Descriptive Statistics

R provides a wide range of functions for obtaining summary statistics. 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(mydata, mean, na.rm=TRUE)

Possible functions used in sapply include **mean, sd, var, min, max, median, range, and quantile**.

There are also numerous R functions designed to provide a range of descriptive statistics at once. For example

# mean,median,25th and 75th quartiles,min,max  
summary(mydata)  
  
# Tukey min,lower-hinge, median,upper-hinge,max  
fivenum(x)

Using the [Hmisc](http://cran.r-project.org/web/packages/Hmisc/index.html) package

library(Hmisc)  
describe(mydata)   
# n, nmiss, unique, mean, 5,10,25,50,75,90,95th percentiles   
# 5 lowest and 5 highest scores

Using the **[pastecs](http://cran.r-project.org/web/packages/pastecs/index.html)**package

library(pastecs)  
stat.desc(mydata)   
# nbr.val, nbr.null, nbr.na, min max, range, sum,   
# median, mean, SE.mean, CI.mean, var, std.dev, coef.var

Using the [psych](http://cran.r-project.org/web/packages/psych/index.html) package

library(psych)  
describe(mydata)  
# item name ,item number, nvalid, mean, sd,   
# median, mad, min, max, skew, kurtosis, se

Summary Statistics by Group

A simple way of generating summary statistics by grouping variable is available in the [psych](http://cran.r-project.org/web/packages/psych/index.html) package.

library(psych)  
describe.by(mydata, group,...)

The [doBy](http://cran.r-project.org/web/packages/doBy/index.html) package provides much of the functionality of SAS PROC SUMMARY. It defines the desired table using a model formula and a function. Here is a simple example.

library(doBy)  
summaryBy(mpg + wt ~ cyl + vs, data = mtcars,   
  FUN = function(x) { c(m = mean(x), s = sd(x)) } )  
# produces mpg.m wt.m mpg.s wt.s for each   
# combination of the levels of cyl and vs

# Frequencies and Crosstabs

This section describes the creation of frequency and contingency tables from categorical variables, along with tests of independence, measures of association, and methods for graphically displaying results.

## 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( )**.

# 2-Way Frequency Table   
attach(mydata)  
mytable <- table(A,B) # A will be rows, B will be columns   
mytable # print table   
  
margin.table(mytable, 1) # A frequencies (summed over B)   
margin.table(mytable, 2) # B frequencies (summed over A)  
  
prop.table(mytable) # cell percentages  
prop.table(mytable, 1) # row percentages   
prop.table(mytable, 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(A, B, C)   
ftable(mytable)

**Table ignores missing values.**To include **NA** as a category in counts, include the table option exclude=NULL if the variable is a vector. If the variable is a factor you have to create a new factor using newfactor <- factor(oldfactor, exclude=NULL).

### xtabs

The **xtabs( )** function allows you to create crosstabulations using formula style input.

# 3-Way Frequency Table  
mytable <- xtabs(~A+B+c, data=mydata)  
ftable(mytable) # print table   
summary(mytable) # chi-square test of indepedence

If a variable is included on the left side of the formula, it is assumed to be a vector of frequencies (useful if the data have already been tabulated).

### Crosstable

The **CrossTable( )** function in the [gmodels](http://cran.r-project.org/web/packages/gmodels/index.html) package produces crosstabulations modeled after PROC FREQ in **SAS** or CROSSTABS in SPSS. It has a wealth of options.

# 2-Way Cross Tabulation  
library(gmodels)  
CrossTable(mydata$myrowvar, mydata$mycolvar)

There are options to report percentages (row, column, cell), specify decimal places, produce Chi-square, Fisher, and McNemar tests of independence, report expected and residual values (pearson, standardized, adjusted standardized), include missing values as valid, annotate with row and column titles, and format as **SAS** or **SPSS** style output!   
See **help(CrossTable)** for details.

## Tests of Independence

### Chi-Square Test

For 2-way tables you can use **chisq.test(**mytable**)** to test independence of the row and column variable. By default, the p-value is calculated from the asymptotic chi-squared distribution of the test statistic. Optionally, the p-value can be derived via Monte Carlo simultation.

