**#\*\*\*Basics\*\*\***

#set working directory:

setwd('U:/documents/Stats/NotPorn') #package ‘base’

getwd() #checks working directory #package ‘base’

#loading and installing packages

install.packages("namePackage") #package ‘base’

library(namePackage) #package ‘base’

#opening files:

read.csv(‘filename’, sep = ';') #package ‘base’

read.spss(‘filename’, rm.na = TRUE, to.data.frame = TRUE)

* rm.na for deleting missing cases
* to.data.frame to make it a data-frame
* #package ‘foreign’

#writing files

write.csv(Dataframename, file = “chooseaname.csv”) #package ‘base’

###checking out your data###

#checking labels & names

#gets variable labels from spss

attr(dataframe, “variable.labels”) #package ‘base’

#checks and sets names of columns in dataframe

names(dataframe) #package ‘base’

#ways of removing missing data

#removes last row from ‘dataframe’, and saves it to another file, ‘dataframe2’.

dataframe2 <- dataframe[-nrow(dataframe),]

#removes row of missing data from ‘dataframe’, and saves it to another dataframe, ‘dataframe2’.

dataframe2 <- na.omit(dataframe)

#checking first 5 & last 5 rows of a data set

#checks first 5 rows ‘dataframe’

head(dataframe) #package ‘base’

#checks last 5 rows of ‘dataframe’

tail(dataframe) #package ‘base’

#checking descriptives

#quick summary of your variables

summary(dataframe) #package ‘base’

#descriptive statistics

describe(dataframe) #package ‘psych’

#describe a variable by a grouping variable

describeBy(dataframe, group = NULL) #package ‘psych’

#more comprehensive descriptive statistics. Useful for 2SE skew & kurt

stat.desc(dataframe) #package ‘pastecs’

#visual check of distribution (for normality, skewness, & kurtosis)

densityplot(dataframe$variable)#package ‘lattice’

#centering variables

scale(dataframe$variable, scale = FALSE))) #package ‘base’

#you could standardize the variable to check for outliers

scale(dataframe$variable) #package ‘base’

length(which(abs(dataframe$variable) > 3)) #package ‘base’

##reshaping dataframe##

#creating a long datafile using melt()

melt( #package ‘reshape’

dataframe,

id = c(‘ppcode’, ‘other factors’), #variables measured one time

measured = c(‘score1’, ‘score2’), #repeated measures

variable\_name = ‘time’) #name new repeated measures

#creating a long datafile using reshape()

(another way to refer to the names of variables is used, both R possible)

reshape( #package ‘reshape’

dataframe,

idvar = names(dataframe[1]), #name identity variable

varying = names(dataframe[3:5]), #repeated measures

timevar = 'time', #name new repeated measures variable

v.names = 'score\_at\_time', #name value of repeated measures variable

direction = 'long' #shape of dataframe

)

#creating a wide datafile using reshape()

reshape( #package ‘reshape’

dataframe\_long,

direction = 'wide')

#creating a wide datafile using cast()

cast(dataframe\_long, pp\_code ~ variable)

# left of the ~ is the variable that identified in the molten (=stacked) data frame which observations belong to the same participant; right of the ~ is the variable that identifies the new columns that will be created

#creating values

#round digits in a vector to a certain amount of digits

round(c(), digits = 0) #package ‘base’

#create vector with 41 values that are normally distributed with mean = 100 and sd = 15.

rnorm(41, mean = 100, sd = 15) #package ‘base’

#sampling data

#there are different ways of sampling data/ selecting subsets

#in this case every subgroup mentioned in the vector c() will be included in the new subset dataframe

sampleddata <- dataframe[(which(dataframe$groupingvariable %in% c("subgroup1", "subgroup2", "subgroup3", etc...))),]

#using subset to select specific columns

sampleddata <- subset(dataframe, select = startcolumn:endcolumn)

#or

sampleddata <- subset(dataframe, select = c(column1,column2, column3))

#ggplot summary

#package ‘ggplot2’

#creating a ggplot object

#the aes defines the looking of the plot: what variables are you going to use but it does not really do anything

ggobject <- ggplot(dataframe, aes(xvariable, yvariable))

