One Way ANOVA with R

Completely Randomized Design - Between Groups

Bruce Dudek

2020-11-07

Contents

Preface

This document can be a standalone "how-to" document for R users. However, it is primarily intended for students in the APSY510/511 statistics sequence at the University at Albany. It is a fairly thorough treatment of graphical and inferential evaluation of one-factor designs. It presumes prior background coverage of the ANOVA logic from standard textbooks such as Howell or Maxwell, Delaney and Kelley (2017). The analyses are intended to parallel and exhaust the methods already covered with SPSS, and to extend them to additional topics.

This book/monograph uses the **bookdown** package (Xie, 2018a) for R (R Core Team, 2018), which was built on top of **rmarkdown** (Allaire et al., 2018) and **knitr** (Xie, 2015). RStudio (RStudio Team, 2015) was used for all writing and R programming.

Chapter 1

Background and R Setup

The goal of this document is provision of a template for using R to evaluate data from a 1-factor design that is typically called a 1-way ANOVA problem. The completely randomized design used for the initial illustration here is a 3-group design. The data come from an exercise in the classic Hays textbook. Later chapters utilize other data sets that have more treatment conditions.

The standard R axiom that there are always multiple ways of performing any task is never more accurate than with the ANOVA models. Beginning with graphical depiction and extending to standard NHST inferences, contrast analysis and post hoc tests, and evaluation of assumptions, the document also includes some rudimentary Bayesian approaches to inference.

This document

- Is intended for use by APSY511 course at UAlbany, but can be more broadly used by data analysts
- Is a fairly full one-way anova exposition for a 3-group design and a second illustration with a five group design
- Implements graphical summaries, numerical descriptions
- Approaches ANOVA as linear modeling and is supplemented with analytical contrasts, and multiple comparison tests
- Implements trend analysis for quantitative IV's
- Includes graphical and inferential evaluation of assumptions.
- Includes sections on Bayesian Inference, Robust methods, and Resampling Methods
- It includes a section on sample size planning with power analysis.

The document is constantly under development:

- Additional work on effect size computations,
- implementation of some newer multiple comparison methods
- additional work on robust and resampling methods

One of the primary goals is to reproduce all the work we have accomplished with the SPSS REGRESSION, GLM, MANOVA and ONEWAY procedures (and then some).

Several R packages are required:

```
#if (!requireNamespace("BiocManager", quietly = TRUE))
#    install.packages("BiocManager")
#BiocManager::install("Biobase", version = "3.8")
```

```
# load packages and import data
library(afex)
library(asbio)
library(BayesFactor)
library(beeswarm)
library(car)
library(coin)
library(dunn.test)
library(emmeans)
library(ez)
library(DTK)
library(ggplot2)
library(ggthemes)
library(granova)
library(gt)
library(gridExtra)
library(KScorrect)
library(knitr)
library(lattice)
library(lawstat)
library(lmboot)
library(lmPerm)
library(lsr)
library(multcomp)
library(multtest)
library(mutoss)
library(nortest)
library(outliers)
library(pgirmess)
library(plotrix)
library(plyr)
library(psych)
library(pwr)
library(rcompanion)
library(sciplot)
library(sjstats)
library(userfriendlyscience)
```

```
library(WRS2)
library(dplyr)
```

Package citations for packages loaded here (in the above order): **afex** (Singmann et al., 2018), **asbio** (Aho, 2019), **BayesFactor** (Morey and Rouder, 2018), **beeswarm** (Eklund, 2016), **car** (Fox et al., 2018), **coin** (Hothorn et al., 2017b), **emmeans** (Lenth, 2019), **ez** (Lawrence, 2016), **DTK** (Lau, 2013), **dunn.test** (Dinno, 2017), **ggplot2** (Wickham et al., 2018), **ggthemes** (Pruzek and Helmreich, 2014), **gridExtra (Auguie, 2017),** gt** (Iannone et al., 2019), **KScorrect** (Novack-Gottshall and Wang, 2018), **knitr** (Xie, 2018b), **lattice** (Sarkar, 2018) **lawstat** (Gastwirth et al., 2017), **lmPerm** (Wheeler and Torchiano, 2016), **lsr** (Navarro, 2015) **multcomp** (Hothorn et al., 2017a), **multtest** (Pollard et al., 2018), **mutoss** (Team et al., 2017), **nortest** (Gross and Ligges, 2015), **outliers** (Komsta, 2011), **pgirmess** (Giraudoux, 2018), **plotrix** (Lemon et al., 2018), **plyr** (Wickham, 2016), **psych** (Revelle, 2019), **pwr** (Champely, 2018), **rcompanion** (Mangiafico, 2019), **sciplot** (Morales et al., 2017), **sjstats** (Lüdecke, 2019), **userfriendlyscience** (Peters, 2017), **WRS2** (Mair and Wilcox, 2018), **dplyr** (Wickham et al., 2019)

1.1 A note on R version and package installations.

At the point in time that this document was created, the transition to R version 4.0 is ongoing and some packages have not been revised to work with R4.0. Installing source files rather than binaries can be a work around for six packages. The general process is to download the appropriate source files from the repository (ending in "tar.gz"). Then use this function to install the package:

```
#install.packages(file.choose(), repos=NULL, type="source")
```

Note that Windows users will need to install the Rtools suite of tools before source package installation is attempted.

https://cran.r-project.org/bin/windows/Rtools/

Two packages that required for permutation tests and bootstrapping, **lmPerm** and **lmboot**, can be obtained from CRAN (search the package name).

Three packages come frome the BioConductor suite of r packages and the core BioConductor installer should also be installed.
https://www.bioconductor.org/

Search for pages of each of these four to download and install the latest package source files. But by the time you read this the normal process of installing the binary files may work (see the **BiocManager** page)

BiocManager Biobase BioGenerics multtest

1.2 Resources

The following list will provide a good start for those needing a broader background in ANOVA techniques and more detailed sources for the primary packages empoyed in this document.

- Salvatore S. Mangiafico's R Companion: [https://rcompanion.org/rcompanion/d_05.html]
- Martin Schweinberger's Blog: [http://www.martinschweinberger.de/blog/one-way-anova/]
- cwoods on RPub: [https://rpubs.com/cwoods/anova]

Chapter 2

Prepare the data and do Exploratory Data Analysis

This data set comes from an experiment described in the textbook by Hays (1994), section 10.16, pg 399.
The study examined whether asynchronous presentation of video and audio recordings of speakers impaired memory of what was viewed/heard. It is a one-factor design, 3 levels. In the synchronous condition (control) audio and visual recordings were synchronoized. In the two asynchronous conditions the audio was slightly ahead of the visual image of the speaker's lips (fast) or slightly behind the visual image (slow). This independent variable is called "factora" in the data set. The dependent variable (called dv in the data set) is number of words recalled from a list of 50 heard by the participant from the presentation of the reocording.

2.1 Import and prepare the data set

The data set is found in a .csv file and has equal sample sizes of ten per group. It is thus a "balanced" design, and a "completely randomized" design.

```
hays <- read.csv("data/hays1.csv",header=TRUE, stringsAsFactors = T)
# better yet, implement the file.choose function so that
# you don't have to change the default folder
# hays <- read.csv(file.choose(),header=TRUE, stringsAsFactors=T)
# note that the initial data frame sees the dv as
# integer rather than numeric, so....
hays$dv <- as.numeric(hays$dv)
# show the structure of the data frame
str(hays)
```

```
## 'data.frame': 30 obs. of  2 variables:
##  $ factora: Factor w/ 3 levels "control","fast",..: 1 1 1 1 1 1 1 1 1 1 ...
##  $ dv     : num  27 28 33 19 25 29 36 30 26 21 ...
```

```
#hays
# I will use the attach function here to simplify our code even though
# current best practices in R recommend against using it.
# The downside of using attach is not encountered in this illustration
attach(hays)
```

2.2 Numerical Summaries

Basic description of the DV as a function of factora is accomplished the the 'describeBy' function from **psych**.

```
describeBy(dv, group=factora, type=2)
```

```
##
##  Descriptive statistics by group
## group: control
##    vars  n mean  sd median trimmed  mad min max range  skew kurtosis   se
## X1    1 10 27.4 5.1   27.5   27.38 3.71  19  36    17 -0.04    -0.09 1.61
## ------------------------------------------------------------
## group: fast
##    vars  n mean  sd median trimmed  mad min max range skew kurtosis   se
## X1    1 10 21.2 5.2   20.5   20.88 4.45  15  30    15 0.66    -0.65 1.65
## ------------------------------------------------------------
## group: slow
##    vars  n mean   sd median trimmed  mad min max range skew kurtosis  se
## X1    1 10 21.8 2.53   22.5      22 2.22  17  25     8 -0.7    -0.29 0.8
```

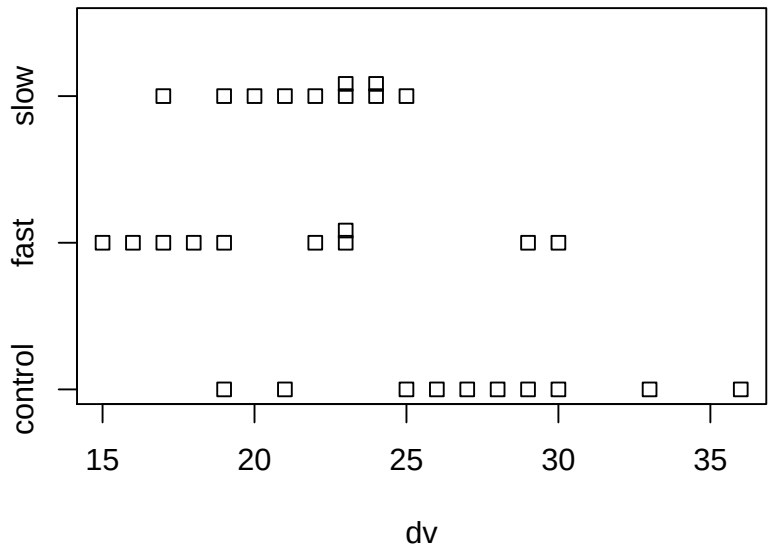
2.3 Graphs of Data by Group

There are a great many ways to draw graphs to display data of the type found even in this simple 3-group design. R can produce them with varying degrees of complexity in the code. Several will be demonstrated here. Some of them are quick tactics to "get to know" your data. Others can be seen as close to publication quality.

2.3.1 Simple EDA Plots

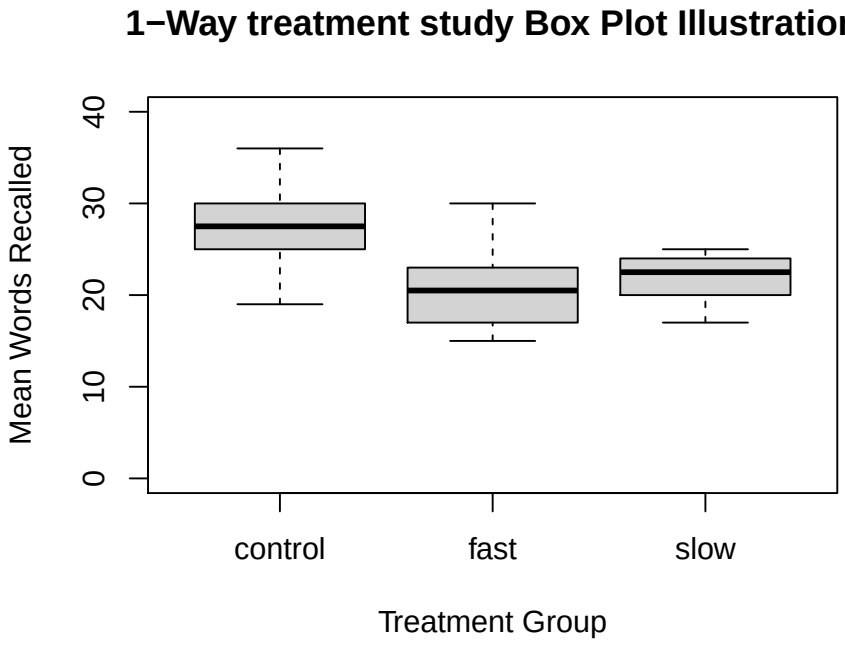
First, we can quickly obtain a simple and very basic plot called a stripchart.

```
stripchart(dv~factora, method="stack")
```



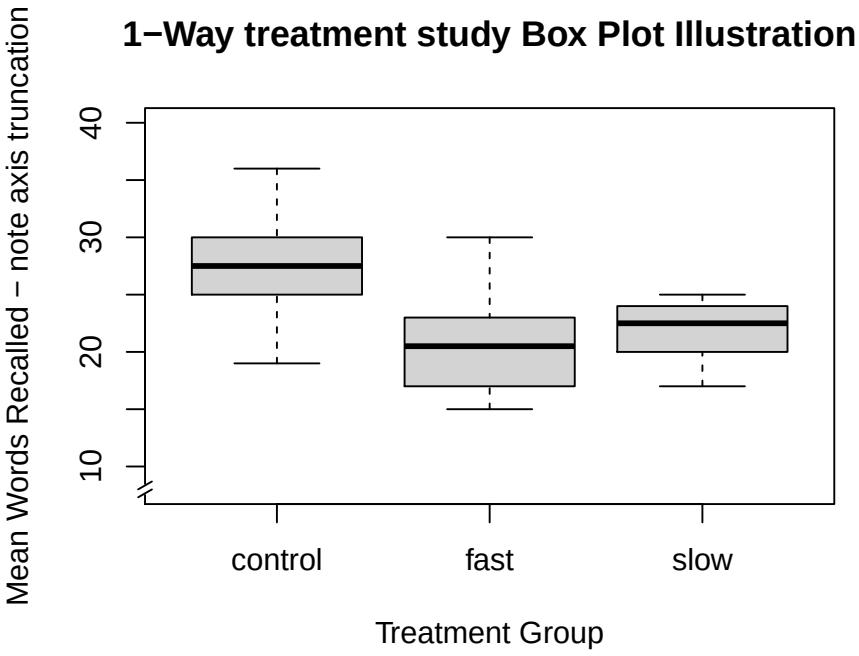
It is always useful to obtain boxplots of the DV for the all groups in a 1-way design. With base system graphics it is possible to display data from multiple conditions on the same plot.

```
boxplot(dv~factora, data=hays,col="lightgray",ylim=c(0,40),
  main="1-Way treatment study Box Plot Illustration",
  xlab="Treatment Group",
  ylab="Mean Words Recalled")
```



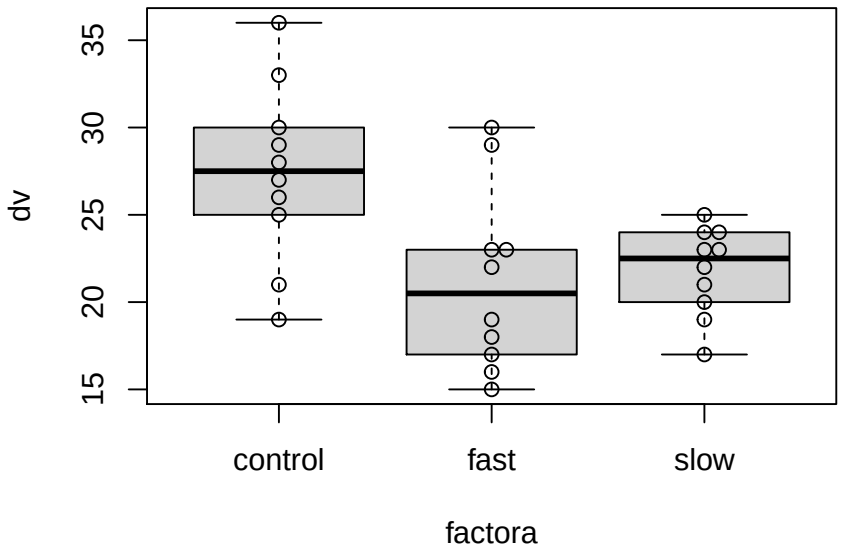
What if we want to truncate the axis and show an axis break? (can't do in 'ggplot2')? Let's do it for this boxplot. The `axis.break` function from the **plotrix** package accomplishes this.

```
#library(plotrix) # to obtain the axis.break function
boxplot(dv~factora, data=hays,col="lightgray",ylim=c(8,40),
        main="1-Way treatment study Box Plot Illustration",
        xlab="Treatment Group",
        ylab="Mean Words Recalled - note axis truncation")
axis.break(2,8,style="slash")
```



One of Tufte's axioms is "show the data." Here, using the beeswarm package, we can draw a box plot with raw data points overlaid. Note that the beeswarm function could have been executed after either of the boxplot graphs drawn above as well. Here it produces the simplest/rudimentary boxplot.

```
boxplot(dv~factora)
beeswarm(dv~factora,add=T)
```



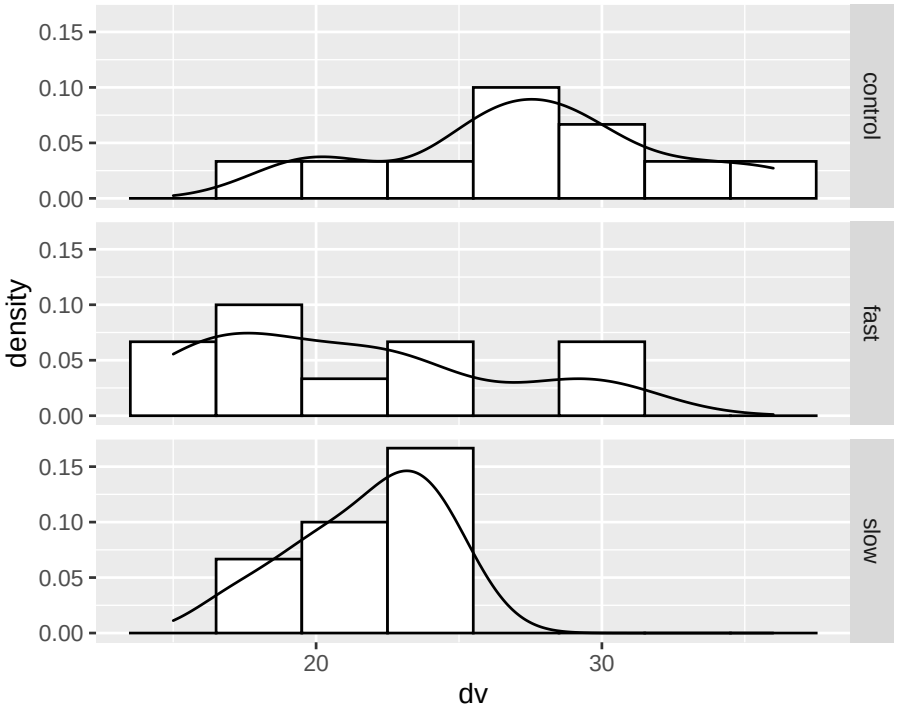
2.3.2 Frequency Histograms

In many situations it is helpful to visualize each group's DV distribution with a frequency histogram. The data set used in this document has small sample size per group, so the frequency histogram is not likely to be useful beyond what can be seen with the simpler stripchart above. Nonetheless, it is useful to put code in place as a template for use in situations where sample size is larger. The approach taken here uses 'ggplot2' to create a plot that employs "small multiples", where different panels are drawn for each group. It is also possible to create a layout of histograms with the 'layout' function for base system graphics, but that approach has already been demonstrated in prior R tutorials.

Here, I kept the ggplot fairly simple so as to be useful for quick EDA.

```
#library(ggplot2)
# first, the core xy plot specs
hbase <- ggplot(hays, aes(x = dv))
# now the base plot
hbase1 <- hbase + geom_histogram(aes(y=..density..), # change y axis to density
                                 bins=8, colour="black", fill="white") +
                  geom_density()
```

```
# now add the specification that creates separate panels for eaach group
hbase2 <- hbase1 + facet_grid(factora ~ .)
hbase2
```



2.3.3 Bar Graphs with Error Bars

The applied sciences rely heavily on a type of bar plots of means, with std errors (or CI's, or SD's) displayed as "whiskers". There is a bit of cottage industry on the web that is heavily critical of these types of graphs, focusing on the fact that information is lost when the data are presented this way. Just do a google search on "dynamite plots" and you will find many blog posts and textbook sections that argue for doing away with them.