### Fisher Exact Test

**fisher.test(**x**)** provides an exact test of independence. x is a two dimensional contingency table in matrix form.

### ****Mantel****-****Haenszel**** test

Use the **mantelhaen.test(**x**)** function to perform a Cochran-Mantel-Haenszel chi-squared test of the null hypothesis that two nominal variables are conditionally independent in each stratum, assuming that there is no three-way interaction. x is a 3 dimensional contingency table, where the last dimension refers to the strata.

### Loglinear Models

You can use the**loglm( )** function in the **MASS** package to produce log-linear models. For example, let's assume we have a 3-way contingency table based on variables A, B, and C.

library(MASS)  
mytable <- xtabs(~A+B+C, data=mydata)

We can perform the following tests:

**Mutual Independence**: A, B, and C are pairwise independent.loglm(~A+B+C, mytable)

**Partial Independence**: A is partially independent of B and C (i.e., A is independent of the composite variable BC).loglin(~A+B+C+B\*C, mytable)

**Conditional Independence:** A is independent of B, given C.loglm(~A+B+C+A\*C+B\*C, mytable)

**No Three-Way Interaction**loglm(~A+B+C+A\*B+A\*C+B\*C, mytable)

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/covariances among numeric variables in   
# data frame mtcars. Use listwise deletion of missing data.   
cor(mtcars, use="complete.obs", method="kendall")   
cov(mtcars, use="complete.obs")

Unfortunately, neither **cor( )**or **cov( )** produce tests of significance, although you can use the **cor.test( )**function to test a single correlation coefficient.

The **rcorr( )** function in the [Hmisc](http://cran.r-project.org/web/packages/Hmisc/index.html) package produces correlations/covariances and significance levels for pearson and spearman correlations. However, input must be a matrix and pairwise deletion is used.

# 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)

Other Types of Correlations

# polychoric correlation  
# x is a contingency table of counts  
library(polycor)  
polychor(x)   
  
# 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  
library(polycor)  
hetcor(x)   
  
  
# partial correlations  
library(ggm)  
data(mydata)  
pcor(c("a", "b", "x", "y", "z"), var(mydata))  
# partial corr between a and b controlling for x, y, z

# 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(y~x) # where y is numeric and x is a binary factor

# independent 2-group t-test  
t.test(y1,y2) # where y1 and y2 are numeric

# paired t-test  
t.test(y1,y2,paired=TRUE) # where y1 & y2 are numeric

# one sample t-test  
t.test(y,mu=3) # Ho: mu=3

You can use the **var.equal = TRUE** option to specify equal variances and a pooled variance estimate. You can use the **alternative="less"** or **alternative="greater"** option to specify a one tailed test.

# 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

For the wilcox.test you can use the **alternative="less"** or **alternative="greater"** option to specify a one tailed test.

[Parametric](https://www.statmethods.net/stats/ttest.html) and [resampling](https://www.statmethods.net/stats/resampling.html) alternatives are available.

The package [pgirmess](https://www.rdocumentation.org/packages/pgirmess/versions/1.6.5) provides nonparametric multiple comparisons. (Note: This package has been withdrawn but is still available in the CRAN archives.)

library(npmc)  
npmc(x)   
# where x is a data frame containing variable 'var'   
# (response variable) and 'class' (grouping variable)

# Multiple (Linear) Regression

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

## Fitting the Model

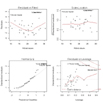
# Multiple Linear Regression Example   
fit <- lm(y ~ x1 + x2 + x3, data=mydata)  
summary(fit) # show results

# 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

## Diagnostic Plots

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

# diagnostic plots   
layout(matrix(c(1,2,3,4),2,2)) # optional 4 graphs/page   
plot(fit)

[](https://www.statmethods.net/stats/images/regdiag.png) click to view

For a more comprehensive evaluation of model fit see [regression diagnostics](https://www.statmethods.net/stats/rdiagnostics.html) or the exercises in [this interactive course on regression.](https://www.datacamp.com/courses/correlation-and-regression)

## Comparing Models

You can compare nested models with the anova( ) function. The following code provides a simultaneous test that x3 and x4 add to linear prediction above and beyond x1 and x2.