#to have something be plotted you have to add geoms. there are different ones

#geom\_smooth will add a line, method will specify the line’s shape, color can be used to plot different lines for different groups. alpha specifies the transparency of the confidence interval of the line and size specifies the size of it

ggobject <- ggobject + geom\_smooth(method = lm, aes(color =groupingvariable), alpha = .1, size = 1)

#geom\_point will give you a scatter plot. You can specify shape or color to seperate groups

ggobject <- ggobject + geom\_point(aes(shape = class))

#to plot summaries of groups (e.g. means) you can specify stat\_summary. fun.y gives what should be summarized, geom specifies how it should be displayed, aes are used to differentiate between groups.

ggobject <- ggobject + stat\_summary(

fun.y = mean,

geom = “bar”,

aes(fill = groupingvariable)

)

#to add error-bars you need a slightly different stat\_summary. You have to use this together with another mean summary like the one above

ggobject <- ggobject + stat\_summary(

fun.y = mean\_cl\_normal,

geom = “errorbar”,

width = 0.2)

#to plot more than 1 dependent variables you have to use reshape() to create a long-format data file, that distinguishes the dependent variable in a factor that can be used in aes just like other grouping variables. In this case you create an ggobject specifying the x and y axis and then a stat\_summary geom separating the dependent variables. Position = “dodge” is necessary in those cases.

ggobject <- ggplot(dataframe, aes(groupingvariable, variable)) + stat\_summary(fun.y = mean, geom = “bar”, aes(fill = variablespecifier), position = “dodge”)

#you can also add an errorbar. The position has to be dodged and 0.9 is the fixed right value.

ggobject <- ggobject + stat\_summary(fun.y = mean\_cl\_normal, geom = “errorbar”, position=position\_dodge(width=0.9), width = 0.2)

#univariate (& multivariate correlations)

#quick and dirty correlation (no p-values)

cor(data$V1,data$V2, method = "pearson")) #package ‘base’

#correlation matrix between 2 variables, without significance.

#choice between Pearson or Spearman.

rcorr( #package ‘Hmisc’

dataframe$variable,

dataframe$variable2,

type = “Pearson” / “Spearman”)

#correlation matrix between >2 variables, with significance (Bill’s favorite).

rcor.test(dataframe[,startingcolumn:howevermuch]) #package ‘ltm’

#if you want to adjust for multiple comparisons (normally not done)

rcor.test(dataframe[,startingcolumn:howevermuch], p.adjust = TRUE, p.adjust.method = "bonferroni")

#fancy scatterplot matrices

#can also use abbreviated form, spm()

#reg.line for regression line

#spread for that small area that tells you how much the points spread

#smooth to add a Loess Fitline

scatterplotMatrix(~variable1 + variable2 + etc…, data = dataframe, reg.line = F, spread = F, smooth) #package ‘car’

#partial correlations

pcor() #package ´ggm´

pcor.test() #package ´ggm´

#t-tests

#t-tests for comparing 1 variable between 2 groups.

t.test(dataframe$variable ~ dataframe$goupingvariable)#package ‘base’

#within subjects t-test

with(dataframe, t.test(score1, score2, paired = TRUE))#package ‘base’

#another way of doing the same t-test

t.test(dataframe$score1, dataframe$score2, paired = TRUE)#package ‘base’

**#Multiple Linear Regression**

**#assumptions about univariate distributions**

#normality of continuous variables: check skewness and kurtosis. stat.desc gives you skew.2SE and kurt.2SE, which absolute value may not exceed 1.

stat.desc(dataframe, norm = TRUE) #package ‘pastecs’

#if (variables are skewed and/or kurtosed){you can decide to transform the variable.

e.g. dataframe$new\_trans <- -1\*(1/(dataframe$oldvariable+1)^2)}

#assessing normality using the shapiro wilk test

#you want this to be non-significant

shapiro.test(dataframe$variable) #package ‘base’

shapiro.test(dataframe$DV)

#interpret in conjunction with histograms in large samples

#outliers

#create Zscores

dataframe$zscore\_variable1 <- scale(dataframe$variable1) #package ‘base’