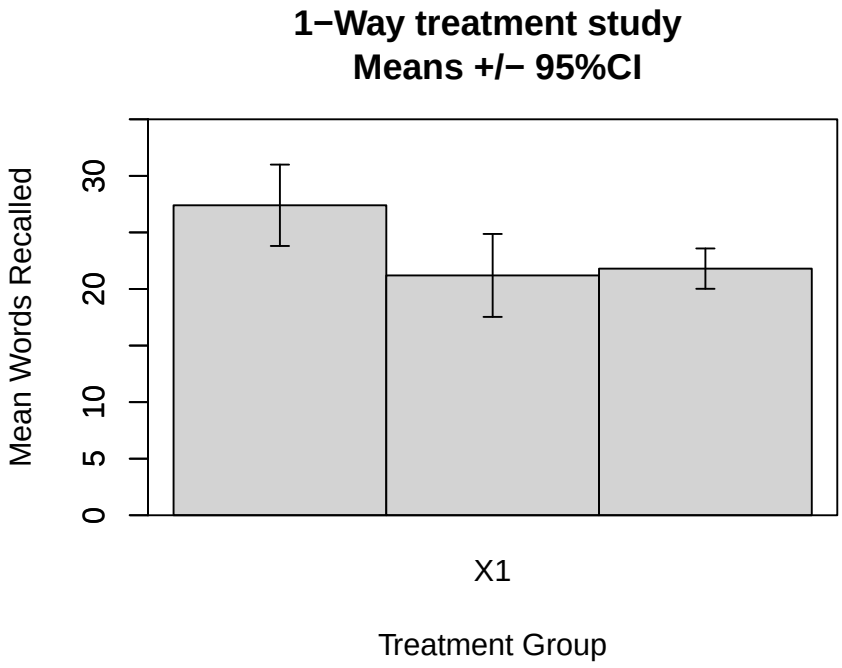
Perhaps this is why they are not always simple to obtain in R. Judicious use of them as summary graphs for "ANOVA" types of analyses strikes me as OK. But if one finds that the data have issues of skewness, outliers, or heteroscedasticity, then "showing the raw data points" might be a better approach (c.f. Tufte) - adding raw data onto the bar graph as outlined in the "scientific graphing practices" part of the course.

Notwithstanding this criticism, I have generated some examples of how to draw these graphs in R. I still feel that commercial graphing software such

as SigmaPlot or Origins would be a better choice for publication quality graphs of these types.

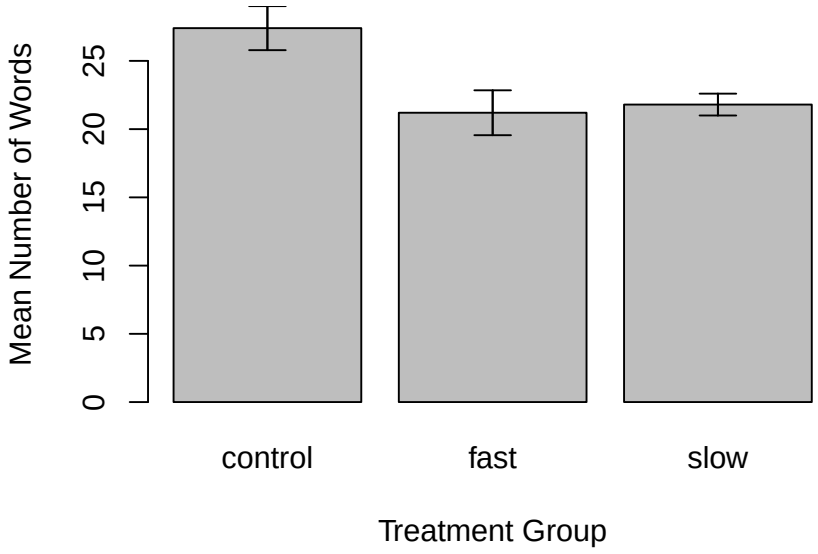
In the 'psych' package, the error.bars.by function gives * a good plot with 95% CI errors as the default. * Other % CI's can be specified - read the help file: ?error.bars.by * colors of the bars are chosen simply to illustrate the capability. These colors are probably not a good choice for either presentation or publication

```
error.bars.by(hays$dv,hays$factora,bars=TRUE, ylim=c(0,35),
 main="1-Way treatment study \n Means +/- 95%CI",
 xlab="Treatment Group",
 ylab="Mean Words Recalled",
 labels=c("control","fast","slow"),
 #colors=c("lemonchiffon","forestgreen", "springgreen4"))
 colors=c("lightgray","lightgray", "lightgray"))
```



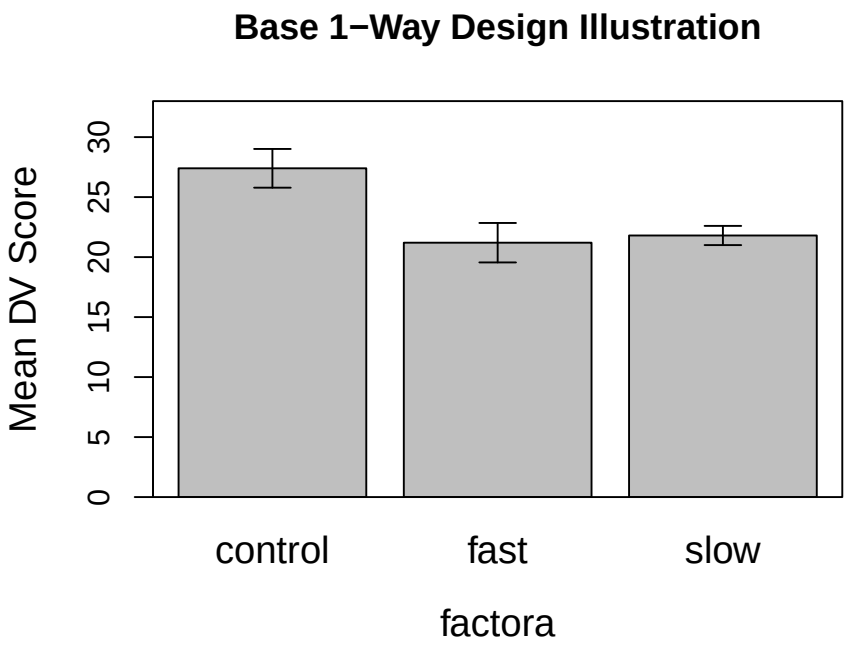
I found, in the `asbio` package, a function to generate bar graphs +/- either std errors or CI's. Here is a graph with means +/- std errors.

```
#require(asbio)
bplot(dv,factora,int="SE",
      xlab="Treatment Group", ylab="Mean Number of Words",
      print.summary=F)
```



Perhaps a nicer way is available from the from 'sciplot' package. default gives +/- 1 std error:

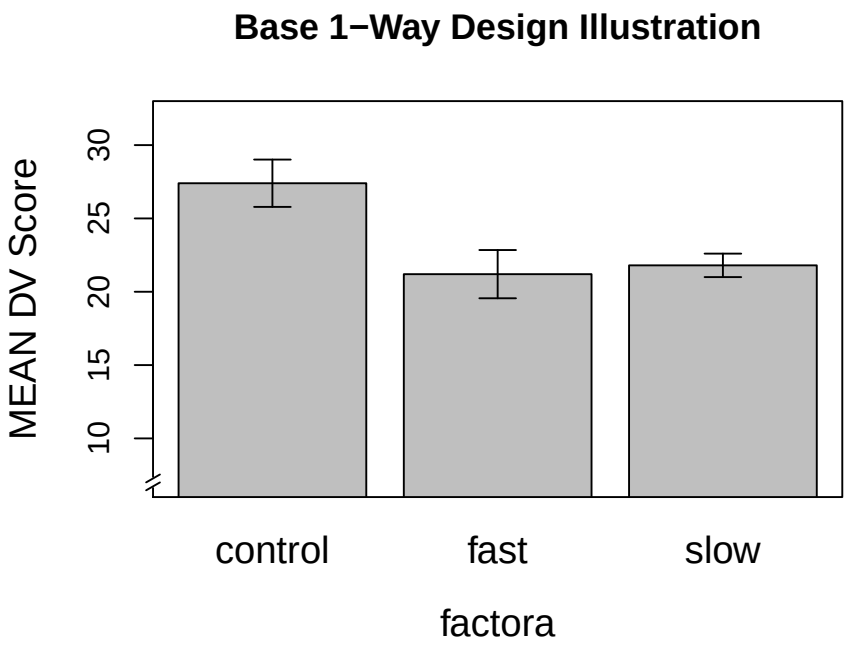
```
#require(sciplot)
#win.graph() # or quartz() or x11()
bargraph.CI(factora,dv,lc=TRUE, uc=TRUE,legend=T,
            cex.leg=1,bty="n",col="gray75",
            ylim=c(0,33),
            ylab="Mean DV Score",main="Base 1-Way Design Illustration",
            cex.names=1.25,cex.lab=1.25)
box()
```




```
#axis(4,labels=F)
```

Next, we can put an axis break on the Y axis to give a visual indicator of the scale break.

```
# use the axis.break function from plotrix to give the slashes
library(plotrix)
#win.graph() # or quartz() or x11()
bargraph.CI(factora,dv,lc=TRUE, uc=TRUE,legend=T,
            cex.leg=1,bty="n",col="gray75",
            ylim=c(6,33),
            ylab="MEAN DV Score",main="Base 1-Way Design Illustration",
            cex.names=1.25,cex.lab=1.25)
box()
#axis.break(4,7,style="slash")
axis.break(2,7,style="slash")
```



2.3.4 GGPLOT2 can draw bar graphs with error bars

This approach requires some preliminary work to establish the means and std errors to be used. * The summarySE function produces a data frame that has the summary stats by group. * It comes from http://www.cookbook-r.com/Manipulating_data/Summarizing_data/ * ggplot cannot work directly on the data frame to produce this type of plot. We need the summary stats first.

```
#read the R script file containing code for the summarySE function
source("summarySE.R")
# now use summarySE on our data
hays_summ <- summarySE(hays, measurevar="dv", groupvars="factora")
str(hays_summ)
```

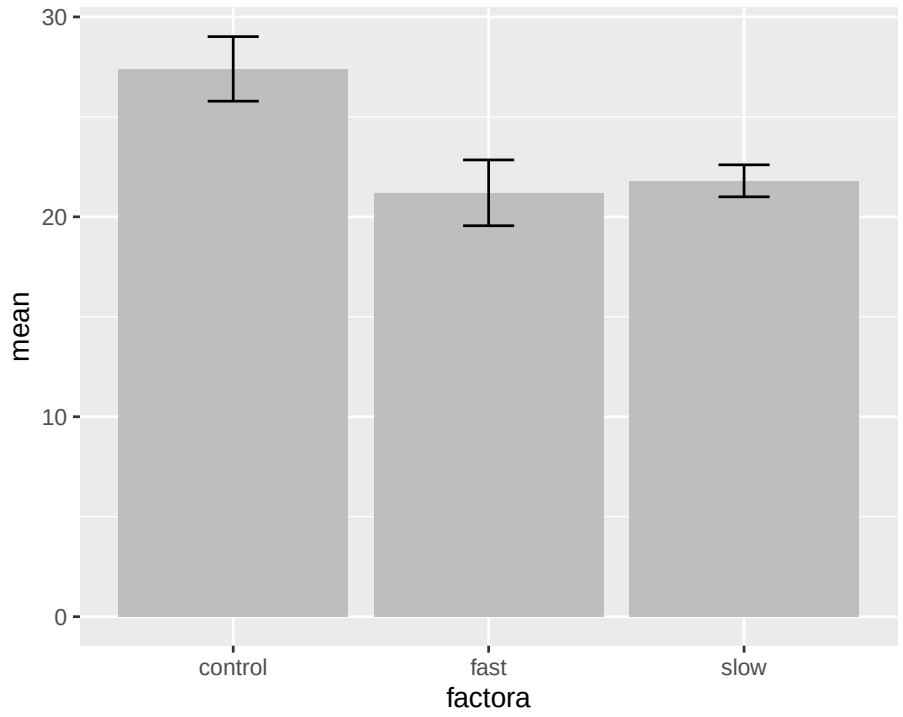
```
## 'data.frame': 3 obs. of  6 variables:
##  $ factora: Factor w/ 3 levels "control","fast",..: 1 2 3
##  $ N      : num  10 10 10
##  $ dv     : num  27.4 21.2 21.8
##  $ sd     : num  5.1 5.2 2.53
##  $ se     : num  1.61 1.65 0.8
##  $ ci     : num  3.65 3.72 1.81
```

```
# rename the column that contains the mean to something more less confusing
colnames(hays_summ) <- c("factora", "N", "mean", "sd", "sem", "95%CI" )
# look at the product
kable(hays_summ)
```

factora	N	mean	sd	sem	95%CI
control	10	27.4	5.103376	1.613829	3.650735
fast	10	21.2	5.202564	1.645195	3.721690
slow	10	21.8	2.529822	0.800000	1.809726

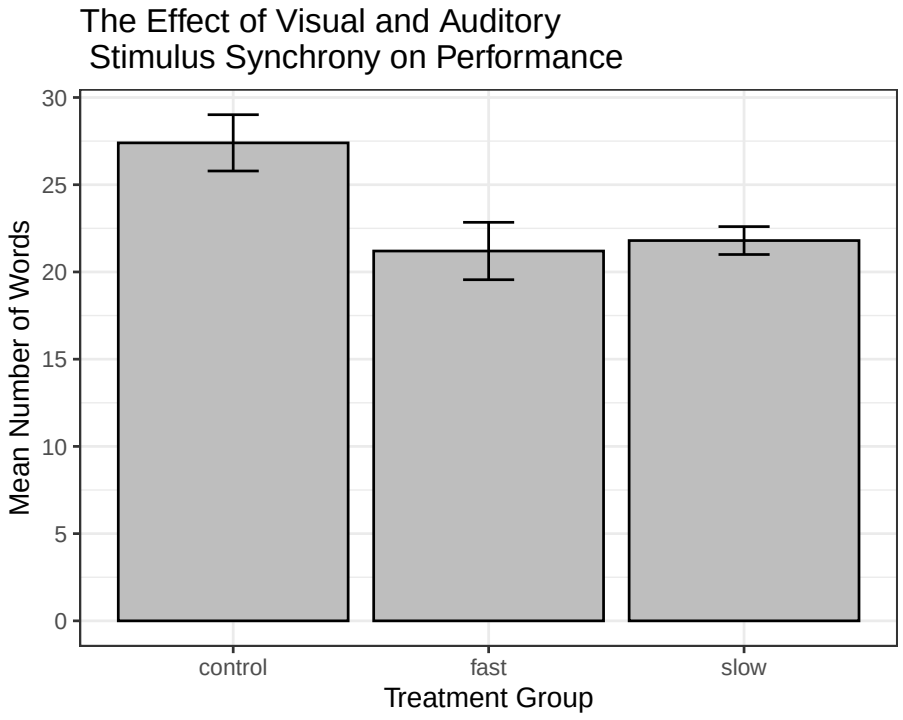
Next, we draw the base plot and add error bars:

```
#win.graph() # or quartz() or x11()
p1 <- ggplot(hays_summ, aes(x=factora, y=mean)) +
             geom_bar(position=position_dodge(), stat="identity", fill="gray")
#p1
# add on std error bars
#win.graph() # or quartz() or x11()
p2 <- p1 +      geom_errorbar(aes(ymin=mean-sem, ymax=mean+sem),
                width=.2,                    # Width of the error bars
                position=position_dodge(.9))
p2
```



A more finished ggplot2 graph might look like this:

```
#win.graph() # or quartz() or x11()
p3 <- ggplot(hays_summ, aes(x = factora, y = mean)) +
  theme_bw() +
  geom_bar(
    position = position_dodge(),
    stat = "identity",
    fill = "gray",
    colour = "black"
  ) +
  geom_errorbar(aes(ymin = mean - sem, ymax = mean + sem),
                width = .2,
                # Width of the error bars
                position = position_dodge(.9)) +
  scale_y_continuous(breaks = 0:30 * 5) +
  xlab("Treatment Group") +
  ylab("Mean Number of Words") +
  ggtitle("The Effect of Visual and Auditory\n Stimulus Synchrony on Performance")
p3
```

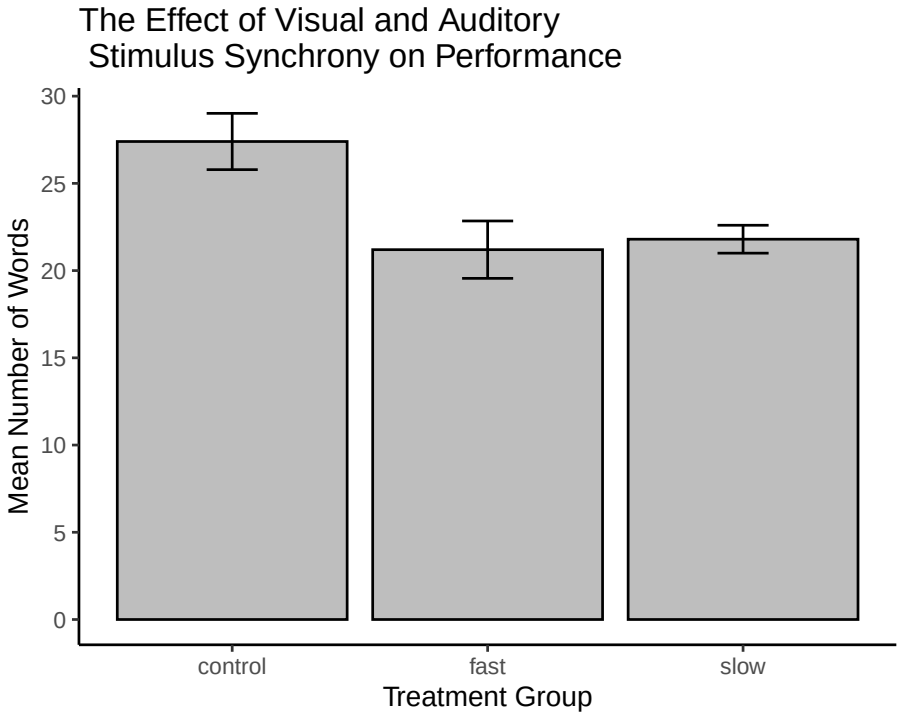


The reader might have noticed a call in this last 'ggplot2' graph called theme_bw(). 'Ggplot2' can handle many different themes available in add-on

packages. I use one called 'ggthemes'.