# compare models  
fit1 <- lm(y ~ x1 + x2 + x3 + x4, data=mydata)  
fit2 <- lm(y ~ x1 + x2)  
anova(fit1, fit2)

## Cross Validation

You can do[K-Fold cross-validation](http://en.wikipedia.org/wiki/Cross-validation)using the cv.lm( ) function in the [DAAG](http://cran.r-project.org/web/packages/DAAG/index.html) package.

# K-fold cross-validation  
library(DAAG)  
cv.lm(df=mydata, fit, m=3) # 3 fold cross-validation

Sum the MSE for each fold, divide by the number of observations, and take the square root to get the cross-validated standard error of estimate.

You can assess **R2 shrinkage** via K-fold cross-validation. Using the **crossval()**function from the [**bootstrap**](http://cran.r-project.org/web/packages/bootstrap/index.html)package, do the following:

# Assessing R2 shrinkage using 10-Fold Cross-Validation   
  
fit <- lm(y~x1+x2+x3,data=mydata)   
  
library(bootstrap)  
# define functions   
theta.fit <- function(x,y){lsfit(x,y)}  
theta.predict <- function(fit,x){cbind(1,x)%\*%fit$coef}   
  
# matrix of predictors  
X <- as.matrix(mydata[c("x1","x2","x3")])  
# vector of predicted values  
y <- as.matrix(mydata[c("y")])   
  
results <- crossval(X,y,theta.fit,theta.predict,ngroup=10)  
cor(y, fit$fitted.values)\*\*2 # raw R2   
cor(y,results$cv.fit)\*\*2 # cross-validated R2

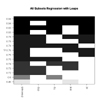
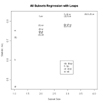
## Variable Selection

Selecting a subset of predictor variables from a larger set (e.g., stepwise selection) is a controversial topic. You can perform stepwise selection (forward, backward, both) using the **stepAIC( )** function from the **MASS**package. **stepAIC( )**performs stepwise model selection by exact AIC.

# Stepwise Regression  
library(MASS)  
fit <- lm(y~x1+x2+x3,data=mydata)  
step <- stepAIC(fit, direction="both")  
step$anova # display results

Alternatively, you can perform all-subsets regression using the **leaps( )** function from the [leaps](http://cran.r-project.org/web/packages/leaps/index.html) package. In the following code nbest indicates the number of subsets of each size to report. Here, the ten best models will be reported for each subset size (1 predictor, 2 predictors, etc.).

# All Subsets Regression  
library(leaps)  
attach(mydata)  
leaps<-regsubsets(y~x1+x2+x3+x4,data=mydata,nbest=10)  
# view results   
summary(leaps)  
# plot a table of models showing variables in each model.  
# models are ordered by the selection statistic.  
plot(leaps,scale="r2")  
# plot statistic by subset size   
library(car)  
subsets(leaps, statistic="rsq")

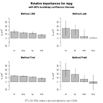
[](https://www.statmethods.net/stats/images/leaps1.png) [](https://www.statmethods.net/stats/images/leaps2.png) click to view

Other options for **plot( )** are bic, Cp, and adjr2. Other options for plotting with**subset( )** are bic, cp, adjr2, and rss.

## Relative Importance

The [relaimpo](http://cran.r-project.org/web/packages/relaimpo/index.html) package provides measures of relative importance for each of the predictors in the model. See help(calc.relimp) for details on the four measures of relative importance provided.

# Calculate Relative Importance for Each Predictor  
library(relaimpo)  
calc.relimp(fit,type=c("lmg","last","first","pratt"),  
   rela=TRUE)  
  
# Bootstrap Measures of Relative Importance (1000 samples)   
boot <- boot.relimp(fit, b = 1000, type = c("lmg",   
  "last", "first", "pratt"), rank = TRUE,   
  diff = TRUE, rela = TRUE)  
booteval.relimp(boot) # print result  
plot(booteval.relimp(boot,sort=TRUE)) # plot result

[](https://www.statmethods.net/stats/images/relimp.png) click to view

## Graphic Enhancements

The [car](http://cran.r-project.org/web/packages/car/index.html) package offers a wide variety of plots for regression, including added variable plots, and [enhanced diagnostic](https://www.statmethods.net/stats/rdiagnostics.html) and [Scatterplots](https://www.statmethods.net/graphs/scatterplot.html).