#check whether any scores are more than 3 SD from the average, these are outliers. The following command returns the amount of outliers.

length(which(abs(dataframe$zscore\_variable1) > 3))

#influential cases

#proportions of standardized residuals larger than |±2| → around 5%

#proportions of standardized residuals larger than |± 2.5| → around 1%

#Every case with a residual > |± 3| could be an outlier

#calculating standardized residuals

dataframe$res\_model <- rstandard(model)

#percentage of residuals > |2|

print(paste("percentage of standardized residuals +- 2 in model:", (length(which(abs(dataframe$res\_model) > 2)) / length(dataframe$res\_model)) \* 100))

#percentage of residuals >2.5

print(paste("percentage of standardized residuals +- 2.5 in model:", (length(which(abs(dataframe$res\_model) > 2.5)) / length(dataframe$res\_model)) \* 100))

#percentage of residuals >3

print(paste("percentage of standardized residuals +- 3 in model:", (length(which(abs(dataframe$res\_model) > 3)) / length(dataframe$res\_model)) \* 100))

#Cook’s distance values should be assessed for cases with standardized residuals of 2 or higher. If > 1 they indicate influential cases.

#creating cook’s distance values.

dataframe$COOKmodel <- cooks.distance(model) #package ‘base’

#creating a vector with only the cook’s distances for standardized residuals > 2

cooksvec <- c(dataframe$COOKmodel[which(abs(EASreg$res\_lm1s) > 2)])

#printing amount of cases that are influential

length(which(abs(dataframe$COOKmodel) > 1))

#other ways of looking at influential cases (though how is a bit vague)

#dfbeta influence

dfbeta(model)>1 #package ‘base’

#DFFit

dffits(myModel) #package ‘base’

#hat values, leverage

hatvalues(myModel) #package ‘base’

#covariance ratio

covratio(myModel) #package ‘base’

**#assumptions about the model**

#independence of errors

durbinWatsonTest(model) #package ‘car’

or

dwt(model) #package ‘car’

* values should be as close to 2 as possible, if smaller than 1 or bigger than 3 assumption may be violated.
* there is also a p-value that should be **non-**significant.

#multicollinearity → correlations among your predictors

#using Variance inflation factor #package ‘car’

vif(model) #higher than 10 indicates multicollinearity

1/vif(model) #--> tolerance, needs to be higher than .2

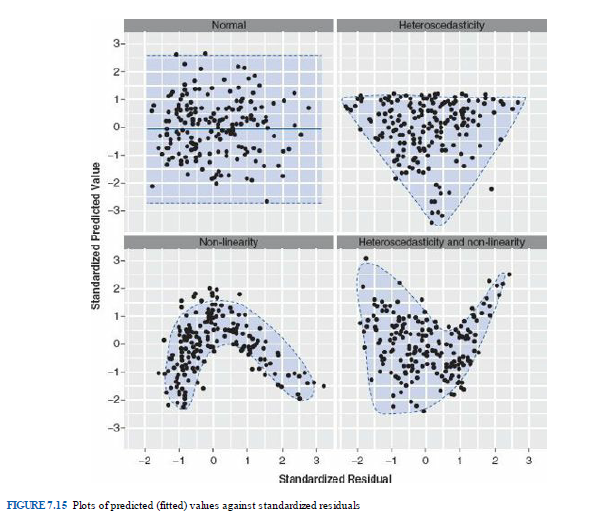
#Tolerance 0.2 to 0.1 à potential problem

#Tolerance < 0.1 à serious problem

#homoscedacity

use plot()

plot(mymodel) #package ‘base’

* first plot: residuals vs fitted values, use for
  + linearity homoscedacity (see plot below for the indications of violations in this plot)
* The residuals "bounce randomly" around the 0 line. This suggests that the assumption that the relationship is linear is reasonable.
* The residuals roughly form a "horizontal band" around the 0 line. This suggests that the variances of the error terms are equal.
* No one residual "stands out" from the basic random pattern of residuals. This suggests that there are no outliers.
* There is also a package to assess this check my R scripts.
  + 
* second plot: Q-Q plot
  + normality of residuals: if they lie on a straight line, they are normal
* third plot: scale-location → ignore this.
* fourth plot: leverage → ignore this too :D

