the model error term. Methods of exploring these assumptions in an ANOVA/ANCOVA/MANOVA framework are discussed here. Regression diagnostics are covered under multiple linear regression.

Outliers

Since outliers can severly affect normality and homogeneity of variance, methods for detecting disparate observerations are described first.

The aq.plot() function in the mvoutlier package allows you to identfy multivariate outliers by plotting the ordered squared robust Mahalanobis distances of the observations against the empirical distribution function of the MD2i. Input consists of a matrix or data frame. The function produces 4 graphs and returns a boolean vector identifying the outliers.

# Detect Outliers in the MTCARS Data

library(mvoutlier)

outliers <-

aq.plot(mtcars[c("mpg","disp","hp","drat","wt","qsec")])

outliers # show list of outliers

outliers click to view

Univariate Normality

You can evaluate the normality of a variable using a Q-Q plot.

# Q-Q Plot for variable MPG

attach(mtcars)

qqnorm(mpg)

qqline(mpg)

qqplot click to view

Significant departures from the line suggest violations of normality.

You can also perform a Shapiro-Wilk test of normality with the shapiro.test(x) function, where x is a numeric vector. Additional functions for testing normality are available in nortest package.

Multivariate Normality

MANOVA assumes multivariate normality. The function mshapiro.test( ) in the mvnormtest package produces the Shapiro-Wilk test for multivariate normality. Input must be a numeric matrix.

# Test Multivariate Normality

mshapiro.test(M)

If we have p x 1 multivariate normal random vector x vector

then the squared Mahalanobis distance between x and μ is going to be chi-square distributed with p degrees of freedom. We can use this fact to construct a Q-Q plot to assess multivariate normality.

# Graphical Assessment of Multivariate Normality

x <- as.matrix(mydata) # n x p numeric matrix

center <- colMeans(x) # centroid

n <- nrow(x); p <- ncol(x); cov <- cov(x);

d <- mahalanobis(x,center,cov) # distances

qqplot(qchisq(ppoints(n),df=p),d,

main="QQ Plot Assessing Multivariate Normality",

ylab="Mahalanobis D2")

abline(a=0,b=1)

mnormal qq plot click to view

Homogeneity of Variances

The bartlett.test( ) function provides a parametric K-sample test of the equality of variances. The fligner.test( ) function provides a non-parametric test of the same. In the following examples y is a numeric variable and G is the grouping variable.

# Bartlett Test of Homogeneity of Variances

bartlett.test(y~G, data=mydata)

# Figner-Killeen Test of Homogeneity of Variances

fligner.test(y~G, data=mydata)

The hovPlot( ) function in the HH package provides a graphic test of homogeneity of variances based on Brown-Forsyth. In the following example, y is numeric and G is a grouping factor. Note that G must be of type factor.

# Homogeneity of Variance Plot

library(HH)

hov(y~G, data=mydata)

hovPlot(y~G,data=mydata)

hov click to view

Homogeneity of Covariance Matrices

MANOVA and LDF assume homogeneity of variance-covariance matrices. The assumption is usually tested with Box's M. Unfortunately the test is very sensitive to violations of normality, leading to rejection in most typical cases. Box's M is available via the boxM function in the biotools package.

Resampling Statistics

The coin package provides the ability to perform a wide variety of re-randomization or permutation based statistical tests. These tests do not assume random sampling from well-defined populations. They can be a reasonable alternative to classical procedures when test assumptions can not be met. See coin: A Computational Framework for Conditional Inference for details.

In the examples below, lower case letters represent numerical variables and upper case letters represent categorical factors. Monte-Carlo simulation are available for all tests. Exact tests are available for 2 group procedures.

Independent Two- and K-Sample Location Tests

# Exact Wilcoxon Mann Whitney Rank Sum Test

# where y is numeric and A is a binary factor

library(coin)

wilcox\_test(y~A, data=mydata, distribution="exact")

# One-Way Permutation Test based on 9999 Monte-Carlo

# resamplings. y is numeric and A is a categorical factor

library(coin)

oneway\_test(y~A, data=mydata,

distribution=approximate(B=9999))

Symmetry of a response for repeated measurements

# Exact Wilcoxon Signed Rank Test

# where y1 and y2 are repeated measures

library(coin)

wilcoxsign\_test(y1~y2, data=mydata, distribution="exact")