## Going Further

### Nonlinear Regression

The **nls** package provides functions for nonlinear regression. See John Fox's [Nonlinear Regression and Nonlinear Least Squares](https://socserv.socsci.mcmaster.ca/jfox/Books/Companion/appendix/Appendix-Nonlinear-Regression.pdf) for an overview. Huet and colleagues' [Statistical Tools for Nonlinear Regression: A Practical Guide with S-PLUS and R Examples](http://www.amazon.com/Statistical-Tools-Nonlinear-Regression-Statistics/dp/0387400818) is a valuable reference book.

### Robust Regression

There are many functions in R to aid with robust regression. For example, you can perform robust regression with the **rlm( )** function in the **MASS** package. John Fox's (who else?) [Robust Regression](https://socserv.socsci.mcmaster.ca/jfox/Books/Companion/appendix/Appendix-Robust-Regression.pdf) provides a good starting overview. The UCLA Statistical Computing website has [Robust Regression Examples](http://www.ats.ucla.edu/stat/R/dae/rreg.htm).

The [robust](http://cran.r-project.org/web/packages/robust/index.html) package provides a comprehensive library of robust methods, including regression. The [robustbase](http://cran.r-project.org/web/packages/robustbase/index.html) package also provides basic robust statistics including model selection methods. And David Olive has provided an detailed online review of [Applied Robust Statistics](http://www.math.siu.edu/olive/ol-bookp.htm) with sample R code.

# Regression Diagnostics

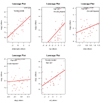
An excellent review of regression diagnostics is provided in John Fox's aptly named [Overview of Regression Diagnostics.](http://socserv.socsci.mcmaster.ca/jfox/Courses/Brazil-2009/index.html) Dr. Fox's [car](http://cran.r-project.org/web/packages/car/index.html) package provides advanced utilities for 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!

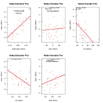
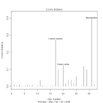
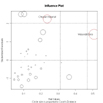
## Outliers

# 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

[](https://www.statmethods.net/stats/images/leverage.png) click to view

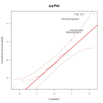
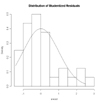
## Influential Observations

# 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" )

[](https://www.statmethods.net/stats/images/avplot.png) [](https://www.statmethods.net/stats/images/CooksD.png) [](https://www.statmethods.net/stats/images/influenceplot.png) click to view

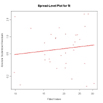
## Non-normality

# 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)

[](https://www.statmethods.net/stats/images/qqplot2.png) [](https://www.statmethods.net/stats/images/studresid.png) click to view

## Non-constant Error Variance

# Evaluate homoscedasticity  
# non-constant error variance test  
ncvTest(fit)  
# plot studentized residuals vs. fitted values   
spreadLevelPlot(fit)

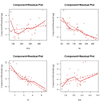
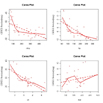
[](https://www.statmethods.net/stats/images/spread.png) click to view

## 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)

[](https://www.statmethods.net/stats/images/crplot.png) [](https://www.statmethods.net/stats/images/ceres.png) click to view

## Non-independence of Errors

# Test for Autocorrelated Errors  
durbinWatsonTest(fit)

## Additional Diagnostic Help

The **gvlma( )** function in the [gvlma](http://cran.r-project.org/web/packages/gvlma/index.html) 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](http://personality-project.org/r/r.guide/r.anova.html#oneway) section of the [Personality Project](https://personality-project.org/index.html). (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](https://www.statmethods.net/input/datatypes.html).