→ the 1st and 2nd plot you get from plot() can also be made using

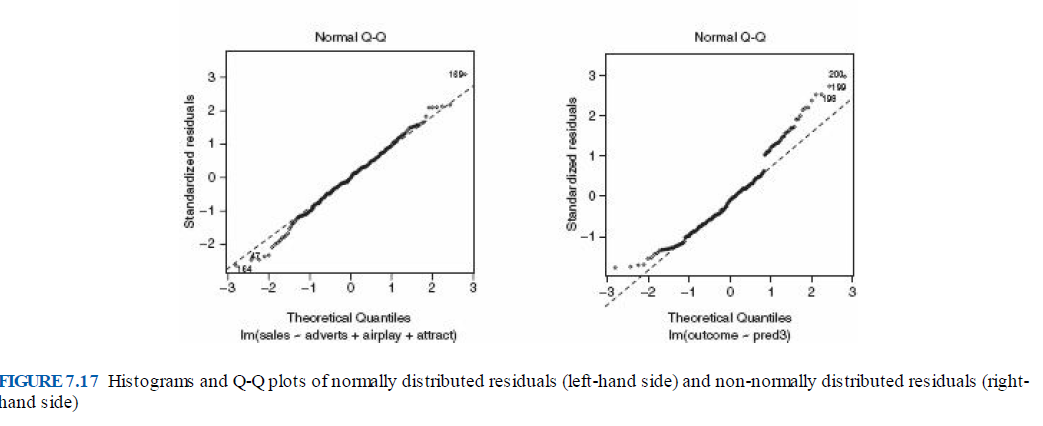
1st

plot(model$fitted.values, residuals(model))

2nd

qqnorm(dataframe$residuals\_model)

qqline(dataframe$residuals\_model)



**#main analyses**

#creating a model by using the lm() function. na.action is default set to deleting missing values casewise. na.exclude does also do this but will substitute the rows with NAs for extractor functions like residuals() or whatever. But we probably do not have to know this.

model <- lm(DV ~ IV, data = dataframe, na.action = na.exclude) #package ‘base’

#in case of interactions:

center IVs if you are looking at interactions to prevent multicollinearity. na.rm removes missing values.

dataframe$c\_variable <-

dataframe$variable – mean(dataframe$variable, na.rm = TRUE)

model\_c <- lm(DV ~ c\_IV, data = dataframe) #package ‘base’

#multiple IVs

model <- lm(DV ~ c\_V1\*c\_V2, data = dataframe)

v1 + v2 → main effect of v1 and v2

v1 : v2 → interaction between v1 and v2

v1 \* v2 → all possible main effects and interactions

#other possibility: compute the interaction yourself

#multiply the centered predictors

dataframe$int\_cV1xcV2 <- dataframe$c\_V1 \* dataframe$c\_V2

#then just add that to the model as an IV

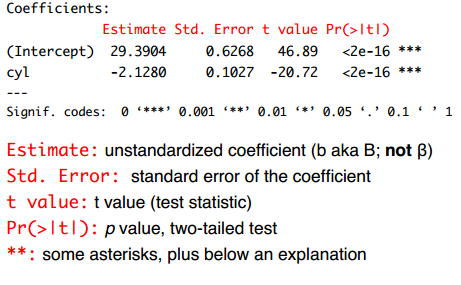
model <- lm(DV ~ V1 + V2 + int\_cV1xcV2, data = dataframe)

#checking the general model estimates etc. Both of the following commands do the same in this case as we are dealing with a linear model object, i.e. we created it with lm(). You will see unstandardized coefficients, standard errors of the coefficients, t-values and p-values to assess the significance of the predictors (see below).

summary(model)

or

summary.lm(model)

* Interpretation: The intercept is the mean where the predictors are 0. The unstandardized estimates indicate how many units the dependent variable will change with a one unit change of this predictor.