One more example is shown here with a slightly different way of constructing the graph. Notice the 'ggplot2' "way" of graph construction. First define the basic plot and then add elements (such as "geoms") with the plus sign. The themes are added similarly. Here I use one called "classic".

```
plotbase <- ggplot(hays_summ, aes(x = factora, y = mean)) +
  geom_bar(
    position = position_dodge(),
    stat = "identity",
    fill = "gray",
    colour = "black"
  ) +
  geom_errorbar(aes(ymin = mean - sem, ymax = mean + sem),
                width = .2,
                # Width of the error bars
                position = position_dodge(.9)) +
  scale_y_continuous(breaks = 0:30 * 5) +
  xlab("Treatment Group") +
  ylab("Mean Number of Words") +
  ggtitle("The Effect of Visual and Auditory\n Stimulus Synchrony on Performance")
#win.graph() # or quartz() or x11()
plot1 <- plotbase + theme_classic()
plot1
```



Many other themes are available from ggthemes package. see [https://cran.r-project.org/web/packages/ggthemes/vignettes/ggthemes.html]

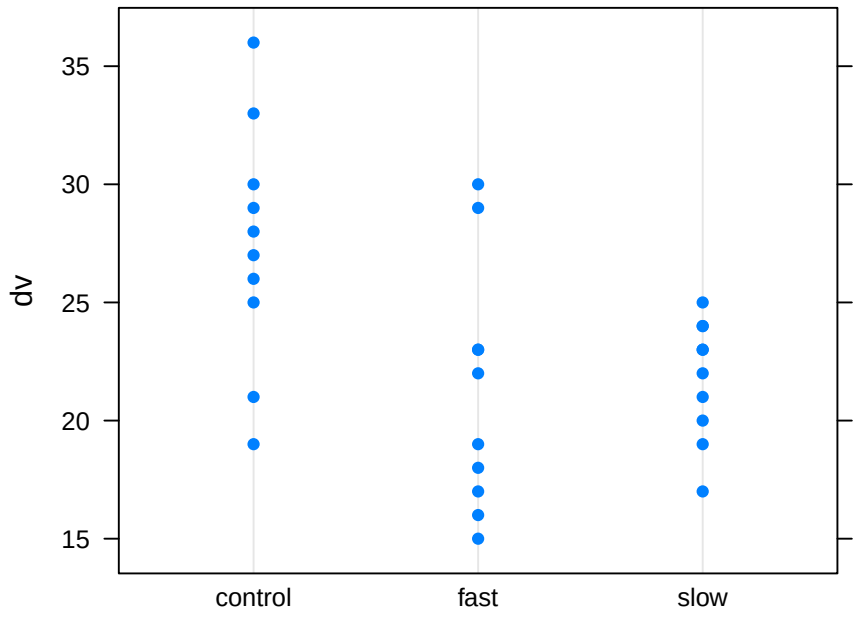
The reader might try any one of the following:

```
plot1 <- plotbase + theme_tufte()
plot1 <- plotbase + theme_grey()
plot1 <- plotbase + theme_dark()
plot1 <- plotbase + theme_economist()
plot1 <- plotbase + theme_excel() # very ugly
plot1 <- plotbase + theme_few() # from Stephen Few
plot1 <- plotbase + theme_fivethirtyeight()
plot1 <- plotbase + theme_gdocs()
plot1 <- plotbase + theme_hc()
plot1 <- plotbase + theme_solarized()
plot1 <- plotbase + theme_stata()
plot1 <- plotbase + theme_wsj()
plot1 <- plotbase + theme_pander()
#plot1
```

2.4 Miscl: Dotplots (lattice), Violin plots and advanced layout with ggplot2

An alternative to the base system stripchart shown above is a simple dot plot. Although there are many ways to accomplish this in R, the `dotplot` function from the **lattice** package is simple to use.

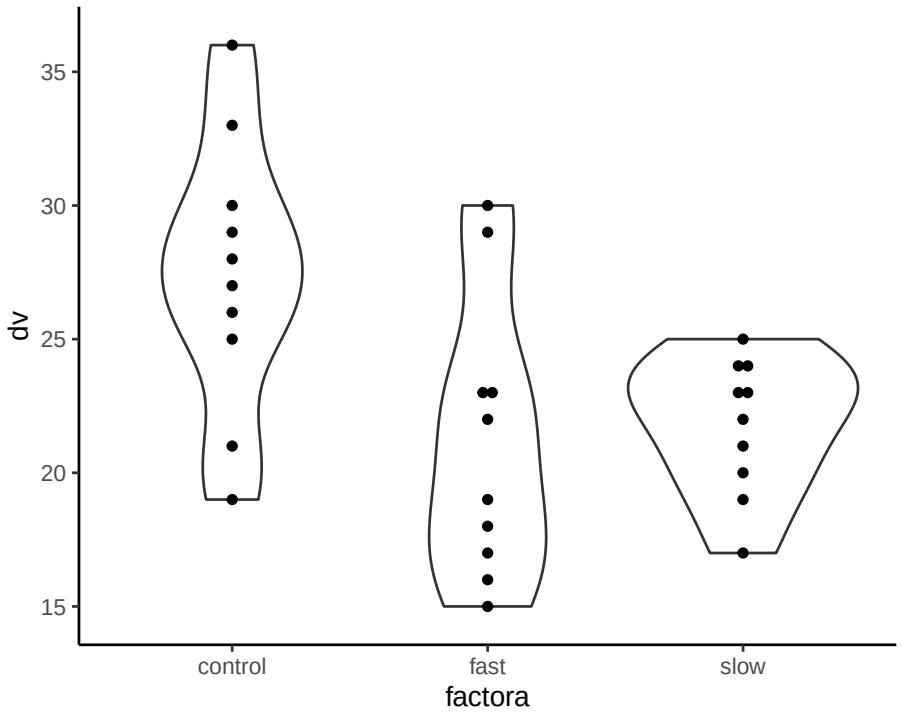
```
# require(lattice)
with(hays, dotplot(dv ~ factora))
```



One useful plot that we have seen before combines the violinplot (showing kernel density curves) with either a boxplot or, in this case, a dotplot. The violinplot is best used when sample sizes are a bit larger than the n=10 in this hays data set. Nonetheless, the capability is illustrated. This figure is generated using **ggplot2** techniques.

```
ggplot(hays, aes(factora, dv)) +
  geom_violin() + geom_dotplot(binaxis='y', stackdir='center', dotsize=.5) +
  theme_classic()
```

```
## `stat_bindot()` using `bins = 30`. Pick better value with `binwidth`.
```



2.4.1 Combining multiple ggplots into one layout

The next plot is an illustration of combining multiple graphs into one layout. First, we will use the hays data set, for continuity. But it is not very useful to draw histograms or kernel density functions with such a low sample size, so a second illustration uses the "cereals" data set.

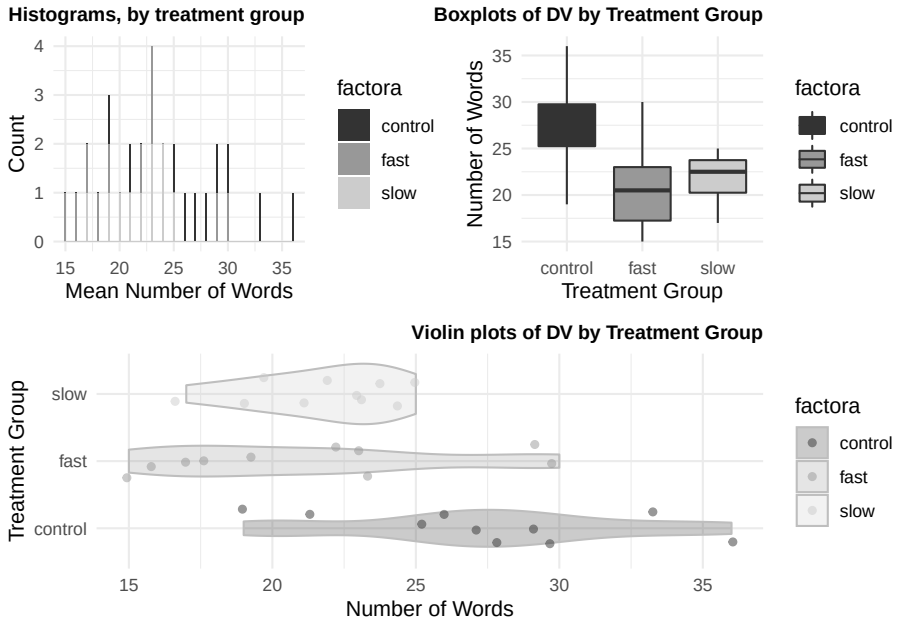
```
# Modeled after https://dmyee.files.wordpress.com/2016/03/advancedggplot.pdf
#Histogram of DV, by treatment condition
p1<-ggplot(data = hays, aes(x = dv, fill=factora)) +
  geom_histogram(binwidth = .1) +
  scale_colour_grey() + scale_fill_grey() +
  xlab("Mean Number of Words") + ylab("Count") +
  ggtitle("Histograms, by treatment group") +
  theme_minimal() +
  theme(plot.title = element_text(size=10, face = "bold", hjust = 1))

# Boxplots of DV, by treatment condition
p2<-ggplot(data = hays, aes(x = factora, y = dv, fill=factora)) +
  geom_boxplot() +xlab("Treatment Group") + ylab("Number of Words") +
  scale_colour_grey() + scale_fill_grey() +
  ggtitle("Boxplots of DV by Treatment Group") +
```

```
  theme_minimal() +
  theme(plot.title = element_text(size=10, face = "bold", hjust = 1))

# Violin plots of DV, by treatment condition
p3<-ggplot(data = hays, aes(x = factora, y = dv, fill=factora)) +
  geom_violin(alpha=.25, color="gray") +
  geom_jitter(alpha=.5, aes(color=factora), position=position_jitter(width=0.3)) +
  scale_colour_grey() + scale_fill_grey() +
  coord_flip() +
  xlab("Treatment Group") + ylab("Number of Words") +
  ggtitle("Violin plots of DV by Treatment Group") +
  theme_minimal() +theme(plot.title = element_text(size=10, face = "bold", hjust = 1))

# Creating a matrix that defines the layout
# (not all graphs need to take up the same space)
lay <- rbind(c(1,2),c(3,3))# Plotting the plots on a grid
grid.arrange(p1, p2, p3, ncol=2, layout_matrix=lay)
```



The second illustration uses the "cereals" data set that has been explored previoulsy. Here, I chose the dependent variable to be the "Healthiness" rating of each cereal type and the categorical/grouping factor is shelf that the cereal was found on in the supermarket. This second illustration also introduces a way to control color for "fills", by group, and uses a colorblind friendly palette of colors.

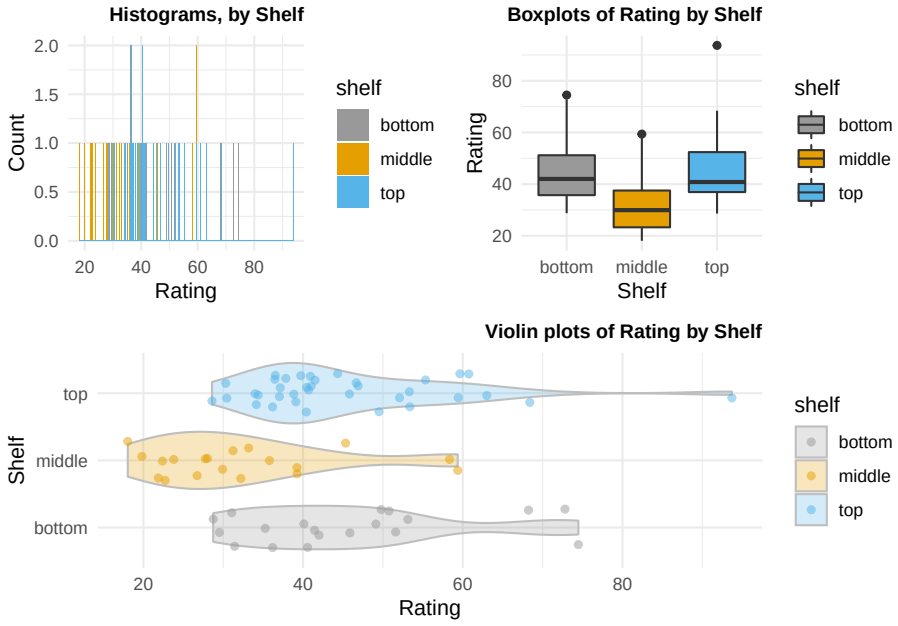
```r
cereals <- read.csv("data/cereal_cold_sugar_shelf.csv", stringsAsFactors=T)
```

```r
# Modeled after https://dmyee.files.wordpress.com/2016/03/advancedggplot.pdf
#Histograms of "Healthiness" Rating" by Shelf
#library("RColorBrewer")
cbPalette <- c("#999999", "#E69F00", "#56B4E9", "#009E73",
               "#F0E442", "#0072B2", "#D55E00", "#CC79A7")
p1<-ggplot(data = cereals, aes(x = rating, fill=shelf)) +
  geom_histogram(binwidth = .1) +
  scale_fill_manual(values=cbPalette) +
  #scale_fill_brewer(palette="Paired") +
  #scale_colour_grey() + scale_fill_grey() +
  xlab("Rating") + ylab("Count") +
  ggtitle("Histograms, by Shelf") +
  theme_minimal() +
  theme(plot.title = element_text(size=10, face = "bold", hjust = 1))

# Boxplots of "Healthiness" Rating" by Shelf
p2<-ggplot(data = cereals, aes(x = shelf, y = rating, fill=shelf)) +
  geom_boxplot() +xlab("Shelf") + ylab("Rating") +
  scale_fill_manual(values=cbPalette) +
  #scale_fill_brewer(palette="Paired") +
  #scale_colour_grey() + scale_fill_grey() +
  ggtitle("Boxplots of Rating by Shelf") +
  theme_minimal() +
  theme(plot.title = element_text(size=10, face = "bold", hjust = 1))

# Violin plots of "Healthiness" Rating" by Shelf
p3<-ggplot(data = cereals, aes(x = shelf, y = rating, fill=shelf)) +
  geom_violin(alpha=.25, color="gray") +
  geom_jitter(alpha=.5, aes(color=shelf), position=position_jitter(width=0.3)) +
  scale_fill_manual(values=cbPalette) + scale_colour_manual(values=cbPalette) +
  #scale_fill_brewer(palette="Paired") +
    #scale_colour_grey() + scale_fill_grey() +
  coord_flip() +
  xlab("Shelf") + ylab("Rating") +
  ggtitle("Violin plots of Rating by Shelf") +
  theme_minimal() +theme(plot.title = element_text(size=10, face = "bold", hjust = 1))

# Creating a matrix that defines the layout
# (not all graphs need to take up the same space)
lay <- rbind(c(1,2),c(3,3))# Plotting the plots on a grid
grid.arrange(p1, p2, p3, ncol=2, layout_matrix=lay)
```


Histograms, by Shelf
Count
Rating
shelf
bottom
middle
top
Boxplots of Rating by Shelf
Rating
Shelf
Violin plots of Rating by Shelf
Shelf
Rating

Chapter 3

Perform the standard parametric 1-way ANOVA

Note: In this chapter we can specify the data frame to be used inside the `aov` and `lm` functions, so we will detach the hays data frame to avoid confusion when contrasts for factors are re-specified. We had attached in in the earlier chapter to make writing code for graphing functions more efficient.

```
detach(hays)
```

The R user is faced with many choices of packages and functions to implement 1-way ANOVAs. We will begin with a very simple and limited function `oneway.test`. The standard method for doing ANOVA is to use the `aov` function from the base package, although `lm` can also be used. They are the most common approaches for obtaining the traditional SS partitioning and F test. We also do a bit more work with them to obtain analytical/orthogonal contrasts here, as well Three other approaches are also outlined. **ezanova**, **granova**, and **afex** are all designed to provide easier and/or additional strategies.

3.1 The `oneway.test` function

A very basic way to do a one factor ANOVA is with the `oneway.test()` function. The "var.equal" argument, when set to TRUE yeilds the standard textbook analysis where the error term (MSerror) is the pooled within-group variances and homogeneity of population variances is assumed.

```
# the var.equal argument permits using the pooled within-group error
# as the F test error term.
oneway.test(dv~factora,var.equal=TRUE, data=hays)
```

```
##
```

```
##  One-way analysis of means
##
## data:  dv and factora
## F = 5.8947, num df = 2, denom df = 27, p-value = 0.007513
```

Changing the var.equal argument is possible. This is desireable when the homogeneity of variance assumption is violated. The logic and desireability of this form of the 1-way ANOVA F test is parallel to the Welch test in the independent samples "t-test". More on alternative approaches when assumptions are violated is provided later in this document.

This Welch test is strongly recommended.

```
# setting var.equal=F results in a generalization of the
# Welch (Fisher-Behrens) form of the test.
# see your textbook or see the help pages for oneway.test() for more information
# notice the fractional df for the denominator term
oneway.test(dv~factora,var.equal=FALSE, data=hays)
```

```
##
##  One-way analysis of means (not assuming equal variances)
##
## data:  dv and factora
## F = 5.0716, num df = 2.000, denom df = 15.914, p-value = 0.01977
```

3.2 The aov and lm functions

The aov function in the base package is a "wrapper" for lm. It uses the 'lm' linear modeling engine, but permits model specification in a similar way to how we have seen lm used for regression. It expects IVs to be factors. The output with use of the summary function is the more traditional ANOVA summary table with SS and df partitioning. The simplicity of its usage is its strength. It's output is, however, limited. We need to use other functions for analytical/orthogonal/singleDF contrasts, multiple comparisons, etc. That follows below.

An extended CAVEAT BEFORE USING AOV:

Even though this script addresses only 1-way designs, there are some issues that are bigger, once we get to factorial designs that require care NOT to simply apply a generalization of these 1-way approaches to factorials with unequal sample sizes per group. I'll elaborate on this a bit here, but in the 1-way situation this discussion is largely irrelevant except for evaluation of individual contrasts and there is a later chapter in this document that addresses the questions associated with them. The anova and summary functions applied to aov objects produce Type I SS. When used in factorial designs, the aov and anova functions will not test hypotheses about unweighted marginal means. Loading the car package provides access to an Anova function (upper case first letter A) that will produce

Type III SS. Obtaining Type III SS for contrasts is not completely direct. You will see that use of the `summary` function on the `aov` model does produce Type I SS for the contrasts. I obtain tests of the contrasts with the `lm` function by applying the summary function to the `lm` model or by use of the `summary.lm` function on an `aov` object. This approach does provide a test of type III SS, but the SS are not listed. See the later chapter on Unequal Sample Sizes for details on this topic.

There is a very large debate on the value of Type I, II vs III SS. In our class, we will address that at a later point in time.

Conclusion: The direct extension, to a factorial design, of the methods outlined here will not always produce exact duplication of the SPSS/SAS methods we cover using TYPE III or UNIQUE SS methods, in factorial designs with unequal N. Since the initial/primary example in this document is balanced (equal N) and not a factorial, these issues are not relevant for this initial Hays data set, except for evaluation of contrasts. The treatment of the Type I and III SS issues here set the stage for the importance of that distinction in factorial designs where marginal means such as main effects might be either weighted or unweighted. We will better be able to grasp the issues there with the foundation put in place here and in the preceding tutorial on linear modeling in R

The "R world" is not terribly well oriented to the experimental design perspective, as it has emerged in the Psychological Sciences. Duplication of the exact approaches we have learned with SPSS/SAS can be done, but more complex designs such as factorial between-groups designs and repeated measures present unique challenges for some of the straightforward methods in SPSS/SAS. This further work is enabled by a solid understanding for the applications of `aov` and `lm` in one factor designs.

3.2.1 Using the aov function

Fortunately, obtaining the results from 'aov' is simpler than the preceding four paragraphs. `aov` utilizes the `lm` functionality to perform its work. Notice that, like `lm`, the data frame can be named as an argument precluding the need for attachment of the dataframe as we did earlier in this document.

```
fit.1 <- aov(dv~factora,data=hays)
summary(fit.1)
```

```
##             Df Sum Sq Mean Sq F value  Pr(>F)
## factora      2  233.9  116.93   5.895 0.00751 **
## Residuals   27  535.6   19.84
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Notice that this result from `aov` was produced by the `summary` function. We have previously seen that applying `summary` on a `lm` regression object produces a

coefficients table rather than the SS partitioning and F test. This is because the ANOVA summary table is the typical summary reported for ANOVAs. One way around this, if we want to see the regression coefficients produced by the analysis is to use the `summary.lm` function on the `aov` object. We see presentation of two coding vectors, expected since there are three categories in the "factora" IV. Later we consider what the coding scheme is that is employed by default.