# 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.](http://personality-project.org/r/)

# 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](https://www.statmethods.net/stats/anovaAssumptions.html).

### 3. Evaluate Model Effects

**WARNING**: R provides [Type I sequential SS](http://afni.nimh.nih.gov/sscc/gangc/SS.html), not the default [Type III marginal SS](http://afni.nimh.nih.gov/sscc/gangc/SS.html)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](https://www.statmethods.net/stats/nonparametric.html) and [resampling](https://www.statmethods.net/stats/resampling.html) alternatives are available.

## Multiple Comparisons

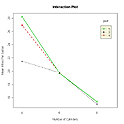
You can get Tukey HSD tests using the function below. By default, it calculates post hoc comparisons on each factor in the model. You can specify specific factors as an option. **Again, remember that results are based on Type I SS!**

# Tukey Honestly Significant Differences  
TukeyHSD(fit) # where fit comes from aov()

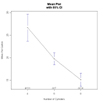
## Visualizing Results

Use [box plots](https://www.statmethods.net/graphs/boxplot.html)and [line plots](https://www.statmethods.net/graphs/line.html) 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](http://cran.r-project.org/web/packages/gplots/index.html)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")

[](https://www.statmethods.net/stats/images/interaction.jpg) 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")

[](https://www.statmethods.net/stats/images/meanplot.jpg) 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

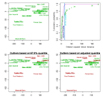
In classical parametric procedures we often assume normality and constant variance for the model error term. Methods of exploring these assumptions in an ANOVA/ANCOVA/MANOVA framework are discussed here. Regression diagnostics are covered under [multiple linear regression.](https://www.statmethods.net/stats/regression.html)

## 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](http://cran.r-project.org/web/packages/mvoutlier/index.html) 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.

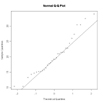
# Detect Outliers in the MTCARS Data  
library(mvoutlier)  
outliers <-   
aq.plot(mtcars[c("mpg","disp","hp","drat","wt","qsec")])  
outliers # show list of outliers

[](https://www.statmethods.net/stats/images/outliers1.png) 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)

[](https://www.statmethods.net/stats/images/qqplot1.jpg) 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 testingnormality are available in [nortest](http://cran.r-project.org/web/packages/nortest/index.html) package.

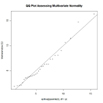
## Multivariate Normality

MANOVA assumes **multivariate normality**. The function**mshapiro.test( )** in the [mvnormtest](http://cran.r-project.org/web/packages/mvnormtest/index.html) 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)

[](https://www.statmethods.net/stats/images/mahalanobis.jpg) 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)

# Resampling Statistics

The [coin](http://cran.r-project.org/web/packages/coin/index.html) 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](http://finzi.psych.upenn.edu/R/library/coin/doc/coin.pdf) for details.

In the examples below, **lower case** letters represent numerical variables and **upper case**letters represent categorical [factors](https://www.statmethods.net/input/datatypes.html). 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))

# 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:

1. sample size
2. effect size
3. significance level = P(Type I error) = probability of finding an effect that is not there
4. 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](http://cran.r-project.org/web/packages/pwr/index.html) package develped by Stéphane Champely, impliments power analysis as outlined by [Cohen (!988)](http://www.amazon.com/Statistical-Power-Analysis-Behavioral-Sciences/dp/0805802835). 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](http://en.wikipedia.org/wiki/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.)](https://www.datacamp.com/courses/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.

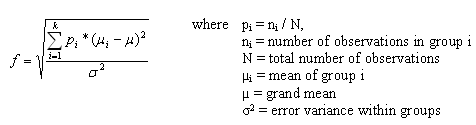
## ANOVA

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 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.

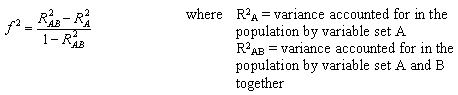
## Linear Models

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



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 = )**

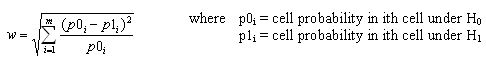
For both two sample and one sample proportion tests, you can specify alternative="two.sided", "less", or "greater" to indicate a two-tailed, or one-tailed test. A two tailed test is the default.

## 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 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](http://cran.r-project.org/web/packages/pwr/index.html) 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)