#for standardized coefficients.

lm.beta(model) #package ‘QuantPsyc’

#other option: standardize IVs and DVs and run model again

dataframe$z\_IV1 <- scale(dataframe$IV1) #zscores variable1

#package ‘base’

#confidence intervals. level = .975 is comparable to a .05 p-value significance level. If the displayed interval does not include 0 your test is significant.

confint(model, level = .975) #package ‘base’

**#Hierarchical Regression**

#run one model with first predictors, then add more predictors and compare the two models

model1 <- lm(DV ~ c\_IV1 + c\_IV2, data = mpg) #just main effects

model2 <- lm(DV ~ c\_IV1 \* c\_IV2, data = mpg) #includes mains and interaction

#compare the two models

anova(model1, model2) #package ‘base’

#if p<.05, the second model explains more variance than the first one.

#check change in variance explained (this is not part of the exam)

lm.deltaR2(model1, model2) #package ‘lmSupport’

#plot the influence of one predictor on the model

plot(effect("c\_V1", model)) #package ‘effects’

#to plot the results you can use ggplot and add a lm geom\_smooth (see ggplot() from package ggplot2)

**Logistic Regression**

**#assumptions**

Assumptions can only be checked **after** creating the model.

#linearity: there is a linear relationship between any continuous predictor & the logit of its outcome. Check by examining the interaction between continuous predictor & its log transformation.

#independence of errors

dwt(model) #package ‘car’

* values should be as close to 2 as possible (1 < x < 3).
* p-value should **not** be significant.

#absence of multivariate outliers & influential cases

#first, check standardized residuals

dataframe$residuals <- rstandard(model) #package ‘car’

densityplot(dataframe$residuals) #package ‘lattice’

* residuals > 2 SD should be less than 5%.
* residuals > 2.5 SD should be less than 1%.
* no residuals > 3 SD.
* violation indicates multivariate outliers.

#second, check dfbetas

dataframe$dfbetas <- dfbeta(model) #package ‘car’

* no absolute values should be > 1.
* violation indicates presence of influential cases (cases that affect the model significantly more than others).

#third, check leverage values

dataframe$leverage <- hatvalues(model) #package ‘car’

densityplot(dataframe$leverage) #package ‘lattice’

* check plot for influential cases.

**#analysis**

Bill uses the following commands:

#STEP 1: import the data frame from SPSS

dataframe <- read.spss(“filename”, to.data.frame = T, use.value.labels = T)

#package ‘foreign’

#STEP 2: create a new data frame with relevant variables only

dataframesub <- subset(dataframe, (conditions), select = (variable names))

#example: creating subset with only “id”, “sex”, “satjob”, and “life”, without missing values

dataframesub <- subset(dataframe, satjob != FALSE & life != FALSE, select = c(“id”, “sex”, “satjob”, “life”))

#STEP 3: check & specify coding for variables used in model

It is recommended to use effects coding for logistic regression.

contrasts(dataframe$variable) <- contr.sum(level) #package ‘base’

#example: effects coding for “sex”, “satjob”, and “life”.

contrasts(dataframesub$sex) <- contr.sum(2)

contrasts(dataframesub$satjob) <- contr.sum(2)

contrasts(dataframesub$life) <- contr.sum(3)

#STEP 4: create the logistic regression model

model <- glm(variables, data = dataframe, family = binomial()) #package ‘base’

#example: are sex and lifestyles predictors of job satisfaction?

model <- glm(satjob ~ sex + life, data = dataframesub, family = binomial())

#STEP 5: check assumptions

dwt(model) #package ‘car’

dataframesub$residuals <- rstandard(model) #package ‘car’

densityplot(dataframesub$residuals) #package ‘lattice’

dataframesub$dfbeta <- dfbeta(model) #package ‘car’

dataframesub$leverage <- hatvalues(model) #package ‘car’

densityplot(dataframesub$leverage) #package ‘lattice’

#STEP 6: check model fit

modelchi <- model$null.deviance - model$deviance #package ‘car’

chidf <- model$df.null - model$df.residual #package ‘car’

chiprob <- 1 - pchisq(modelchi, chidf) #package ‘car’

modelchi; chidf; chiprob

#STEP 7: calculate R-square

R2.hl <- modelchi/model$null.deviance #package ‘car’