```
summary.lm(fit.1)
```

```
##
## Call:
## aov(formula = dv ~ factora, data = hays)
##
## Residuals:
##    Min     1Q Median     3Q    Max
##   -8.4   -2.7    0.4    2.1    8.8
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)   27.400      1.408  19.454  < 2e-16 ***
## factorafast   -6.200      1.992  -3.113  0.00435 **
## factoraslow   -5.600      1.992  -2.811  0.00907 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 4.454 on 27 degrees of freedom
## Multiple R-squared:  0.3039,Adjusted R-squared:  0.2524
## F-statistic: 5.895 on 2 and 27 DF,  p-value: 0.007513
```

The `anova` function can also be applied to the `aov` object and gives slightly different pieces of information than did application of the `summary` function above (number of significant places for F and p values).

```
anova(fit.1)
```

```
## Analysis of Variance Table
##
## Response: dv
##           Df Sum Sq Mean Sq F value   Pr(>F)
## factora    2 233.87 116.933  5.8947 0.007513 **
## Residuals 27 535.60  19.837
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

3.2.2 Using the `lm` function for an ANOVA

We can also obtain the same analysis using `lm` directly. Here, `summary` on the `lm` object gives the expected coefficients table, but `anova` on that object gives

an ANOVA summary table with slightly different characteristics.

```
# use the linear models function (lm) to do the analysis
# from this, can you determine whether lm uses dummy or effect coding?
fit.2 <- lm(dv~factora, data=hays)
summary(fit.2)
```

```
##
## Call:
## lm(formula = dv ~ factora, data = hays)
##
## Residuals:
##    Min     1Q Median     3Q    Max
##   -8.4   -2.7    0.4    2.1    8.8
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)   27.400      1.408  19.454  < 2e-16 ***
## factorafast   -6.200      1.992  -3.113  0.00435 **
## factoraslow   -5.600      1.992  -2.811  0.00907 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 4.454 on 27 degrees of freedom
## Multiple R-squared:  0.3039,Adjusted R-squared:  0.2524
## F-statistic: 5.895 on 2 and 27 DF,  p-value: 0.007513
```

```
anova(fit.2)
```

```
## Analysis of Variance Table
##
## Response: dv
##           Df Sum Sq Mean Sq F value   Pr(>F)
## factora    2 233.87 116.933  5.8947 0.007513 **
## Residuals 27 535.60  19.837
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

3.2.3 What is the default coding scheme for the factor?

If you couldn't determine whether dummy (indicator) coding or effect coding was used (based on regression coefficient values and the cell means as we have previously covered) then the following code may help. Recall the earlier document where we examined how to control/change contrast choices and that `lm` uses dummy coding by default and calls it `contr.treatment`. Note which category is the reference.

```
contrasts(hays$factora)
```

```
##         fast slow
## control    0    0
## fast       1    0
## slow       0    1
```

3.3 Analytical and orthogonal contrasts for one factor ANOVA models

If we want to employ our own contrasts we can create them and reassign those to factora. The goal is to create an orthogonal set. It is possible to request the helmert set directly, but notice that it gives what we have called "reverse helmert" and not the set that we want to use based on our prior work with this data set.

```
contrasts(hays$factora) <- "contr.helmert"
contrasts(hays$factora)
```

```
##         [,1] [,2]
## control   -1   -1
## fast       1   -1
## slow       0    2
```

There is not a method for automatically rearranging the pattern for the built-in helmert set so we need to create our own vectors. This is the preferred, and more generalizable, approach that we can use in many different designs, and with many different kinds of orthogonal sets. This method was reviewed in the earlier tutorial on coding vectors in R.

```
# now apply orthogonal contrasts of our choosing.
contrasts.factora <- matrix(c(2,-1,-1, 0,1,-1),ncol=2)
contrasts(hays$factora) <- contrasts.factora
contrasts(hays$factora)
```

```
##         [,1] [,2]
## control    2    0
## fast      -1    1
## slow      -1   -1
```

The method for requesting analysis of these contrasts employs the 'split' argument in the 'summary' function. I find it to be somewhat cumbersome, especially for factorial designs. Nonetheless, it works well, and the summary table is easily readable. Note that this "split" approach yields F tests that are based on Type I SS. See the unequal sample size chapter below for detailed discussion. In this current data set with equal N, there is no distinction between Type I and Type III SS.

```
fit.3 <- aov(dv~factora, data=hays)
summary(fit.3,
        split=list(factora=list(ac1=1, ac2=2)))
```

```
##                Df Sum Sq Mean Sq F value  Pr(>F)
## factora         2  233.9  116.93   5.895 0.00751 **
##   factora: ac1  1  232.1  232.07  11.699 0.00200 **
##   factora: ac2  1    1.8    1.80   0.091 0.76555
## Residuals      27  535.6   19.84
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

The summary table from the `lm` model object produces t-tests rather than F-tests and SS listings. However, those t's are the square roots of the F's just seen, so both approaches yield the same outcome. But this is only true when sample sizes are equal. If they are unequal, the summary table from the `lm` model fit produces tests based on the equivalence to Type III SS.

```
fit.3lm <- lm(dv~factora, data=hays)
summary(fit.3lm)
```

```
##
## Call:
## lm(formula = dv ~ factora, data = hays)
##
## Residuals:
##    Min     1Q Median     3Q    Max
##   -8.4   -2.7    0.4    2.1    8.8
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  23.4667     0.8132  28.858   <2e-16 ***
## factora1      1.9667     0.5750   3.420    0.002 **
## factora2     -0.3000     0.9959  -0.301    0.766
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 4.454 on 27 degrees of freedom
## Multiple R-squared:  0.3039,Adjusted R-squared:  0.2524
## F-statistic: 5.895 on 2 and 27 DF,  p-value: 0.007513
```

If we apply this `split` approach to the `lm` object, an identical table of results is produced, as expected. This request is the way to obtain the parameter estimates. Note the equivalence to what we found using the "manual" regression procedure in SPSS. It is also equivalent to the first summary table seen just above for the fit.3lm model object.

```
fit.4 <- lm(dv~factora, data=hays)
summary(fit.4,
        split=list(factora=list(ac1=1, ac2=2)))
```

```
##
## Call:
## lm(formula = dv ~ factora, data = hays)
##
## Residuals:
##    Min     1Q Median     3Q    Max
##   -8.4   -2.7    0.4    2.1    8.8
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  23.4667     0.8132  28.858   <2e-16 ***
## factora1      1.9667     0.5750   3.420    0.002 **
## factora2     -0.3000     0.9959  -0.301    0.766
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 4.454 on 27 degrees of freedom
## Multiple R-squared:  0.3039,Adjusted R-squared:  0.2524
## F-statistic: 5.895 on 2 and 27 DF,  p-value: 0.007513
```

An alternative, and perhaps simpler, way to obtain the contrasts is to apply the `summary` function to an `aov` object. We saw above that that applying `summary` to an `aov` object produces tests of the contrasts, only when the `split` argument was used (code repeated here).

```
summary(fit.3,
        split=list(factora=list(ac1=1, ac2=2)))
```

However, since `aov` is a wrapper to `lm` we can use the `summary` function in a different way by calling `summary.lm`. This will produce the same t-tests of the specified contrasts that we saw above when `summary` was applied to an `lm` fit object. This is a good way to do analyses only using `aov`, and the t-tests in the table are equivalent to those shown above when starting with the `lm` object, and thus equivalent to Type III SS decompositions. For the oneway design, these t's are the square roots of the F tests for the contrasts produced when using the `split` function on the `aov` object.

```
summary.lm(fit.3)
```

```
##
## Call:
## aov(formula = dv ~ factora, data = hays)
##
```

```
## Residuals:
##    Min     1Q Median     3Q    Max
##   -8.4   -2.7    0.4    2.1    8.8
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  23.4667     0.8132  28.858   <2e-16 ***
## factora1      1.9667     0.5750   3.420    0.002 **
## factora2     -0.3000     0.9959  -0.301    0.766
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 4.454 on 27 degrees of freedom
## Multiple R-squared:  0.3039,Adjusted R-squared:  0.2524
## F-statistic: 5.895 on 2 and 27 DF,  p-value: 0.007513
```

The reader is urged to compare this table to the output from SPSS REGRESSION where we first utilized this orthogonal contrast set with the hays data set.

3.3.1 Manually computing contrast SS for verification

To be thorough, and to connect these analyses to the basic formulaic concepts put in place in initial lectures, let's see if we can find a way to manually compute these contrast SS using some matrix/vector manipulation in R. First we need to find the linear combination "psi" value - only done here for the first vector, the 2 -1 -1 contrast.

```
# First lets obtain the group means and place them in a vector.
# We can do this two ways.
# a simple way is to use the 'aggregate' function
# which produces a data frame of the means
xbars <- aggregate(dv~factora, hays, mean)
xbars
```

```
##   factora   dv
## 1 control 27.4
## 2    fast 21.2
## 3    slow 21.8
```

```
#str(xbars)
# a second way to extract the means by group is using 'dplyr',
# pipes, and a summarizing function.
# not run here
#@xbars <-
#hays %>%
#  group_by(factora) %>%
#  summarise (mean_dv = mean(dv))
```

```
# now extract the means from the dataframe and create a vector
v1 <- xbars$dv
v1
```

```
## [1] 27.4 21.2 21.8
```

```
#extract the first contrast from the contrasts matrix
v2 <- contrasts(hays$factora)[,1]
v2
```

```
## control    fast    slow
##       2      -1      -1
```

```
# For our purposes it doesn't matter if these vectors
# are row or column vectors.
# we just need the dot product
# take the dot product of these vectors.
vectorprod <- v1%*%v2
vectorprod
```

```
##      [,1]
## [1,] 11.8
```

Once we have this "psi" quantity, we can use our generic SS formula to compute SS. We have seen this same 11.8 value in our earlier work with this data set, several times actually.

$$\text{SS}_{AnalyticalContrast} = \frac{n_j*(\Psi)^2}{\sum_{j-1}^{k} c_j^2}$$

For our data set n=10 and the sum of the squared coefficients is 6. So,

```
SS_ac1 <- (10*(11.8**2))/6
SS_ac1
```

```
## [1] 232.0667
```

This matches the SS for the first contrast produced with the 'split' function, as applied to the `aov` fit.3 object above.

3.3.2 The Bonferroni family and other alpha-inflation control methods for analytical contrasts

If we need to control of post hoc error rate inflation with the tests of these contrasts, R has a useful function called `p.adjust`. The types of adjustments are in the 'bonferroni' style of control. `p.adjust permits choice of a large number of methods. Output from only one of them is shown here, but the logic for changing to the others is obvious from`

the commented code. The reader should look at thep.adjust‘ help page.

For our orthogonal set, any of the methods listed here are appropriate. However, since there are only two contrasts, we don't find any difference among the approaches.

So, the laying out of the different approaches can serve as a template for designs where you might have more than two contrasts. Note that I had to submit the p values for the each of the contrasts examined in the preceding summary(fit.4)specification.

The result here is almost too trivial to need the software. With two tests, the bonferroni adjust is just to double the observed p-value. And since only the first was significant above at the nominal alpha level of .05 it is the only one we really had to adjust and we could have multiplied it by two in our heads. Nonetheless, this puts in place a template for usage in more complex situation.

Understand that the logic is to submit p-values of ALL followup tests so that the number of tests can be determined by 'p.adjust' in its computation.

With the Bonferroni adjustment, our first contrast remains significant since .004 $<$ .05.

Later in this document several other multiple comparison and post hoc methods are reviewed.

```
# now do a bonferroni adjustment on the p values from these two contrasts.
# we can use the p.adjust function for this.  it has many other
# options for modified bonferroni adjustments - read the help: ?p.adjust
# For our orthogonal/independent contrasts, the bonferroni, holm, hochberg, and
# Benjamini/Hochberg (fdr) method are appropriate.  However, since there are only two
# contrasts, we don't find any difference among the approaches.
# So, the laying out of the different approaches can serve as a template for
# designs where you might have more than two contrasts.
# note that I had to submit the p values for the each of the contrasts
# examined in the preceding summary(fit.4)specification:
p.adjust(c(.00200,.07655), method="bonf")
```

```
## [1] 0.0040 0.1531
```

```
#p.adjust(c(.00200,.07655), method="holm")
#p.adjust(c(.00200,.07655), method="hoch")
#p.adjust(c(.00200,.07655), method="hommel")
#p.adjust(c(.00200,.07655), method="fdr")
```

3.4 A note about testing analytical contrasts in R

While the above method using the 'split' function is workable for the 1-factor ANOVA model, it can become quite challenging to extend its usage to complex designs with multiple factors, and especially with repeated measures designs.

The **phia** package will provide some capability for those broader needs.

The **granova** package has a function that permits use of contrasts, in a 1-way, and is exemplified below.

The most flexible approach is probably using the **emmeans** package. It's usage for our 1-way Hays design is found in the Post Hoc and Multiple comparison section below and will be revisited with later designs. Even though I placed its usage illustration in the "post hoc" chapter, **emmeans** can also be used for planned contrasts.

3.5 Use of the 'Anova' function from the car package

Above, we discussed the fact that use of the base package 'anova' function on an 'aov' object produced Type I SS. The issue of Type I vs Type III SS is not relevant for the current data set since there is equal n in the groups. Nonetheless, it is useful to put in place an introduction to the use of the 'Anova' function here and the specification required to obtain Type III SS.

An `Anova` argument called "type" permits request of the various "types" of SS. Once again, we see that with equal N, the TYPE III SS equals the Type I SS (and F/p values) produced by `anova` in various illustrations above.

```
#require(car)
Anova(fit.1,type="III")
```

```
## Anova Table (Type III tests)
##
## Response: dv
##             Sum Sq Df  F value    Pr(>F)
## (Intercept) 7507.6  1 378.4638 < 2.2e-16 ***
## factora      233.9  2   5.8947  0.007513 **
## Residuals    535.6 27
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

3.6 Recommended approach for a basic 1-way ANOVA with planned contrasts

Although this section has reviewed many options, the core initial approach to a 1-way ANOVA can be fairly quick and direct with just a few steps:

1. Choose a set of planned contrasts and create the orthogonal set.
2. Perform the omnibus ANOVA with the `aov` function
3. Examine the omnibus F test and the single df contrast tests with either the "split" capability or with `summary.lm`
4. Follow up with post hoc tests, effect size computations, evaluation of assumptions and possible alternative methods outlined in later chapters and/or the `p.adjust` methods shown earlier in this chapter.

Here is the sequence (repeating the functions shown above):

First create the contrasts:

```
contrasts.factora <- matrix(c(2,-1,-1, 0,1,-1),ncol=2)
contrasts(hays$factora) <- contrasts.factora
contrasts(hays$factora)
```

```
##         [,1] [,2]
## control    2    0
## fast      -1    1
## slow      -1   -1
```

Perform the base omnibus ANOVA and partition the factor into contrasts using "split":

```
fit.3 <- aov(dv~factora, data=hays)
summary(fit.3,
        split=list(factora=list(ac1=1, ac2=2)))
```

```
##                Df Sum Sq Mean Sq F value  Pr(>F)
## factora         2  233.9  116.93   5.895 0.00751 **
##   factora: ac1  1  232.1  232.07  11.699 0.00200 **
##   factora: ac2  1    1.8    1.80   0.091 0.76555
## Residuals      27  535.6   19.84
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

or, obtain inferences on the contrasts by using `summary.lm`. This will produce "t-tests" of the coding vectors, but in an equal-N situation the squares of the t's will equal the F's. With unequal N's the t-tests are tests equivalent to Type III SS (a more important issue in Factorial designs than in 1-way designs)

```
summary.lm(fit.3)
```

```
##
```

```
## Call:
## aov(formula = dv ~ factora, data = hays)
##
## Residuals:
##    Min     1Q Median     3Q    Max
##   -8.4   -2.7    0.4    2.1    8.8
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  23.4667     0.8132  28.858   <2e-16 ***
## factora1      1.9667     0.5750   3.420    0.002 **
## factora2     -0.3000     0.9959  -0.301    0.766
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 4.454 on 27 degrees of freedom
## Multiple R-squared:  0.3039,Adjusted R-squared:  0.2524
## F-statistic: 5.895 on 2 and 27 DF,  p-value: 0.007513
```

Chapter 4

Alternatives to `aov` and `lm` for 1-way ANOVA

Several add-on packages provide alternate approaches to doing ANOVAs. Both **ez** and **afex** are broadly capable and attempt to simplify some of the techniques to doing the various components of analyzing different designs. A choice among them, vs using `aov` and/or `lm` is largely a matter of style and preference but each adds components that may be desired. The **granova** functions are useful for exploring some of the theoretical components of ANOVA, using unique graphical approaches. They are probably most helpful as instructional tools, and the 1-way design illustration here is an example of that.

4.1 The `oneway` function from the userfriendlyscience package

The **userfriendlyscience** package has a group of functions that are intended to ease the transistion of researchers from use of SPSS to R. It contains a flexible function named `oneway` for implementation of 1-way ANOVA.

A potentially nice graph that shows data points and violin plots for each group can also be provided, but unfortunately it creates a line graph. Corrections for heterogeneity of variance are also provided.

The function has a capability of implementing the Levene test, but since it is the mean-centered version, that argument is not included here. Arguments for returning means and other descriptive statistics are present but not included here since, as of this writing, it produces an error in how the `psych::describeBy` function is implemented.

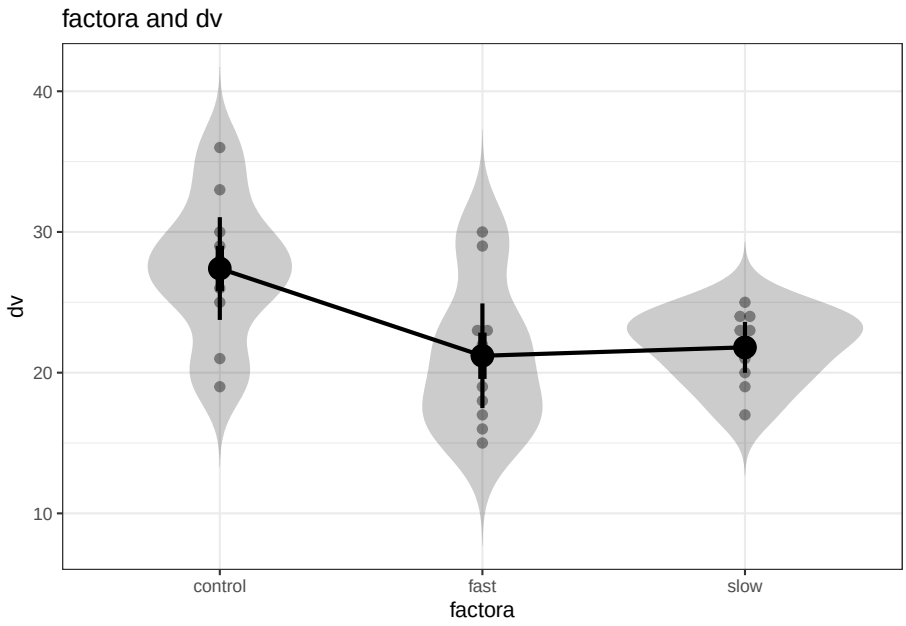
In a later chapter, we will see how to obtain post hoc tests by using a "posthoc"

argument.

```
oneway(y=hays$dv, x=hays$factora, plot=TRUE, corrections=TRUE)
```

```
## ### Oneway Anova for y=dv and x=factora (groups: control, fast, slow)
```

```
## Registered S3 methods overwritten by 'ufs':
##   method                       from
##   grid.draw.ggProportionPlot userfriendlyscience
##   pander.associationMatrix   userfriendlyscience
##   pander.dataShape           userfriendlyscience
##   pander.descr               userfriendlyscience
##   pander.normalityAssessment userfriendlyscience
##   print.CramersV             userfriendlyscience
##   print.associationMatrix    userfriendlyscience
##   print.confIntOmegaSq       userfriendlyscience
##   print.confIntV             userfriendlyscience
##   print.dataShape            userfriendlyscience
##   print.descr                userfriendlyscience
##   print.ggProportionPlot     userfriendlyscience
##   print.meanConfInt          userfriendlyscience
##   print.multiVarFreq         userfriendlyscience
##   print.normalityAssessment  userfriendlyscience
##   print.scaleDiagnosis       userfriendlyscience
##   print.scaleStructure       userfriendlyscience
##   print.scatterMatrix        userfriendlyscience
```



```
## Omega squared: 95% CI = [.03; .5], point estimate = .25
## Eta Squared: 95% CI = [.06; .46], point estimate = .3
##
##                                      SS Df     MS    F    p
## Between groups (error + effect) 233.87  2 116.93 5.89 .008
## Within groups (error only)       535.6 27  19.84
##
##
## ### Welch correction for nonhomogeneous variances:
##
## F[2, 15.91] = 5.07, p = .02.
##
## ### Brown-Forsythe correction for nonhomogeneous variances:
##
## F[2, 21.95] = 5.89, p = .009.
```

4.2 Using the granova package

The *__granova__ package provides a unique approach to the oneway ANOVA design. It includes a rather neat graph that summarizes the ANOVA components. BEWARE, the "contrast coefficients" are not the analytical contrast types of coefficients we have been working with. Instead, we would call these treatment deviation contrasts - deviations of each cell mean from the grand mean. These are thus the regression coefficient values that emerge from effect coding.