# Freidman Test based on 9999 Monte-Carlo resamplings.

# y is numeric, A is a grouping factor, and B is a

# blocking factor.

library(coin)

friedman\_test(y~A|B, data=mydata,

distribution=approximate(B=9999))

Independence of Two Numeric Variables

# Spearman Test of Independence based on 9999 Monte-Carlo

# resamplings. x and y are numeric variables.

library(coin)

spearman\_test(y~x, data=mydata,

distribution=approximate(B=9999))

Independence in Contingency Tables

# Independence in 2-way Contingency Table based on

# 9999 Monte-Carlo resamplings. A and B are factors.

library(coin)

chisq\_test(A~B, data=mydata,

distribution=approximate(B=9999))

# Cochran-Mantel-Haenzsel Test of 3-way Contingency Table

# based on 9999 Monte-Carlo resamplings. A, B, are factors

# and C is a stratefying factor.

library(coin)

mh\_test(A~B|C, data=mydata,

distribution=approximate(B=9999))

# Linear by Linear Association Test based on 9999

# Monte-Carlo resamplings. A and B are ordered factors.

library(coin)

lbl\_test(A~B, data=mydata,

distribution=approximate(B=9999))

Many other univariate and multivariate tests are possible using the functions in the coin package. See A Lego System for Conditional Inference for more details.

Power Analysis

Overview

Power analysis is an important aspect of experimental design. It allows us to determine the sample size required to detect an effect of a given size with a given degree of confidence. Conversely, it allows us to determine the probability of detecting an effect of a given size with a given level of confidence, under sample size constraints. If the probability is unacceptably low, we would be wise to alter or abandon the experiment.

The following four quantities have an intimate relationship:

sample size

effect size

significance level = P(Type I error) = probability of finding an effect that is not there

power = 1 - P(Type II error) = probability of finding an effect that is there

Given any three, we can determine the fourth.

Power Analysis in R

The pwr package develped by Stéphane Champely, impliments power analysis as outlined by Cohen (!988). Some of the more important functions are listed below.

function power calculations for

pwr.2p.test two proportions (equal n)

pwr.2p2n.test two proportions (unequal n)

pwr.anova.test balanced one way ANOVA

pwr.chisq.test chi-square test

pwr.f2.test general linear model

pwr.p.test proportion (one sample)

pwr.r.test correlation

pwr.t.test t-tests (one sample, 2 sample, paired)

pwr.t2n.test t-test (two samples with unequal n)

For each of these functions, you enter three of the four quantities (effect size, sample size, significance level, power) and the fourth is calculated.

The significance level defaults to 0.05. Therefore, to calculate the significance level, given an effect size, sample size, and power, use the option "sig.level=NULL".

Specifying an effect size can be a daunting task. ES formulas and Cohen's suggestions (based on social science research) are provided below. Cohen's suggestions should only be seen as very rough guidelines. Your own subject matter experience should be brought to bear.

(To explore confidence intervals and drawing conclusions from samples try this interactive course on the foundations of inference.)

t-tests

For t-tests, use the following functions:

pwr.t.test(n = , d = , sig.level = , power = , type = c("two.sample", "one.sample", "paired"))

where n is the sample size, d is the effect size, and type indicates a two-sample t-test, one-sample t-test or paired t-test. If you have unequal sample sizes, use

pwr.t2n.test(n1 = , n2= , d = , sig.level =, power = )

where n1 and n2 are the sample sizes.

For t-tests, the effect size is assessed as

Cohen d

Cohen suggests that d values of 0.2, 0.5, and 0.8 represent small, medium, and large effect sizes respectively.

You can specify alternative="two.sided", "less", or "greater" to indicate a two-tailed, or one-tailed test. A two tailed test is the default.

ANOVA

For a one-way analysis of variance use

pwr.anova.test(k = , n = , f = , sig.level = , power = )

where k is the number of groups and n is the common sample size in each group.

For a one-way ANOVA effect size is measured by f where

Cohen f

Cohen suggests that f values of 0.1, 0.25, and 0.4 represent small, medium, and large effect sizes respectively.

Correlations

For correlation coefficients use

pwr.r.test(n = , r = , sig.level = , power = )

where n is the sample size and r is the correlation. We use the population correlation coefficient as the effect size measure. Cohen suggests that r values of 0.1, 0.3, and 0.5 represent small, medium, and large effect sizes respectively.