R.cs <- 1 - exp((model$deviance - model$null.deviance) / (samplesize))

#package ‘car’

R.n <- R.cs / (1 - (exp(-(model$null.deviance / (samplesize)))))

#package ‘car’

R2.hl; R.cs; R.n

#STEP 8: examine results

summary(model) #package ‘car’

#STEP 9: compute odds ratio and CI

exp(model$coefficients) #package ‘base’

exp(confint(model))

#STEP 10: compare with the interaction model

model2 <- glm(satjob ~ sex \* life, data = dataframesub, family = binomial())

## repeat STEP 5, 6, & 7 to check model diagnostics

#compare models

anova(model, model2)

**AN(C)OVA**

#creating factors

#example! :)

W4$STYLE = (W4$RESPONSE \* 10) + W4$DEMAND #create new variable

#recode variables to make sure authoritative has lowest score

W4$STYLE[W4$STYLE==00] <- 4 #indifferent

W4$STYLE[W4$STYLE==01] <- 3 #authoritarian

W4$STYLE[W4$STYLE==10] <- 2 #permissive

W4$STYLE[W4$STYLE==11] <- 1 #authoritative

#label values

W4$STYLE <- factor(

W4$STYLE,

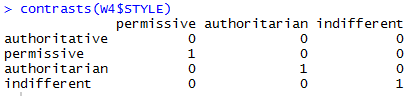
levels = c(1,2,3,4),

labels = c("authoritative", "permissive", "authoritarian", "indifferent")

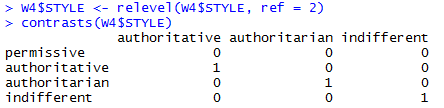
)

#for changing the reference category( in this example to level 2 = permissive)

#original contrast



use command → relevel(factor, ref = 2)



**Assumptions**

#Homogeneity of variance between groups

#You only have to perform one of the tests (normally Levene cause it’s Bill’s favorite)

#Levene’s test

leveneTest(data$variable, group = data$groupvariable) #package ‘car’

#Hartley’s F-max

vars <- with(dataframe, tapply(contIV, bsf, var))

Fmax <- max(vars)/min(vars)

levelsbsf <- levels(dataframe[,bsf])

print(

paste(

"Approximate p-value is",

1-pmaxFratio( #package ‘SuppDists’

Fmax,

length(bsf),

(min(

length(

which(

dataframe[bsf]==levelsbsf[1])),

length(

which(dataframe[bsf]==levelsbsf[2]))

)-1))))

#Independence of observations

is not assessed statistically: It’s just the assumption that every person is only included once in the data set, i.e. you’re not dealing with a repeated measures design.

#Normal distributions within groups

by(dataframe$V1, dataframe$bsf, densityplot)

by(dataframe$V1, dataframe$bsf, histogram)

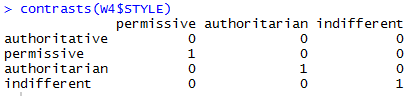
#contrasts! :D

for some good explanations:

http://www.ats.ucla.edu/stat/r/library/contrast\_coding.htm

#default setting is **dummy**

→ compares each level of the categorical variable to a fixed reference level.



#create contrast for **deviation/effects**

devcont <- matrix(nrow=4, ncol=3)

#create effects contrast for 4 levels with first level as reference

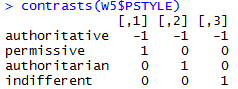
devcont[,1] <- c(-1,1,0,0)

devcont[,2] <- c(-1,0,1,0)

devcont[,3] <- c(-1,0,0,1)

#set contrasts

contrasts(dataframe$bsf) <- devcont

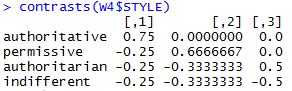


#create contrast for **helmert**

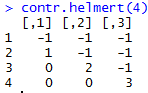
compares each level of a categorical variable to the mean of the subsequent levels

my.helmert <- matrix(c(3/4, -1/4, -1/4, -1/4, 0, 2/3, -1/3, -1/3, 0, 0, 1/2, -1/2), ncol = 3)

contrasts(dataframe$V1) <- my.helmert

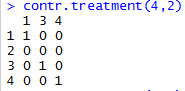
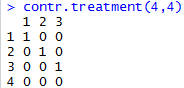