The `granova.1w` function produces both a standard numeric output and a figure. The components of the figure are described in class, but it is simple to find the grand mean, the cell means and the data on the plot.

One caution about implementing `granova.1w`:

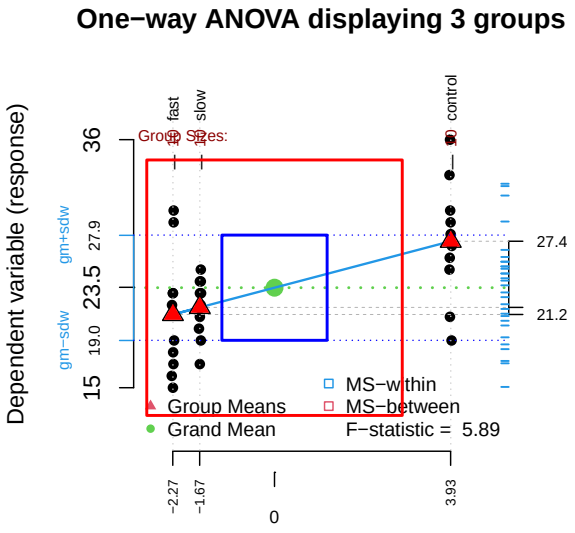
`granova.1w` can take either a data frame as the first argument, or the DV and IV variable names as the first two arguments. If we were passing the data frame as the first argument, the data frame must meet exact specifications. Only DV values can be included and the different columns must be the different groups - equal sample sizes required. This is a bit of a clunky method so we will use the second method. We execute the code by passing the DV and IV names, and can do that because the "hays" data frame is attached.

When the numerator and denominator MS for the F test are depicted as "squares", it becomes clear that the F ratio is the ratio of the area of MSbetween to MSwithin. The X axis is scaled as the deviation quantity for each cell mean relative to the grand mean; therefore the X axis has a mean of zero. Unfortunatley for the color blind, the red triangles for cell means/deviations and the green circle for the grand mean may not be distinguishable via color but the shapes are distinct.

	x
Grandmean	23.47
df.bet	2.00
df.with	27.00
MS.bet	116.93
MS.with	19.84
F.stat	5.89
F.prob	0.01
SS.bet/SS.tot	0.30

	Size	Contrast Coef	Wt'd Mean	Mean	Trim'd Mean	Var.	St. Dev.
fast	10	-2.27	21.2	21.2	20.33	27.07	5.20
slow	10	-1.67	21.8	21.8	22.17	6.40	2.53
control	10	3.93	27.4	27.4	27.50	26.04	5.10

```
#require(granova)
kable(granova.1w(hays$dv, hays$factora,res=TRUE))
```

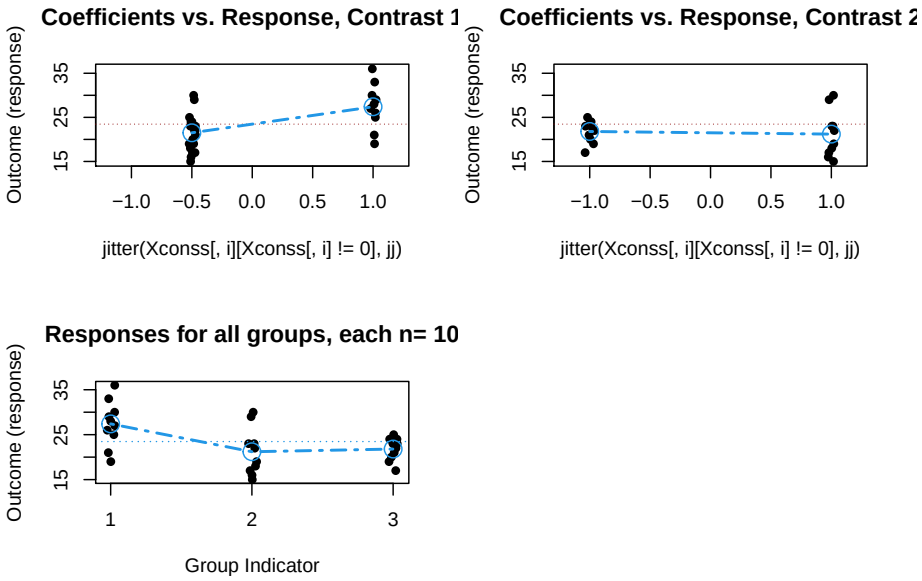


Contrast coefficients based on group means and sizes

Granova also permits use of predifined, a priori, contrasts of the style we have used so heavily, with the `granova.contr` function. If we define the same orthogonal set used above (for fit.3 and fit.4), then the `granova.contr` function generates some nice plots along with the tests of the contrasts, and a standardized effect size estimate for each contrast. Very Nice. BUT BE CAREFUL……'granova.contr' needs to have equal N, and a data frame sorted on the

factor (the grouping variable, in the order implied by the contrast coefficients.

```
contrasts.factora <- matrix(c(2,-1,-1, 0,1,-1),ncol=2)
granova.contr(hays$dv,contrasts.factora)
```




```
## $summary.lm
##
## Call:
## lm(formula = resp ~ contrst)
##
## Residuals:
##    Min     1Q Median     3Q    Max
##   -8.4   -2.7    0.4    2.1    8.8
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  23.4667     0.8132  28.858   <2e-16 ***
## contrst1      3.9333     1.1500   3.420    0.002 **
## contrst2     -0.3000     0.9959  -0.301    0.766
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 4.454 on 27 degrees of freedom
## Multiple R-squared:  0.3039,Adjusted R-squared:  0.2524
## F-statistic: 5.895 on 2 and 27 DF,  p-value: 0.007513
##
```

```
##
## $means.pos.neg.coeff
##    neg  pos diff stEftSze
## 1 21.5 27.4  5.9     1.32
## 2 21.8 21.2 -0.6    -0.13
##
## $contrasts
##      [,1] [,2]
## [1,]  1.0    0
## [2,] -0.5    1
## [3,] -0.5   -1
##
## $group.means.sds
##       [,1] [,2]  [,3]
## Means 27.4 21.2 21.80
## S.D.s  5.1  5.2  2.53
##
## $data
##       [,1] [,2] [,3]
##  [1,]   27   23   23
##  [2,]   28   22   24
##  [3,]   33   18   21
##  [4,]   19   15   25
##  [5,]   25   29   19
##  [6,]   29   30   24
##  [7,]   36   23   22
##  [8,]   30   16   17
##  [9,]   26   19   20
## [10,]   21   17   23
```

Conclusion:

Granova is a fascinating approach. It gives all of the basics we need. The graphs that are produced are very helpful in an instructional context, reminding us what "Mean Squares" means. But I have never seen this plot used for a research publication. One would likely have to spend a large amount of time convincing an editor and reviewers of its value.

The recommendation here is to use it as a sort of exploratory device rather than for the production of a publication quality graph.

4.3 Use of the ez package

Among a number of additional ways to obtain ANOVA results for experimental designs, the **ez** package intends to be comprehensive ANOVA modeling suite. It requires a unique way of specifying the DV, IV, and several other components.

Once the syntax is learned, many R users have found it to be efficient. I find that it adds nothing to what we have accomplished above for 1-way designs, but it may be useful for factorial designs.

The EZ package contains the ezAnova function which serves as a wrapper for many of the things shown above. It is a quick way to obtain a fairly full analysis. It does not appear to enable inferences regarding analytical/orthogonal contrasts. **emmeans** would have to be used as a supplement in order to obtain them.

Note at the outset that functions from **ez** require a data frame that contains a case number (subject number) variable. Since the original hays data frame was sorted by group, it made it easy to simply specify subject (case) number as the sequential order number of the case in the data frame. Since the **ez** ANOVA functions permit an argument that specifies the data frame, we don't have to be concerned about conflicts arising because the original hays data frame was "attached".

```
#require(ez)
# in order to use ez functions we need a subject number variable in the data frame
snum <- ordered(rownames(hays))
hays2 <- cbind(snum,hays)
# examine hays2
headTail(hays2)
```

```
##     snum factora  dv
## 1      1 control  27
## 2      2 control  28
## 3      3 control  33
## 4      4 control  19
## ... <NA>    <NA> ...
## 27    27    slow  22
## 28    28    slow  17
## 29    29    slow  20
## 30    30    slow  23
```

```
# first, obtain descriptives; note that FLSD is Fisher's LSD critical value
# note that Fisher's Least Significant Difference is computed as
# sqrt(2)*qt(.975,DFd)*sqrt(MSd/N), where N is taken as the mean N
# per group in cases of unbalanced designs.
descript1 <- ezStats(
           data = hays2,
                   dv = .(dv),
                   wid= .(snum),
                   between= .(factora))
```

```
## Coefficient covariances computed by hccm()
```

```
print(descript1)
```

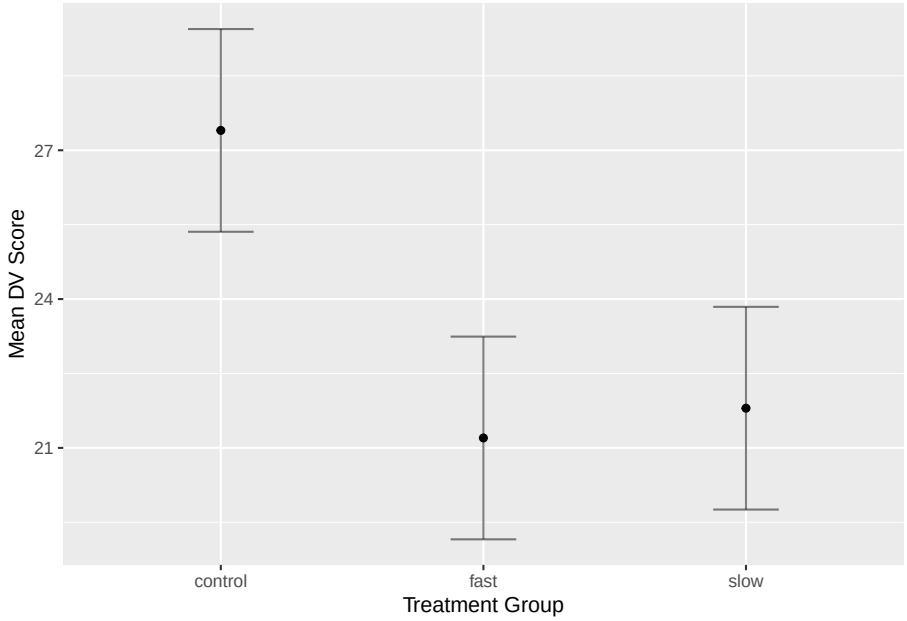
```
##   factora  N Mean       SD     FLSD
## 1 control 10 27.4 5.103376 4.086908
## 2    fast 10 21.2 5.202563 4.086908
## 3    slow 10 21.8 2.529822 4.086908
```

The `ezPlot` function provides graphing capabilities tailored to the experimental design. I found it interesting that the error bars on the figure are derived from a generalized standard error of the mean based on the MSerror from the ANOVA, rather than the individual standard errors for each group, which are not actually used for anything.

```
# now plot the cell means and error bars.
# error bars based on GSEM as described above for Fishers LSD
plot1 <- ezPlot(
           data = hays2,
                   dv = .(dv),
                   wid= .(snum),
                   between= .(factora),
                   x = .(factora),
                   do_lines=FALSE, do_bars=TRUE,
                   x_lab= 'Treatment Group',
                   y_lab= 'Mean DV Score')
```

```
## Coefficient covariances computed by hccm()
```

```
print(plot1)
```



Use of the `ezANOVA` function involves a novel style of code, using a non-intuitive dot notation for specifying IVs, but one becomes accustomed to this fairly quickly. Once again, the same values for SS/F/pvalue are produced. An added benefit is that the Levene homogeneity of variance test is automatically generated. It is important that an effect size indicator is also provided ("ges", discussed elsewhere - or read the help page on the **ez** package).

```
# now do the 1way ANOVA
fit.4ez <- ezANOVA(
           data = hays2,
                   dv = .(dv),
                   between= .(factora),
                   wid=snum,
                   detailed=TRUE)
```

```
## Coefficient covariances computed by hccm()
```

```
print(fit.4ez)
```

```
## $ANOVA
##    Effect DFn DFd      SSn   SSd        F           p p<.05       ges
## 1 factora   2  27 233.8667 535.6 5.894698 0.007512513     * 0.3039335
##
## $`Levene's Test for Homogeneity of Variance`
##   DFn DFd      SSn   SSd       F         p p<.05
## 1   2  27 27.46667 184.7 2.00758 0.1538727
```

By default, `ezANOVA` employs Type II SS decompositions. Although this distinction is only relevant for factorial designs (and unequal sample sizes) it would be useful to put in place the argument that permits specification of other SS types. The following code requests Type III SS. Unfortunately the output does not label the SS Type, so one has to be certain with the code.

```
# now do the 1way ANOVA
fit.4bez <- ezANOVA(
           data = hays2,
                   dv = dv,
                   between= .(factora),
                   wid=snum,
                   type=3,
                   detailed=TRUE)
```

```
## Coefficient covariances computed by hccm()
```

```
print(fit.4bez)
```

```
## $ANOVA
##        Effect DFn DFd        SSn   SSd          F            p p<.05       ges
## 1 (Intercept)   1  27 16520.5333 535.6 832.812547 7.900265e-22     * 0.9685978
## 2     factora   2  27   233.8667 535.6   5.894698 7.512513e-03     * 0.3039335
##
## $`Levene's Test for Homogeneity of Variance`
##   DFn DFd      SSn   SSd       F         p p<.05
## 1   2  27 27.46667 184.7 2.00758 0.1538727
```

EZ permits resampling tests. Here, we implement a permutation test. I show the code but suppress a lengthy status printout while it is doing the resampling. The result of the test is obtained in the next code chunk. The permutation algorithm is fairly slow and I don't know if the `ezPerm` function can address analytical contrasts. We will revist permutation tests in a later chapter.

```
# now do the 1way anova as a permutation test
fit.5ezperm <- ezPerm(
           data = hays2,
                   dv = dv,
                   between= .(factora),
                   snum,
                   perm= 1e3)
```

```
fit.5ezperm
```

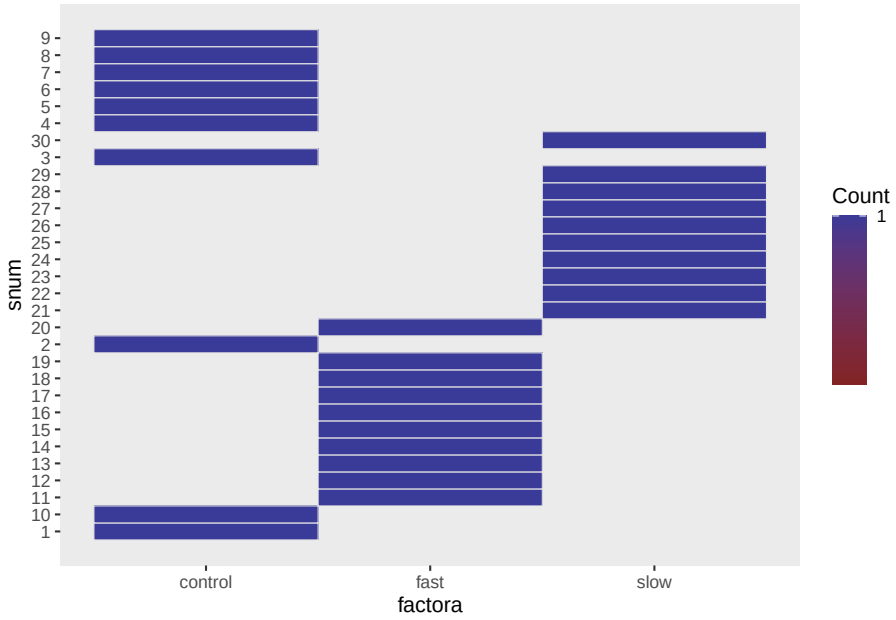
```
##    Effect     p p<.05
## 1 factora 0.006     *
```

The **ez** packages has functions to perfom bootstraping, but right now older code that I used is broken and I've not yet sorted out why, so bootstrapping excluded

for now. We can return to bootstrapping in a later chapter.

One additional capability in **ez** is an interesting graphical approach to displaying the data set. T This plot is Not the values of the DV, but locations where the various groups of cases are in the ordering of the data frame.

```
# other useful functions in the ez package
#win.graph()
ezDesign(
           data = hays2,
                   x= .(factora),
                   y= .(snum),
                   row=NULL,col=NULL,
                   cell_border_size=8)
```



4.4 Using the afex package

The **afex** package is a wrapper for `lm`, **car**, and **lme4** functions, the latter to do mixed models. The stated goals are a comprehensive set of functions for virtually all kinds of ANOVA models. My initial investigations into its usage have engendered a mix of optimism and concern over the scope of what it can do, the kinds of decisions that were made about restricting some kinds of ANOVA approaches, and ease of use. What is presented here is the bare bones analysis of our 3 group design. This section may expand as I become more skilled with **afex**.

First, lets use the "car" flavor of the afex approach to model the 3-group data set. Note that **afex** requires a data frame that has an identifier variables (such as subject number). I created such a data frame for the **ez** package functions above, so we can use the same "hays2" data frame here as well. Note that even though we have "attached" the original hays data frame, there is no conflict that arises by using the same variable names in hays2. This is because the **afex** functions permit naming the data frame as an argument.

In using afex, we choose one of three styles of specifying the model. The first shown here models on how the specification would be made in doing ANOVAs using functions from the **car** package. The **afex** package functon is thus `aov_car`. Unlike what we have seen above, this approach requires specification of the error factor - the snum variable is the case identifier and serves this role. We can think of the error as the variation in the DV due to snum within the factora levels.