Linear Models

For linear models (e.g., multiple regression) use

pwr.f2.test(u =, v = , f2 = , sig.level = , power = )

where u and v are the numerator and denominator degrees of freedom. We use f2 as the effect size measure.

cohen f2

Cohen f2 alternate

The first formula is appropriate when we are evaluating the impact of a set of predictors on an outcome. The second formula is appropriate when we are evaluating the impact of one set of predictors above and beyond a second set of predictors (or covariates). Cohen suggests f2 values of 0.02, 0.15, and 0.35 represent small, medium, and large effect sizes.

Tests of Proportions

When comparing two proportions use

pwr.2p.test(h = , n = , sig.level =, power = )

where h is the effect size and n is the common sample size in each group.

Cohen h

Cohen suggests that h values of 0.2, 0.5, and 0.8 represent small, medium, and large effect sizes respectively.

For unequal n's use

pwr.2p2n.test(h = , n1 = , n2 = , sig.level = , power = )

To test a single proportion use

pwr.p.test(h = , n = , sig.level = power = )

For both two sample and one sample proportion tests, you can specify alternative="two.sided", "less", or "greater" to indicate a two-tailed, or one-tailed test. A two tailed test is the default.

Chi-square Tests

For chi-square tests use

pwr.chisq.test(w =, N = , df = , sig.level =, power = )

where w is the effect size, N is the total sample size, and df is the degrees of freedom. The effect size w is defined as

Cohen w

Cohen suggests that w values of 0.1, 0.3, and 0.5 represent small, medium, and large effect sizes respectively.

Some Examples

library(pwr)

# For a one-way ANOVA comparing 5 groups, calculate the

# sample size needed in each group to obtain a power of

# 0.80, when the effect size is moderate (0.25) and a

# significance level of 0.05 is employed.

pwr.anova.test(k=5,f=.25,sig.level=.05,power=.8)

# What is the power of a one-tailed t-test, with a

# significance level of 0.01, 25 people in each group,

# and an effect size equal to 0.75?

pwr.t.test(n=25,d=0.75,sig.level=.01,alternative="greater")

# Using a two-tailed test proportions, and assuming a

# significance level of 0.01 and a common sample size of

# 30 for each proportion, what effect size can be detected

# with a power of .75?

pwr.2p.test(n=30,sig.level=0.01,power=0.75)

Creating Power or Sample Size Plots

The functions in the pwr package can be used to generate power and sample size graphs.

# Plot sample size curves for detecting correlations of

# various sizes.

library(pwr)

# range of correlations

r <- seq(.1,.5,.01)

nr <- length(r)

# power values

p <- seq(.4,.9,.1)

np <- length(p)

# obtain sample sizes

samsize <- array(numeric(nr\*np), dim=c(nr,np))

for (i in 1:np){

for (j in 1:nr){

result <- pwr.r.test(n = NULL, r = r[j],

sig.level = .05, power = p[i],

alternative = "two.sided")

samsize[j,i] <- ceiling(result$n)

}

}

# set up graph

xrange <- range(r)

yrange <- round(range(samsize))

colors <- rainbow(length(p))

plot(xrange, yrange, type="n",

xlab="Correlation Coefficient (r)",

ylab="Sample Size (n)" )

# add power curves

for (i in 1:np){

lines(r, samsize[,i], type="l", lwd=2, col=colors[i])

}

# add annotation (grid lines, title, legend)

abline(v=0, h=seq(0,yrange[2],50), lty=2, col="grey89")

abline(h=0, v=seq(xrange[1],xrange[2],.02), lty=2,

col="grey89")

title("Sample Size Estimation for Correlation Studies\n

Sig=0.05 (Two-tailed)")

legend("topright", title="Power", as.character(p),

fill=colors)

Using with( ) and by( )

There are two functions that can help write simpler and more efficient code.

With

The with( ) function applys an expression to a dataset. It is similar to DATA= in SAS.

# with(data, expression)

# example applying a t-test to a data frame mydata

with(mydata, t.test(y ~ group))

By

The by( ) function applys a function to each level of a factor or factors. It is similar to BY processing in SAS.

# by(data, factorlist, function)

# example obtain variable means separately for

# each level of byvar in data frame mydata

by(mydata, mydata$byvar, function(x) mean(x))