#!Achtung! contr.helmert in R is **reverse helmert** coding (== comparing each level of categorical variable to the mean of the previous level(s))



#you can change the reference level

e.g. in dummy coding:



#some stuff from slides:

#criteria contrasts need to fulfill:

* Independent (orthogonal): Contrasts must not interfere with each other (they must test unique hypotheses).
* Dichotomous: Each contrast should compare only two chunks of variation.
* K - 1: You should always end up with one less contrast than the number of groups.

#mentioned in the lecture:

* Dummy (default):
  + contr.treatment(#levels, base = referencelevel)
  + contr.SAS(#levels)
* Deviation: contr.sum(#levels)
* Helmert: contr.helmert(#levels)
* Polynomial: contr.poly(#levels)

**#model diagnostics (assumptions of the model)**

#create a model

model <- aov(DV~IV, data = dataframe) #package ‘base’

#Independent errors: For any two observations the residual terms should be uncorrelated (or independent).

#Durbin– Watson test, which tests for serial correlations between errors. Specifically, it tests whether adjacent residuals are correlated. The test statistic can vary between 0 and 4, with a value of 2 meaning that the residuals are uncorrelated. A value greater than 2 indicates a negative correlation between adjacent residuals, whereas a value less than 2 indicates a positive correlation. The size of the Durbin–Watson statistic depends upon the number of predictors in the model and the number of observations. As a very conservative rule of thumb, values less than 1 or greater than 3 are definitely cause for concern; however, values closer to 2 may still be problematic depending on your sample and model. R also provides a p-value of the autocorrelation. Be very careful with the Durbin–Watson test, though, as it depends on the order of the data: if you reorder your data, you’ll get a different value.

dwt(model) #package ‘car’

#check residuals (outliers)

dataframe$residuals<-rstandard(model)

dataframe$leverage<-hatvalues(model)

#check influential cases

by(dataframe$residuals, dataframe$bsf, densityplot)

by(dataframe$leverage, dataframe$bsf, densityplot)

**#main analyses**

#you can use lm() or aov() objects, the output will be the same.

to see differences between groups you specified with your contrast:

summary.lm()

→ summary() works the same way if you have a lm() object

#for the overall model F and beta-value of the factor use

summary.aov()

→ summary() works the same way if you have an aov() object

#if you have to use different sum-of-squares, depending on whether or not you included interactions use Anova()

#default SS is 1 in R instead of 3 like in SPSS.

#type = “1” (sequential)

This type of SS depends on the order in which even main effects are listed in the MODEL statement.

Pros:

(1) SS for all the effects add up to the total SS, which is not generally true for any other types.

(2) Preferable when effects of some factors (e.g., control measures) should be considered before others.

Cons:

(1) Hypotheses depend on the order in which effects are specified. Lack of invariance is a problem for certain designs.

(2) Not appropriate for factorial designs

#type = “2” (partially sequential)

A and B main effects will both be adjusted for each other (since neither contains the other), but will NOT be adjusted for A\*B (since it contains both A and B). A\*B will be adjusted for both main effects.

Pros:

(1) appropriate for model building.

(2) most powerful type when there is no interaction

(3) invariant to the order in which main effects are entered into the model.

Cons:

(1) Not appropriate for models with multiple interactions

(2) Not appropriate for factorial designs

#type = “3” (weighted)

SS for a given effect is adjusted for all other effects listed in the MODEL statement, regardless of whether they contain the given effect or not. In particular A and B main effects WILL be adjusted for the A\*B interaction.

Pros:

(1) Not sample size dependent, so better for unbalanced data,

(2) Interactions have equal priority for explaining variance

Cons:

(1) testing main effects in the presence of interactions

(2) Not appropriate for designs with missing cells

#type = “4” (balanced)

•Goodnight’s method for missing cell layouts

–similar to Type III in spirit, with a different strategy for compensating for missing cells when estimating the model parameters.