Note that the `aov_car` function changes factor coding to what we have called "effect" coding (also known as deviation coding). In R, this is defined as `contr.sum`.

```
# needs data frame with subj id. use hays2.csv
fitx1 <- aov_car(dv~factora + Error(snum), data=hays2)
```

```
## Contrasts set to contr.sum for the following variables: factora
```

```
fitx1
```

```
## Anova Table (Type 3 tests)
##
## Response: dv
##    Effect    df   MSE       F  ges p.value
## 1 factora 2, 27 19.84 5.89 ** .304    .008
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '+' 0.1 ' ' 1
```

A bit better layout and info provision is possible with the 'return' argument, although we lose the "ges" effect size indicator.

```
# obtain a fuller table but without the effect size
fitx2 <- aov_car(dv~factora + Error(snum), data=hays2, return="univariate" )
```

```
## Contrasts set to contr.sum for the following variables: factora
```

```
fitx2
```

```
## Anova Table (Type III tests)
##
## Response: dv
##              Sum Sq Df  F value    Pr(>F)
## (Intercept) 16520.5  1 832.8125 < 2.2e-16 ***
## factora       233.9  2   5.8947  0.007513 **
```

```
## Residuals     535.6 27
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Notice that with `aov_car`, the SS decomposition is, by default, Type III. Adding a "type" argument permits control of this characteristic. But in our 1way design with equal N, there is no difference in Type I, II, and III SS. The illustration is done here as a placeholder for what we will do later with factorial designs.

```
# obtain a fuller table but without the effect size
fitx2b <- aov_car(dv~factora + Error(snum), data=hays2, return="univariate", type=2)
```

```
## Contrasts set to contr.sum for the following variables: factora
```

```
fitx2b
```

```
## Anova Table (Type II tests)
##
## Response: dv
##           Sum Sq Df F value   Pr(>F)
## factora   233.87  2  5.8947 0.007513 **
## Residuals 535.60 27
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Postscript on current status of **afex** usage here:

Some advantages of using **afex** for other designs will be seen in later work with factorial and repeated measures designs. However, for our one-way anova needs here, this package adds only one thing. That is the "generalized effect size" (called ges) obtained with fitx1. In a oneway model, the ges is identical to the eta squared. The basic ANOVA with SS, df, F's and pvalue are obviously the same as we have derived many other ways earlier in this document.

In conjunction with 'glht' and **emmeans**, **afex** has some broadly capable methods for implementing analytical contrasts and post hoc testing methods. Some of those illustrations are found below.

Chapter 5

Post Hoc and Multiple Comparison Methods

Could be a very long section..... I am trying to keep it brief as a template, rather than an extended exposition on multiple comparisons. For all of these functions the user should carefully examine the help pages and any documentation vignettes before using them. In some cases, the function can be applied to ANOVA model objects already created, such as our `aov` and `lm` fits from above (fit.1 and fit.2). Other multiple comparison functions permit specification of the design within the function arguments.

5.1 The commonly used TUKEY test

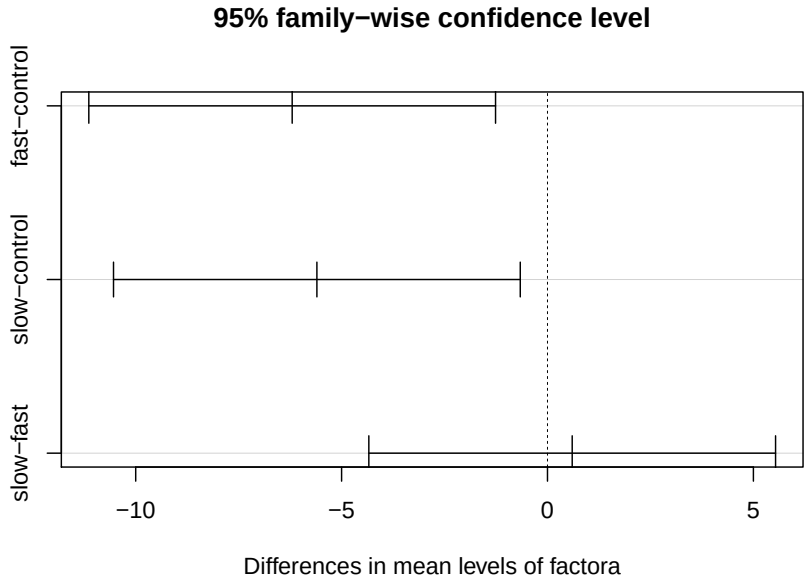
Tukey's HSD is easily accomplished on an 'aov' fit via a function in the base stats package that is loaded upon startup. We can use the original fit.1 `aov` object.

```
tukeyfit1 <- TukeyHSD(fit.1, conf.level=.95)
tukeyfit1
```

```
##   Tukey multiple comparisons of means
##     95% family-wise confidence level
##
## Fit: aov(formula = dv ~ factora, data = hays)
##
## $factora
##              diff        lwr        upr     p adj
## fast-control -6.2 -11.138591 -1.2614086 0.0117228
## slow-control -5.6 -10.538591 -0.6614086 0.0238550
## slow-fast     0.6  -4.338591  5.5385914 0.9513012
```

A graphical depiction of the result from 'TukeyHSD' function is easily obtained.

```
plot(tukeyfit1)
```



5.2 Using the asbio package for Post Hoc pairwise comparisons

Several functions in **asbio** are helpful for various multiple comparison tests. They each produce all possible pairwise group comparisons. The reader should utilize the help (e.g., '?lsdCI') for documentation on these functions to see the array of options available.

First, the so-called Fisher's Protected LSD test is obtained. Recall that when there are three groups, simulation work has shown that performing the LSD tests following a significant omnibus F test does afford protection from error rate inflation. But when the design has more than three groups, the LSD test CANNOT be recommended.

Several functions below simply specify the DV and IV as vectors, which can be done since the hays data frame is attached.

```
# The asbio package has several functions that permit pairwise post hoc
# multiple comparison tests:
#require(asbio)
lsdCI(hays$dv,hays$factora)
```

```
##
## 95% LSD confidence intervals
##
##                       LSD Diff    Lower    Upper  Decision Adj. p-value
## mucontrol-mufast 4.08691  6.2  2.11309 10.28691 Reject H0      0.00435
## mucontrol-muslow 4.08691  5.6  1.51309  9.68691 Reject H0      0.00907
## mufast-muslow    4.08691 -0.6 -4.68691  3.48691    FTR H0      0.76555
```

Next is a simple bonferroni adjustment.

```
#require(asbio)
bonfCI(hays$dv,hays$factora)
```

```
##
## 95% Bonferroni confidence intervals
##
##                   Diff    Lower    Upper  Decision Adj. p-value
## mucontrol-mufast  6.2  1.11592 11.28408 Reject H0     0.013052
## mucontrol-muslow  5.6  0.51592 10.68408 Reject H0     0.027217
## mufast-muslow    -0.6 -5.68408  4.48408    FTR H0            1
```

asbio also has a function for the standard Tukey test.

```
#require(asbio)
tukeyCI(hays$dv,hays$factora)
```

```
##
## 95% Tukey-Kramer confidence intervals
##
##                   Diff    Lower    Upper  Decision Adj. p-value
## mucontrol-mufast  6.2  1.26141 11.13859 Reject H0     0.011723
## mucontrol-muslow  5.6  0.66141 10.53859 Reject H0     0.023855
## mufast-muslow    -0.6 -5.53859  4.33859    FTR H0     0.951301
```

Implementation of the Scheffe test is also possible

```
#require(asbio)
scheffeCI(hays$dv,hays$factora)
```

```
##
## 95% Scheffe confidence intervals
##
##                   Diff    Lower    Upper  Decision Adj. P-value
## mucontrol-mufast  6.2  1.04108 11.35892 Reject H0     0.015929
## mucontrol-muslow  5.6  0.44108 10.75892 Reject H0     0.031227
## mufast-muslow    -0.6 -5.75892  4.55892    FTR H0     0.955717
```

In designs with a control group and several treatment conditions, a recommended approach is the use of the Dunnett test. This controls alpha-inlation for all pairwise comparisons involving the control condition vs each treatment.

```
# the asbio package permits implementation of the Dunnett test
# with specification of the ANOVA model and the level of the control group
#require(asbio)
dunnettCI(hays$dv,hays$factora,control="control")
```

```
##
## 95% Dunnett confidence intervals
##
##                   Diff      Lower     Upper  Decision
## mufast-mucontrol -6.2 -10.848181 -1.551819 Reject H0
## muslow-mucontrol -5.6 -10.248181 -0.951819 Reject H0
```

5.3 Additional multiple comparison functions

An alternative function for performing the Dunnett test is found in **multcomp**. With any future work in R, you will see frequent use of the `ghlt` and `mcp` functions. One can simply pass the 'aov' fit object to the function. It is also possible to change the type of MC test to other "flavors" withthe 'linfct' argument. See the help page on this function. `ghlt` is a very widely used function for multiple comparisons.

```
# The Dunnett test can also be obtained from the multcomp package.
#require(multcomp)
fit1.dunnett <- glht(fit.1,linfct=mcp(factora="Dunnett"))
# obtain CI's to test each difference
confint(fit1.dunnett, level = 0.95)
```

```
##
##   Simultaneous Confidence Intervals
##
## Multiple Comparisons of Means: Dunnett Contrasts
##
##
## Fit: aov(formula = dv ~ factora, data = hays)
##
## Quantile = 2.3335
## 95% family-wise confidence level
##
##
## Linear Hypotheses:
##                      Estimate lwr      upr
## fast - control == 0  -6.2000 -10.8480  -1.5520
## slow - control == 0  -5.6000 -10.2480  -0.9520
```

The **DTK** package implements an alternative to the Tukey HSD method that permits application to data sets with unequal N's and heterogenous variances.

Notice that the CI's are not the same for the base Tukey application and the DTK application owing to this capability for unequal N's (which we don't have) and heterogeneity.

Once again, the DV and IV are specified without the data frame name since the hays data frame is attached.

```
#require(DTK)
# first, repeat the Tukey HSD test procedure in DTK to compare to above:
TK.test(hays$dv,hays$factora) # should give the same outcome as both to Tukey HSD approaches
```

```
##   Tukey multiple comparisons of means
##     95% family-wise confidence level
##
## Fit: aov(formula = x ~ f)
##
## $f
##              diff        lwr        upr     p adj
## fast-control -6.2 -11.138591 -1.2614086 0.0117228
## slow-control -5.6 -10.538591 -0.6614086 0.0238550
## slow-fast     0.6  -4.338591  5.5385914 0.9513012
```

```
# now request the Dunnett-Tukey-Kramer test:
DTK.test(hays$dv,hays$factora)
```

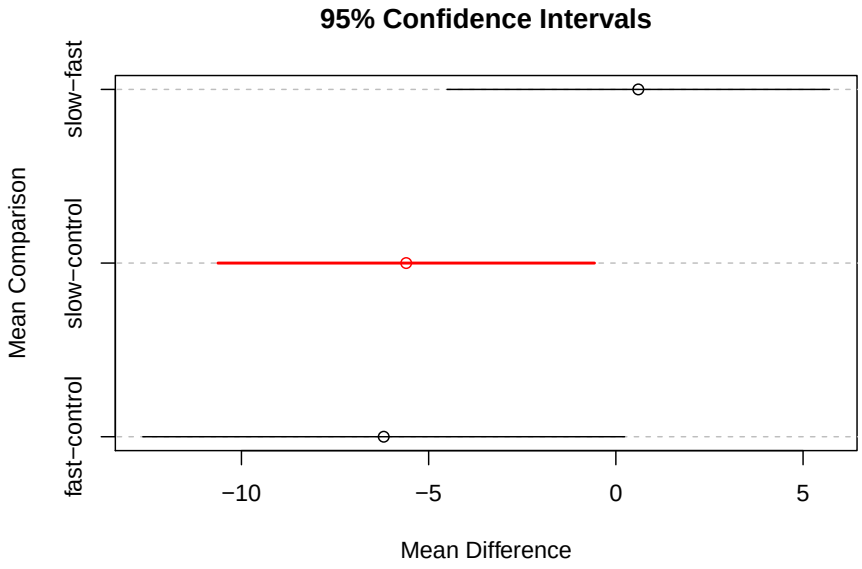
```
## [[1]]
## [1] 0.05
##
## [[2]]
##              Diff   Lower CI   Upper CI
## fast-control -6.2 -12.634414  0.2344137
## slow-control -5.6 -10.629056 -0.5709442
## slow-fast     0.6  -4.507666  5.7076663
```

The outcome with the Hays data set may be surprising. Note that, with 'DTK',the first comparison, fast vs ctl has a CI that now overlaps zero, and is thus "NS". The second comparison, slow vs ctl is a smaller mean difference, but the adjusted CI does not overlap zero. This difers from the standard Tukey HSD test.

Confusing? now look back at the descriptives stats for these three groups. note that even though the homogeneity of variance tests were NS (a later section in this document), the std dev is smaller for the "slow" group. Since DTK does not use the pooled within group variance as the "error term" and uses Welch or Fisher-Behrens type error, the differences in the within-group variances can affect the outcome. The error df will also be smaller since the pooled MS isnot used. Such discrepancies from "expected" outcomes can become even more extreme when n's are unequal. This is a nice function to have available.

We can also plot the CI's derived from DTK.

```
DTK.result <- DTK.test(hays$dv,hays$factora)
DTK.plot(DTK.result)
```



5.4 REGWQ is a recommended test

The **mutoss** package, provides an implementaiton of the Ryan / Einot and Gabriel / Welch test procedure (REGWQ) It is recognized as an acceptable improvement on the logic of the Newman Keuls procedure, a method which is not recommended. REGWQ typically requires equal N, but the package author has created a modification that permits unequal N using the Welch t test logic.

REGWQ is typically described as an approach using the studentized range statistic (as does Tukey HSD and Neuman Keuls), but the r-value for the q statistic is the average of the Tukey r and the N-K r. It is thought that it provides more power than N-K, while adeuately controlling Type I error inflation (that N-K does not).

```
#require(mutoss)
#note: mutoss requires three packages not available on CRAN.  Obtain them from
# bioconductor. These packages are
# Biobase
# BioGenerics
# multtest
# to install, see
```

```
# http://www.bioconductor.org/packages/release/bioc/html/Biobase.html
# http://www.bioconductor.org/packages/release/bioc/html/BiocGenerics.html
# http://www.bioconductor.org/packages/release/bioc/html/multtest.html
regwq1 <- regwq(hays$dv~hays$factora, alpha=.05, data=hays)
```

```
## #----REGWQ - Ryan / Einot and Gabriel / Welsch  test procedure
##
## Number of hyp.:  3
## Number of rej.:  2
##   rejected pValues adjPValues
## 1        3  0.0091     0.0091
## 2        1  0.0117     0.0117
```

```
regwq1
```

```
## $adjPValues
## [1] "0.0117" "0.7655" "0.0091"
##
## $rejected
## [1]  TRUE FALSE  TRUE
##
## $statistic
## [1] 4.402 0.426 3.976
##
## $confIntervals
##              [,1] [,2] [,3]
## control-fast  6.2   NA   NA
## slow-fast     0.6   NA   NA
## control-slow  5.6   NA   NA
##
## $errorControl
## An object of class "ErrorControl"
## Slot "type":
## [1] "FWER"
##
## Slot "alpha":
## [1] 0.05
##
## Slot "k":
## numeric(0)
##
## Slot "q":
## numeric(0)
```

5.5 The Games-Howell Modification of the Tukey Test

A commonly used post hoc test by psychology researchers is the games-howell modification of the tukey test for situations when unequal samples sizes and heterogeneity of variance are present.

5.5.1 The `oneway` function from userfriendlyscience has capabilities for post hoc tests.

The `oneway` function will provide the standard set of `p.adjust` capabilities for alpha rate adjustment (e.g., "holm", "fdr", "bonferroni", etc). It will also implement the Tukey test and the Games-Howell modification of the Tukey test.

```
oneway(y=hays$dv, x=hays$factora, posthoc=c("tukey"))
```

```
## ### Oneway Anova for y=dv and x=factora (groups: control, fast, slow)
##
## Omega squared: 95% CI = [.03; .5], point estimate = .25
## Eta Squared: 95% CI = [.06; .46], point estimate = .3
##
##                                      SS Df     MS    F    p
## Between groups (error + effect) 233.87  2 116.93 5.89 .008
## Within groups (error only)       535.6 27  19.84
##
##
## ### Post hoc test: tukey
##
##              diff lwr    upr   p adj
## fast-control -6.2 -11.14 -1.26 .012
## slow-control -5.6 -10.54 -0.66 .024
## slow-fast    0.6  -4.34  5.54  .951
```

```
oneway(y=hays$dv, x=hays$factora, posthoc=c("games-howell"))
```

```
## ### Oneway Anova for y=dv and x=factora (groups: control, fast, slow)
##
## Omega squared: 95% CI = [.03; .5], point estimate = .25
## Eta Squared: 95% CI = [.06; .46], point estimate = .3
##
##                                      SS Df     MS    F    p
## Between groups (error + effect) 233.87  2 116.93 5.89 .008
## Within groups (error only)       535.6 27  19.84
##
##
## ### Post hoc test: games-howell
##
```

```
##              diff  ci.lo ci.hi    t    df    p
## fast-control -6.2 -12.08 -0.32 2.69 17.99 .038
## slow-control -5.6 -10.35 -0.85 3.11 13.17 .021
## slow-fast     0.6  -4.23  5.43 0.33 13.03 .943
```

5.5.2 A second way to implement the Games-Howell test.

I have found a script that does the test; I cannot vouch for the accuracy of this code - still testing it, but it does match what was produced by the `oneway` function above.

```
# http://aoki2.si.gunma-u.ac.jp/R/src/tukey.R
source("tukey_gh.R")
tukeygh(data=hays$dv,group=hays$factora,method="Games-Howell")
```

```
## $result1
##         n Mean Variance
## Group1 10 27.4 26.04444
## Group2 10 21.2 27.06667
## Group3 10 21.8  6.40000
##
## $Games.Howell
##              t       df          p
## 1:2 2.6902894 17.99333 0.03791342
## 1:3 3.1089795 13.17132 0.02087408
## 2:3 0.3279782 13.03080 0.94268531
```

Aaron Schliegel has independently offered another function to perfom the GH test, but I have not yet examined it and compared to the ones I use here. [https://rpubs.com/aaronsc32/games-howell-test]

5.6 Using the 'pairwise.t.test' function for MC tests

This section descibes the very helpful 'pairwise.t.test' function. It has similarities to the logic we saw above with 'p.adjust'. It has two major attracive features: 1. It permits use of many of the error-rate control methods in the "bonferroni" family that we have seen recommended. 2. It permits use of either the pooled within-group error term (when the HOV assumption is satisfied) or a Welch approach to the t statistic when heterogeneity of variance is present. This is the 'pool.sd' argument.

I list code for choosing may of the flavors of correction, but only show output for two.

```
# first, just do pairwise comparisons with bonferroni corrections
# assumes having done a set of three "contrasts".  should match results seen
```

```r
# above in the BonferroniCI function from asbio.
pairwise.t.test(hays$dv,hays$factora,pool.sd=TRUE,p.adj="bonf")
```

```
##
##  Pairwise comparisons using t tests with pooled SD
##
## data:  hays$dv and hays$factora
##
##      control fast
## fast 0.013   -
## slow 0.027   1.000
##
## P value adjustment method: bonferroni
```

```r
# We can change the approach to the Holm, Hochberg, Hommel,
# Benjamini and Hochberg, Benjamini&Yekutieli, and fdr corrections,
# as well as "none" which will give the same thing as the LSD test.
# Choice of these depends on several factors, including whether
# the contrasts examined are independent (and they are not since they
# are all of the pairwise comparisons,
# Of these modified bonferroni type of approaches, only the "BY"
# approach is probably the most appropriate here, since some of our
# comparisons are correlated and BY permits that correlation to be either
# positive or negative
#pairwise.t.test(dv,factora,pool.sd=TRUE,p.adj="holm")
#pairwise.t.test(dv,factora,pool.sd=TRUE,p.adj="hochberg")
#pairwise.t.test(dv,factora,pool.sd=TRUE,p.adj="hommel")
#pairwise.t.test(dv,factora,pool.sd=TRUE,p.adj="BH")
#pairwise.t.test(dv,factora,pool.sd=TRUE,p.adj="BY")
pairwise.t.test(hays$dv,hays$factora,pool.sd=TRUE,p.adj="fdr")
```

```
##
##  Pairwise comparisons using t tests with pooled SD
##
## data:  hays$dv and hays$factora
##
##      control fast
## fast 0.013   -
## slow 0.014   0.766
##
## P value adjustment method: fdr
```

```r
#pairwise.t.test(dv,factora,pool.sd=TRUE,p.adj="none")
```

Now demonstrate two of these tests again, but use 'pool.sd=FALSE'.

```
# note that setting pool.sd to FALSE changes the outcome since
# it employs a Welch or Fisher-Behrens type of approach
pairwise.t.test(hays$dv,hays$factora,pool.sd=FALSE,p.adj="bonf")
```

```
##
##  Pairwise comparisons using t tests with non-pooled SD
##
## data:  hays$dv and hays$factora
##
##      control fast
## fast 0.045   -
## slow 0.025   1.000
##
## P value adjustment method: bonferroni
```

```
# or
pairwise.t.test(hays$dv,hays$factora,pool.sd=FALSE,p.adj="BY")
```

```
##
##  Pairwise comparisons using t tests with non-pooled SD
##
## data:  hays$dv and hays$factora
##
##      control fast
## fast 0.041   -
## slow 0.041   1.000
##
## P value adjustment method: BY
```

It is interesting that contol vs fast is found to be significant here for the bonferroni test (p=.045). So simply using the non-pooled error is not sufficient to produce a NS test as was the case above for 'DTK'. 'DTK' is more conservative because of the Tukey family derivation instead of the bonferroni family derivation.

5.7 The Neuman-keuls test

Psychology researchers had commonly used the Neuman-Keuls test for many decades, since it afforded a power improvement over the Tukey HSD test. Simulation work has now found that it underperforms in the core goal of controlling Type I error rate inflation. I cannot recommend it so I do not do an example here. However, if once is forced to use it (by misinformed advisors or collaboraters), it is possible to find R functions to do it.

One possibility is the 'SNK.test' function in the **agricolae** package.

5.8 The glht function for post hoc tests and contrasts

The **multcomp** package has some capabilities for multiple comparisons that are useful for a variety of model objects, including `aov` and `lm`. If we apply `glht` here, requesting Tukey adjustments for pairwise comparisons, we find that it matches the output from the `TukeyHSD` function above (at least to three decimals). We use the fit.3lm object first produced with the `lm` function.

```
#library(multcomp)
glht.tukey <-glht(fit.3lm, linfct = mcp(factora="Tukey"))
summary(glht.tukey)
```

```
##
##   Simultaneous Tests for General Linear Hypotheses
##
## Multiple Comparisons of Means: Tukey Contrasts
##
##
## Fit: lm(formula = dv ~ factora, data = hays)
##
## Linear Hypotheses:
##                       Estimate Std. Error t value Pr(>|t|)
## fast - control == 0   -6.200      1.992  -3.113   0.0117 *
## slow - control == 0   -5.600      1.992  -2.811   0.0239 *
## slow - fast == 0       0.600      1.992   0.301   0.9513
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
## (Adjusted p values reported -- single-step method)
```

```
confint(glht.tukey)
```

```
##
##   Simultaneous Confidence Intervals
##
## Multiple Comparisons of Means: Tukey Contrasts
##
##
## Fit: lm(formula = dv ~ factora, data = hays)
##
## Quantile = 2.48
## 95% family-wise confidence level
##
##
## Linear Hypotheses:
##                       Estimate lwr      upr
## fast - control == 0  -6.2000 -11.1398  -1.2602
```

```
## slow - control == 0  -5.6000 -10.5398  -0.6602
## slow - fast == 0      0.6000  -4.3398   5.5398
```

The same `glht` function can also handle contrasts and they don't have to be orthogonal. For example,

```
contr <- rbind("Ctl-Fast" = c(1, -1, 0),
               "Ctl-SLow" = c(1, 0, -1),
               "Fast-Slow"= c(0,1,-1))
glht.contr1 <-glht(fit.3lm,
                   linfct = mcp(factora=contr),
                   test = adjusted())
summary(glht.contr1)
```

```
##
##   Simultaneous Tests for General Linear Hypotheses
##
## Multiple Comparisons of Means: User-defined Contrasts
##
##
## Fit: lm(formula = dv ~ factora, data = hays)
##
## Linear Hypotheses:
##                Estimate Std. Error t value Pr(>|t|)
## Ctl-Fast == 0     6.200      1.992   3.113   0.0117 *
## Ctl-SLow == 0     5.600      1.992   2.811   0.0238 *
## Fast-Slow == 0   -0.600      1.992  -0.301   0.9513
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
## (Adjusted p values reported -- single-step method)
```

```
confint(glht.contr1)
```

```
##
##   Simultaneous Confidence Intervals
##
## Multiple Comparisons of Means: User-defined Contrasts
##
##
## Fit: lm(formula = dv ~ factora, data = hays)
##
## Quantile = 2.479
## 95% family-wise confidence level
##
##
## Linear Hypotheses:
##                Estimate lwr     upr
```

```
## Ctl-Fast == 0   6.2000   1.2622 11.1378
## Ctl-SLow == 0   5.6000   0.6622 10.5378
## Fast-Slow == 0 -0.6000  -5.5378  4.3378
```

Chapter 6

Beginning to Explore the emmeans package for post hoc tests and contrasts

The **emmeans** package is one of several alternatives to facilitate post hoc methods application and contrast analysis. It is a reletively recent replacement for the **lsmeans** that some R users may be familiar with. It is intended for use with a wide variety of ANOVA models, including repeated measures and nested designs where the initial modeling would employ 'aov', 'lm' 'ez' or 'lme4' (mixed models).

6.1 Using emmeans for pairwise post hoc multiple comparisons.

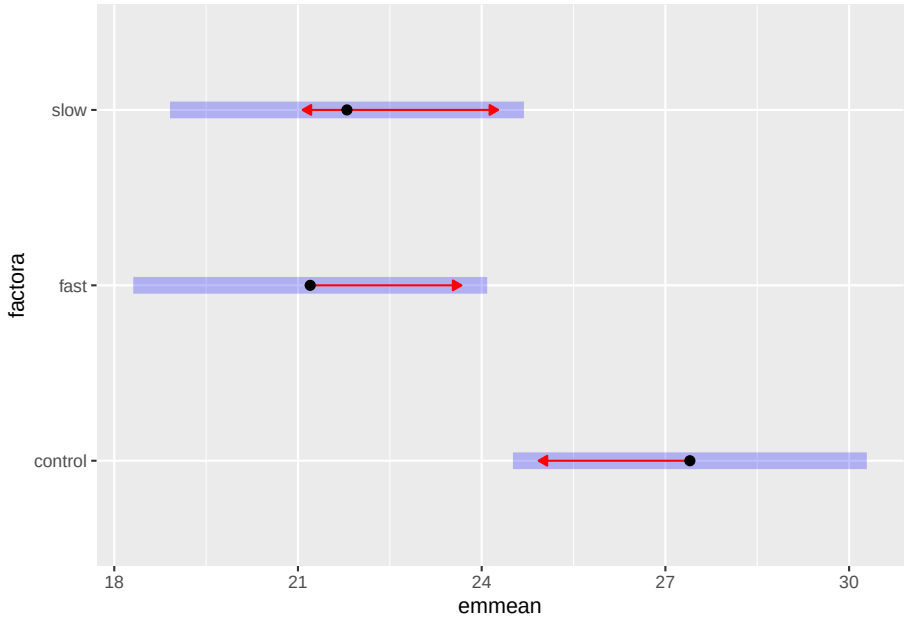
A minimal illustration is presented here. First is a "pairwise" approach to followup comparisons, with a p-value adjustment equivalent to the Tukey test. The `emmeans` function requires a model object to be passed as the first argument. We could use either fit1 (the aov object) or fit2 (the lm object) originally created in the base Anova section of this document.

```
#library(emmeans)
# reminder: fit.2 <- lm(dv~factora, data=hays)
fit2.emm.a <- emmeans(fit.2, "factora", data=hays)
pairs(fit2.emm.a, adjust="tukey")
```

```
##  contrast       estimate   SE df t.ratio p.value
##  control - fast      6.2 1.99 27  3.113  0.0117
##  control - slow      5.6 1.99 27  2.811  0.0239
```

```
##  fast - slow        -0.6 1.99 27 -0.301  0.9513
##
## P value adjustment: tukey method for comparing a family of 3 estimates
```

```
plot(fit2.emm.a, comparisons = TRUE)
```




```
#pairs(fit2.emm.a, adjust="none")
```

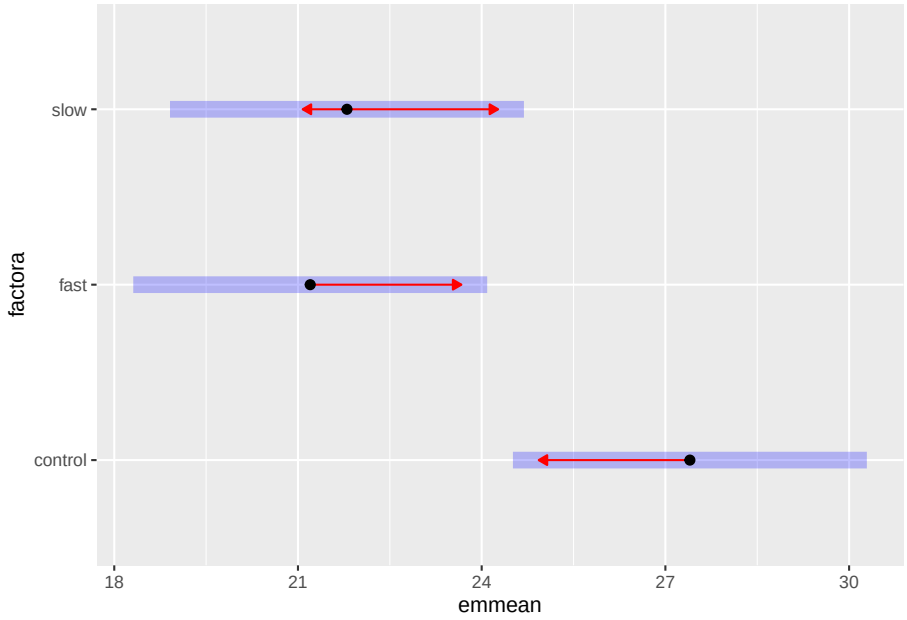
The blue bars on the plot are the confidence intervals. The red arrowed lines represent a scheme to determine homogeneous groups. If the red lines overlap for two groups, then they are not signficantly different using the method chosen.

The 'adjust' argument can take one of several useful methods. 'tukey' is default, but others including 'sidak', 'bonferroni', etc can be specified. Specifying 'none' produces unadjusted p-values. See help with '?emmeans::summary.emmGrid' for details. Here is an example using the 'holm' method of adjustment.

```
library(emmeans)
fit2.emm.b <- emmeans(fit.2, "factora", data=hays)
pairs(fit2.emm.b, adjust="holm")
```

```
##  contrast       estimate   SE df t.ratio p.value
##  control - fast      6.2 1.99 27  3.113  0.0131
##  control - slow      5.6 1.99 27  2.811  0.0181
##  fast - slow        -0.6 1.99 27 -0.301  0.7655
##
## P value adjustment: holm method for 3 tests
```

```
plot(fit2.emm.b, comparisons = TRUE)
```




```
#pairs(fit2.emm.a, adjust="none")
```

6.2 Analytical Contrasts

Next, we will create linear contrasts and test them. Notice that in "testing" the contrast, no alpha-rate control adustments are made. This produces t-values that are the square root of the F's we found above for the 'split' approach on 'aov' or the regression coefficient t values from 'lm' objects with 'summary'. It is also possible to obtain confidence intervals on the contrasts, and I show how an adjustment can be done (but it wouldn't make sense to adjust on the CIs as I did here and not on the tests; both should be the same).

Interpreting the CIs is problematic. They are in the scale of 2, -1, -1 and 0, 1, -1 (thus the 11.8 and -.6 psi values we have seen several times previously for this data set. It is all well and good if the only thing we are using the CIs for is to evaluate whether they overlap zero (as a proxy for the hypothesis test). But the actual range of values is arbitrarily dependent on the values of the Coefficients (thus their variance). One strategy might be to implement what we saw in SPSS UNIANOVA. We could constrain the largest coefficient value to be a "1", and use fractions for the remainder, when necessary.

So a redo of the earlier approach to contrasts above could look like this using `test` function on the contrasts fit to the emm model grid of means:

```
lincombs <- contrast(fit2.emm.a,
               list(ac1=c(1,-.5,-.5), ac2=c(0,1,-1))) # second one not changed
test(lincombs, adjust="none")
```

```
##  contrast estimate   SE df t.ratio p.value
##  ac1           5.9 1.72 27  3.420  0.0020
##  ac2          -0.6 1.99 27 -0.301  0.7655
```

```
confint(lincombs, adjust="sidak")
```

```
##  contrast estimate   SE df lower.CL upper.CL
##  ac1           5.9 1.72 27     1.82     9.98
##  ac2          -0.6 1.99 27    -5.32     4.12
##
## Confidence level used: 0.95
## Conf-level adjustment: sidak method for 2 estimates
```

Now the psi value for the contrasts are directly related to how we speak about what the contrasts are evaluating. The mean of "control" is 5.9 units above the average of the "fast" and "slow" conditions. And for the second contrast, "fast" is .6 units below the mean of "slow". In this scaling, the CIs are more diretly interpretable at their edges. But make sure to note that the t values and p values do not change with this scaling change.

6.3 Concluding comments on emmeans

The **emmeans** package is a very powerful tool. But it is almost overkill for a one-way design. Its utility will become impressive for factorial between-groups designs, for repeated measures designs, and for linear mixed effect models. The goal is to revisit it with the first two of those three applications.

Chapter 7

Assumptions: Evaluation and Methods for handling their violation

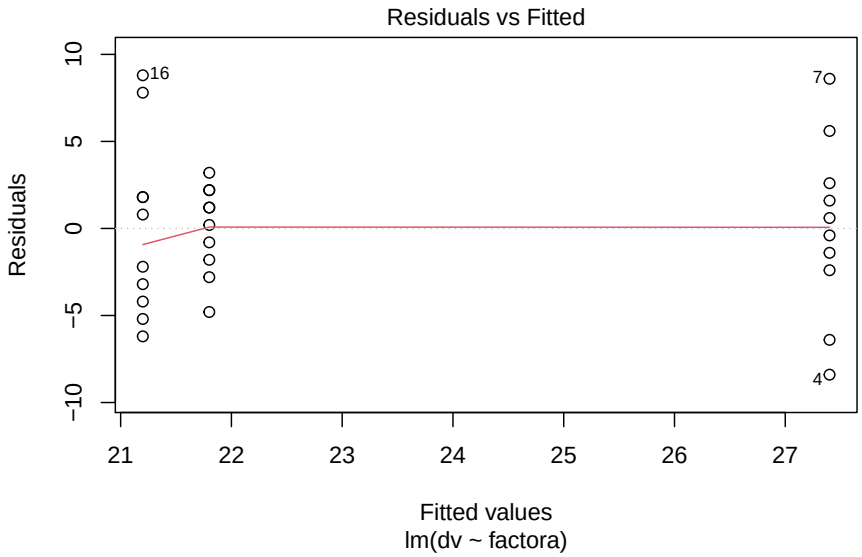
This section reviews both graphical and inferential methods for evaluating the assumptions for the 1-way ANOVA omnibus F test, and the use of the pooled within-group error for contrasts and multiple comparisions

7.1 Graphical evaluation of Residuals

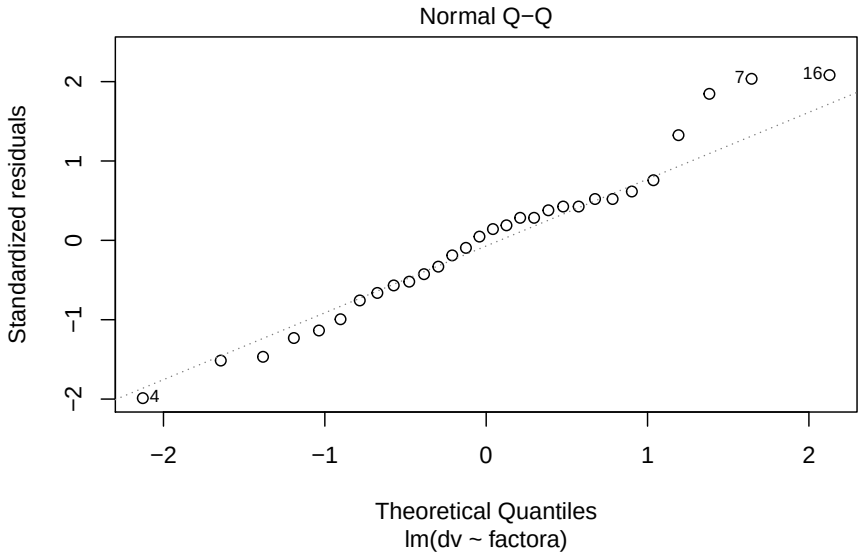
In order to provide graphical displays for residuals analysis, R provides a very simple approach when working with linear models. All that is required is to request a plot of the model and this provides the relevant residuals plots. All four fits from above yield the same residuals, so lets just look at fit.2 since it used 'lm'. This is the same approach we have covered for regresion models in R.