Anova(Model, type="II") or Anova(Model, type="2") #package ‘car’

Anova(Model, type="III") or Anova(Model, type="3") #package ‘car’

drop1(Model, ~., test = “F”)

**#post-hoc tests**

#you are testing the differences between adjusted group means with the glht() function. You can choose for dose=”Tukey” or dose=”Dunnett”. Normally, we use Tukey.

postHocs<-glht(Model,linfct=mcp(dose="Tukey“)) #package ‘multcomp’

summary(postHocs)

#to check the adjusted cell means:

adjustedMeans <- effect(“variable”, model, se=TRUE)

summary(adjustedMeans)

#print SEs

adjustedMeans$se

**RM ANOVA**

#For RM-Anovas you will have to create a dataframe in long-format to have variable for the combination of all factors (see the note on reshape()). A way to create that variable would be:

groups<-gl(9, 1, 180, labels = c("beerpos", "beerneg", "beerneut", "winepos", "wineneg", "wineneut", "waterpos", "waterneg", "waterneut"))

→ This only works for equal group sizes. Otherwise you will have to do it in a different way, for example:

dataframe$groups <- factor(paste(data$factor1, data$factor2, sep = "\_"))

**#assumptions**

#checking normality for every group seperately

by(dataframe$DV, list(dataframe$bsf1, dataframe$bsf2), stat.desc, norm = TRUE)

#homogeneity of variance

#use Levene’s test for every factor (including the interactions)

leveneTest(longdataframe$DV, longdataframe$IV1, center = median)

leveneTest(longdataframe$DV, interaction(dataframe$IV1, dataframe$IV2), center = median) #package ‘car’

#Homogeneity of Covariance, aka Sphericity

#Mauchly’s test for sphericity is included in ezANOVA from package ‘ez’

model <- ezANOVA(data = dataframe, dv = .(outcome), wid = .(id), within = .(repeated measures), between = .(group measures), type = 3, detailed = FALSE) #package ‘ez’

#example

attitudeModel<-ezANOVA(data = longAttitude, dv = .(attitude), wid = .(participant), within = .(imagery, drink), type = 3, detailed = TRUE)

#post hocs for RM Anova

pairwise.t.test(longdataframe$DV, longdatadrame$IV, paired = TRUE, p.adjust.method = "bonferroni")

**Factorial Designs overview (from slides)**

* Independent
  + 2+ between-participant factors
  + 2+ IVs both measured with different participants
* Repeated-measures
  + 1+ between- and 1+ within-participant factors
  + 1+ IVs both measured with same participants
* Mixed
  + 1+ between- and 1+ within-participant factors
  + 2+ IVs with 1+ measured with same participants and 1+ measured with different participants

**#\*\*\*OverviewPackages\*\*\***

install.packages(‘whatever’) # for reasons

install.packages('psych') # for describe() and describeBy()

install.packages('lattice') # for densityplot()

install.packages('foreign') # for importing SPSS-data with read.spss

install.packages('pastecs') # for stat.desc()

install.packages('ltm') # for rcor.test()

install.packages('reshape') # for melt/cast

install.packages('ggplot2') # for plotting shit

install.packages('Hmisc') # loading SPSS files

install.packages('ggthemes') # cause it’s pretty

install.packages('car') # for Levene's test

install.packages('SuppDists') # for Fmax

install.packages('ltm') # for linear regression

install.packages('anova') # for F-change in hierarchical regression

install.packages('multcomp') # for Tukey posthoc

install.packages('compute.es') # for effect sizes

install.packages("effects") # for effects

install.packages(“ez”) # for repeated measures ANOVA

#assumptions

#Durbin-Watson test for independence of errors.

#Values should be between 1 and 3.

dwt(modelname) #package ‘car’

#Levene’s Test for Homogeneity of Variance

#If the result is significant, then variances are not homogeneous

leveneTest(dataframe$variable ~ datafame$groupingvariable, center = median) #package ‘car’

#VIF check for multicollinearity

#VIF values should NOT be larger than 10

vif(modelname) #package ‘car’

Good luck everyone!!

Lots of love,

Julian, Kris & ilse