The first, commented, code line ('plot(fit.2)') produces all four plots and the user is required to click or hit the enter key to advance through the set. Instead, I request each individually, but only display the two we are most familiar with since they are the relevant ones here.

```
#win.graph()
#plot(fit.2)
# now produce the same 4 graphs in their own windows
plot(fit.2,which=1) #residuals vs yhats
```



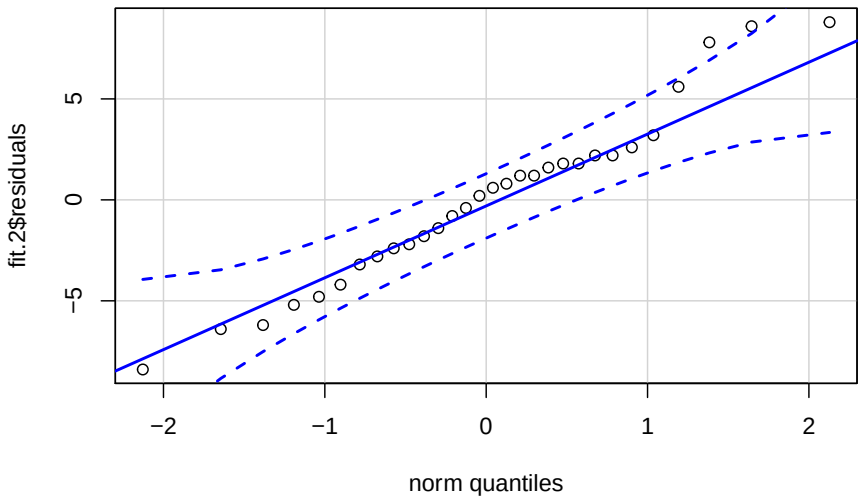

```
plot(fit.2,which=2) #normal probability plot
```




```
#plot(fit.2,which=3)
#plot(fit.2,which=5)
```

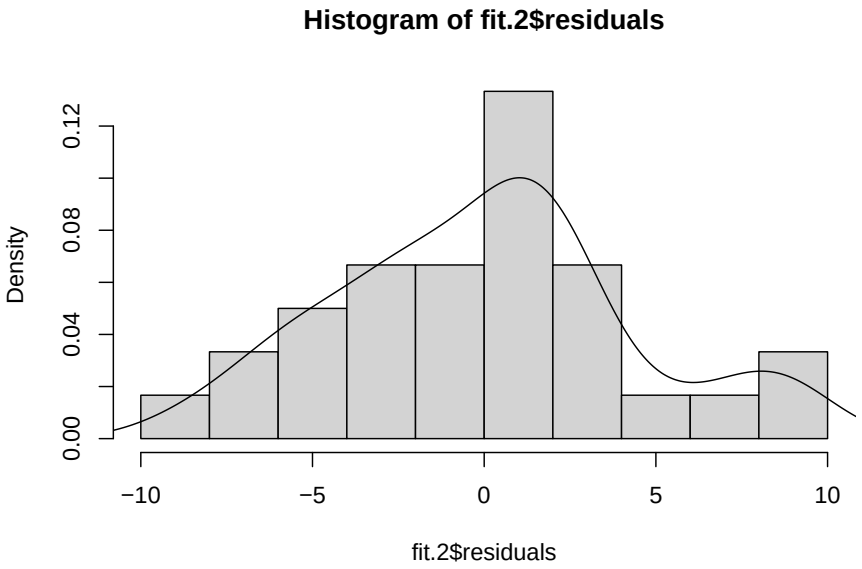
A better qq plot is available in the **car** package. It adds a confidence envelope to the qqnormal plot in order to better gauge, visually, how well the variable fits a normal distribution. Here, the residuals are extracted from the fit.2 `lm` object and passed to the `qqPlot` function.

```
car::qqPlot(fit.2$residuals, distribution="norm", id=FALSE)
```



These standard plots produced by the base system 'plot' function do not include a frequency histogram of the residuals. It is useful, and can be done quickly by extracting the residuals from the 'lm' fit object and passing them directly to the 'histogram' function. In this illustration the histogram is not embellished with any additional text or style changes from defaults. The histogram would be more useful in larger N data sets than for this small example. In this example the deviation from normality is no too bad, especially for a small sample size study.

```
hist(fit.2$residuals, breaks=8, prob=TRUE)
lines(density(fit.2$residuals))
```



7.2 Inferences about the Normality Assumption

R provides several approaches to evaluation of the normality of a variable. The most well-known test is the Shapiro-Wilk test. That test is shown here along with the Anderson-Darling test which is (by my read) a preferred approach. Other tests are also available, and the code is shown, although commented out and not executed here. Note an alternative way of passing the 'lm' fit object residuals to a function rather than the 'fit.2$residuals' approach taken for the freqency histogram above. These tests do not permit rejection of the null hypothesis of residual normality, an outcome that is consistent with the visualizations done above.

```
# Perform the Shapiro-Wilk test for normality on the residuals
shapiro.test(residuals(fit.2))
```

```
##
##  Shapiro-Wilk normality test
##
## data:  residuals(fit.2)
## W = 0.97096, p-value = 0.5656
```

```
# use tests found in the nortest package, as we reviewed for multiple regression
#library(nortest)
ad.test(residuals(fit.2)) #get Anderson-Darling test for normality (nortest package must be
```

```
##
```

```
##  Anderson-Darling normality test
##
## data:  residuals(fit.2)
## A = 0.30011, p-value = 0.5599
```

```
#cvm.test(residuals(fit.2)) #get Cramer-von Mises test for normality (nortest package must
#lillie.test(residuals(fit.2)) #get Lilliefors (Kolmogorov-Smirnov) test for normality (nor
#pearson.test(residuals(fit.2)) #get Pearson chi-square test for normality (nortest package
#sf.test(residuals(fit.2)) #get Shapiro-Francia test for normality (nortest package must be
```

7.3 Inferential tests regarding the homogeneity of Variance Assumption

Several tests are available in R for the ANOVA-related homogeneity of variance assumption. The reader will recall that we saw other tests of homoscedasticity for regression models in R code illustrations for those models. They are also appropriate here, but are not repeated in order to save space. The most common tests are those in the "Levene" family, but others are shown here as well.

The first test is The Bartlett Test from stats package (stats is part of the base install). This test is not identical to the Bartlett-Box test produced by SPSS MANOVA, but it is close. The Bartlett-Box F-test reported in MANOVA is an adaptation to the univariate case of the Box's M test for multivariate data. The Bartlett test in R is based on an original paper by Bartlett (1937?) and should probably converge very closely on the Bartlett-Box F approximation. The Bartlett test statistic is a chi-squared approximation. This test is sensitive to departures from the normality in the DV.

```
bartlett.test(dv~factora,data=hays)
```

```
##
##  Bartlett test of homogeneity of variances
##
## data:  dv by factora
## Bartlett's K-squared = 4.702, df = 2, p-value = 0.09527
```

The Cochran's C test for outlying variances is implemented in the **outliers** package. The function 'cochran.test' requires the model specification and the data frame for execution. The result appears to match the test result produced SPSS MANOVA for this same data set, but df are different. I have yet to explore the df question for each of these implementations. MANOVA reports 9,3 df where 'cochran.test' reports 10,3 df. The "sample estimates" are the variances of the individual cells.

```
cochran.test(dv~factora, hays)
```

```
##
```

```
##  Cochran test for outlying variance
##
## data:  dv ~ factora
## C = 0.45482, df = 10, k = 3, p-value = 0.5093
## alternative hypothesis: Group fast has outlying variance
## sample estimates:
##  control     fast     slow
## 26.04444 27.06667  6.40000
```

We can find the Levene Test in the **car** package this is actually the Brown-Forsythe modification of the Levene test to be median-centered rather than mean-centered. We have seen that what some SPSS procedures call the Levene test is actually the original Levene test that uses mean-centering and is sensitive to the normality assumption. Here, the 'leveneTest' function has a default of median centering (the desireable approach)

```
#require(car)
leveneTest(dv~factora,data=hays)
```

```
## Levene's Test for Homogeneity of Variance (center = median)
##       Df F value Pr(>F)
## group  2  2.0076 0.1539
##       27
```

The levene test can also be obtained from the **lawstat** package. The lawtstat version is more flexible, permitting mean centering,median centering, or trimmed-mean centering as well as bootstrapping.

First, let's use the mean-centering approach and demonstrate that this is what SPSS uses in one-way and GLM (older versions of GLM)

```
#require(lawstat)
levene.test(hays$dv,hays$factora,location="mean")
```

```
##
##  Classical Levene's test based on the absolute deviations from the mean
##  ( none not applied because the location is not set to median )
##
## data:  hays$dv
## Test Statistic = 2.0588, p-value = 0.1472
```

Next is the median centering aproach to match leveneTest from **car**, and to provide the Brown-Forsythe method that was the modification of Levene to be more robust against non-normality.

```
levene.test(hays$dv,hays$factora,location="median")
```

```
##
##  Modified robust Brown-Forsythe Levene-type test based on the absolute
##  deviations from the median
```

```
##
## data:  hays$dv
## Test Statistic = 2.0076, p-value = 0.1539
```

A robust method used trimmed-mean centering; 20% of the scores are trimmed here.

```
levene.test(hays$dv,hays$factora,location="trim.mean",trim.alpha=.2)
```

```
##
##  Modified robust Levene-type test based on the absolute deviations from
##  the trimmed mean ( none not applied because the location is not set to
##  median )
##
## data:  hays$dv
## Test Statistic = 2.0179, p-value = 0.1525
```

This next approach returns to median centering, but implements the Obrien test which uses a correction factor described by Obrien (1978) and removal of zeros by Hines and Hines (2000). This is supposed to be a useful improvement on the basic Levene method and can be recommended.

```
levene.test(hays$dv,hays$factora,location="median", correction.method="zero.correction")
```

```
##
##  Modified robust Brown-Forsythe Levene-type test based on the absolute
##  deviations from the median with modified structural zero removal
##  method and correction factor
##
## data:  hays$dv
## Test Statistic = 2.4397, p-value = 0.1085
```

The 'levene.test' function from **lawstat** also permits a bootstrapping approach to obtaining a test of the HOV asumption using the basic Levene median centering approach. This is recommended if marked non-normality of the within-cell distributions exists.

```
#     uses the median-centered approach.  from Lim and Loh, 1996
levene.test(hays$dv,hays$factora,location="median",bootstrap=TRUE)
```

```
##
##  bootstrap Modified robust Brown-Forsythe Levene-type test based on the
##  absolute deviations from the median
##
## data:  hays$dv
## Test Statistic = 2.0076, p-value = 0.117
```

The Fligner-Killeen test is a method that produces a chi-squared test statistic for a homogeneity of variance test. The Fligner-Killen method is a non-parametric approach to the median-centering absolute value method. I've not read any

literature comparing it's efficacy to the suite of Levene-related methods. The user should do some homework on this method before employing it.

```
fligner.test(hays$dv~hays$factora,data=hays)
```

```
##
##  Fligner-Killeen test of homogeneity of variances
##
## data:  hays$dv by hays$factora
## Fligner-Killeen:med chi-squared = 4.0696, df = 2, p-value = 0.1307
```

Finally, as an instructional tool, we can build our own function to do the median-centered Levene Test. This gives the reader the details on what the test actually does. In addition, it exposes the reader to the 'tapply' function which is a very useful member of the "apply" family of functions. It permits us to do an operation (subtracting their group median from individual DV scores). These new absolute value deviations are then simply analyzed with a one-way anova (using 'lm' here). Note that the p-value matches what we obtained above for both 'leveneTest' and 'levene.test'.

```
# Here,I build the Brown-Forsythe modification of the Levene test from scratch:
#  This should give you some insight into a bit of "The R Way"
bf.levene <- function(y, group)
   {
   group <- as.factor(group) # just to make certain the IV is a "factor"
   medians <- tapply(y, group, median)
   deviation <- abs(y - medians[group])
   anova(lm(deviation ~ group))
   }
bf.levene(hays$dv,hays$factora)
```

```
## Analysis of Variance Table
##
## Response: deviation
##           Df  Sum Sq Mean Sq F value Pr(>F)
## group      2  27.467 13.7333  2.0076 0.1539
## Residuals 27 184.700  6.8407
```

7.4 A plot of cell means vs variances

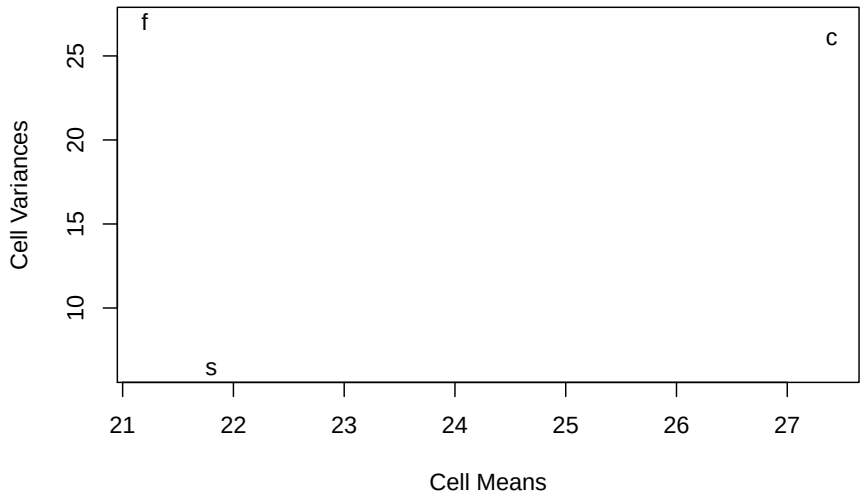
A caveat at the outset: this type of plot is not terribly useful for studies with only three groups. But with larger factorial anovas, it might be helpful.

We discussed this plot previously. Recall that this type of plot helps evaluate whether heterogeneity of variance might have arisen from a simple scaling issue. If so, then scale transformations may help. E.g,, a postive mean-variance correlation reflects a situation where a log transformation or a fractional exponent transformation of the DV might produce homoscedasticity. I'm also using the

`tapply` function here in ways that we have not covered. `tapply` is an important function in dealing with factors.

The issue with this particular design/plot is that a scatterplot with only three points cannot possibly be relied upon to reveal a pattern. Nonetheless, it puts in place a technique for future use in other design.

```
plot(tapply(hays$dv, hays$factora, mean), tapply(hays$dv, hays$factora, var), xlab = "Cell M
             ylab = "Cell Variances", pch = levels(hays$factora))
```



7.5 What to do when assumptions are violated

Useful strategies are available for addressing inferential questions from a 1-factor ANOVA design when assumptions are violated. Their number is very large. Some will be listed here. What is important for the student is to realize that alternatives to both the omnibus F-test and followup questions (e.g. contrasts, post hoc and other multiple comparison methods) are available.

Alternatives to the standard Omnibus F-test:

- Welch F test when Heterogeneity of Variance is present. This test was illustrated above with the 'oneway.test' funciton.
- A Brown-Forsythe modification of the F test is analogous to the Welch F test and is found in output from the `oneway` function earlier in this document.
- Bootstrapping or permutation tests when non-normality is present A per-

mutation test was seen above with the 'exAnova' function. Bootstrapping is also availble in R and can be applied to ANOVA models. These Resampling methods are covered briefly in a later chapter
- Robust methods based on robust central tendency estimators (see the **WRS2** package in R and much work by Rand Wilcox - and a later chapter in this document)
- Nonparametric methods are often used when DV distributions are divergent from normality. One of those is covered below, the Kruskall-Wallis test.

Alternatives to Post Hoc and Multiple Comparison Tests:

- Several MC tests are explicitly designed to cope with distributional assumption issues in ANOVA data sets. These are discussed below in the Post Hoc/MC section
- One very commonly used approach in the psychological sciences is the "Games-Howell" modification of the Tukey test for situations with Heterogeneity of Variance and Unequal N. Other more recent approaches may be an improvement on the G-H method, but have not received as much usage since they had only recently become available in software.
- The DTK test (Duncan/Tukey/Kramer) is another modification of the Tukey HSD test for applications when there is unequal N and heterogeneity of variance. See the Post Hoc / Multiple Comparisons section below.
- The Kruskall-Wallis test can also be followed up with nonparametric approaches to pairwise multiple comparisons (see below)

Alternatives for Analytical Contrasts when assumptions are violated:

- Most commonly, when heterogeneity of variance is present, the approach to follow up analysis involves one of the pairwise multiple comparison methods designed to cope with that heterogeneity (see below).
- However, since any analytical contrast is viewed as a linear combination, standard errors can be derived in the same manner as the Welch omnibus test is designed. I am still looking for a simple implementation of this in R with regard to ANOVA contrasts. The **emmeans** package provides one possible solution and was introduced above I am still sorting out some confusions about use of **emmeans** for these purposes.

Chapter 8

Effect Sizes, Power, and Sample Size Planning

8.1 Effect Sizes

The multipel R-squared available from the 'lm' fit objects is also called eta-squared. The **afex** package functions give the "generalized eta squared" statistic.

Note that in a 1-way design, partial eta squared values will equal the eta squared value, and partial omega squared values will equal omega squared values.

One useful function is provided here for computation of several commonly used effect sizes (**sjstats**).

Caveats:
1. I am not yet certain how well the sjstats package functions work for larger designs with unequal N. They should be ok, but it is tricky to compute something like omega squared when there is unequal N. 2. Still working on functions to produce effect sizes for analytical contrasts.

The `anova_stats` function strikes me as very useful. It not only returns the basic ANOVA, but provides several effect size indices, including Cohen's f. Note that the "power" value is a post-hoc power calculation that we have reviewed as a problematic concept in how it has often been applied. Refer to the stattoolkit bibliography for the literature on this.

```
# require(sjstats)
kable(anova_stats(fit.1))
```

	term	df	sumsq	meansq	statistic	p.value	etasq	partial.etasq	omegasq	partial
factora	factora	2	233.867	116.933	5.895	0.008	0.304	0.304	0.246	
...2	Residuals	27	535.600	19.837	NA	NA	NA	NA	NA	

The `anova_stats` function can work on either an `anova` object or one of class `Anova`. For larger factorial designs, this would be an important way of obtaining effect sizes based on differing SS Types that can be specified. In the above section, we passed the already-created `anova` fit object to `anova_stats`, but we can also do the whole analysis in one line of code using the **car** package `Anova` function to specify SS Type. The reader should use caution with this method. The effect size statistics are different values when Type III SS even for this balanced N model. I suspect that anova_stats may have a bug. Nonetheless, here is a template of how that might work:

```
contrasts(hays$factora) <- contr.sum
kable(anova_stats(Anova(aov(dv~factora, data=hays), type=2)))
```

	term	sumsq	meansq	df	statistic	p.value	etasq	partial.etasq	omegasq	partial
factora	factora	233.867	116.933	2	5.895	0.008	0.304	0.304	0.246	
...2	Residuals	535.600	19.837	27	NA	NA	NA	NA	NA	

Two functions, `eta_sq` and `omega_sq` permit confidence interval calculation for either full or partial effect size statistics. I have found some odd results for the CI calculations when requesting the partial effect size statistics here, so caution is urged. The partial effect size statistic concept really has no meaning in 1way anovas anyway.

```
omega_sq(fit.1,   ci.lvl = .95)
```

```
##      term omegasq conf.low conf.high
## 1 factora   0.246   -0.038     0.481
```

```
eta_sq(fit.1, ci.lvl = .95)
```

```
##      term etasq conf.low conf.high
## 1 factora 0.304    0.032     0.525
```

Another option for effect size calculation is the `etaSquared` function from the **lsr** package. It can take an argument that permits specification of SS Type, but that will not matter in this balanced design.

```
library(lsr)
#etaSquared(fit.1, type=1)
#etaSquared(fit.1, type=2)
etaSquared(fit.1, type=3)
```

```
##            eta.sq eta.sq.part
## factora 0.3039335   0.3039335
```

8.2 Power and sample size planning for completely randomized 1-factor ANOVA designs

The **pwr** package provides a fairly comprehensive way to estimate sample size requirements when designing studies. For a one-factor design, the logic of the code is very straight forward. In other work, we have seen how to use GPower as well.

To use the `pwr.anova.test` funtion:

- We need to tell it how many groups.
- We need to have an estimate of the within-group std deviation (assumes homogeneity of variance)
- We need to have a set of expected outcome values for the sample means. With these means and the within-group variation (the sd), we can estimate cohen's effect size statistic (the "f")

Since we have been working with a 3-group design, lets see how we might have planned for that with the 'pwr.anova.test' function. I set the means and sd arbitrarily in this example - they would normally be chosen on the basis of informed prior information, perhaps from pilot studies or published literature.

It is instructive to "fiddle" with this code, changing the means and the sd to see how it affects the desired n per group. This is a quick alternative to GPower.

```
#library(pwr)
groups = 3
means = c(25,20,20)
sd = 5
grand.mean  = mean(means)
efsize = sqrt( sum( (1/groups) * (means-grand.mean)^2) ) /sd #cohen's "f" effect size
efsize
```

```
## [1] 0.4714045
```

```
pwr.anova.test(k = groups,
               n = NULL,
               f = efsize,
               sig.level = 0.05,
               power = 0.90)
```

```
##
##      Balanced one-way analysis of variance power calculation
##
##               k = 3
##               n = 20.01726
##               f = 0.4714045
##       sig.level = 0.05
##           power = 0.9
```

```
##
## NOTE: n is number in each group
```

This, and other **pwr** functions work by passing all but one of the relevant characteristics to the function. By leaving out sample size, and passing alpha, power, and effect size, the minimal sample size per group required to acheive that power is returned.

Chapter 9

Bayesian Inference for 1-way ANOVAs

There are many ways to approach Bayesian Inference, even for a simple design such as a 1way ANOVA. e.g., https://rpubs.com/dgolicher/jags_one_way_anova

If you intend to do a lot of Bayesian statistics you would find it helpful to learn the BUGS/JAGS language, which can be accessed in R via the R2OpenBUGS or R2WinBUGS packages.

But a different Bayesian approach has gained much currency in recent years, especially in the behavioral and life sciences. This is the Bayes Factor approach and is the one illustrated in this chapter.

9.1 A BayesFactor approach to Oneway ANOVA

This section is not meant to be a comprehensive treatment of Bayesian alternatives to the traditional ANOVA. It provides simple code to extract a bayes factor for the omnibus model and then attempts to demonstrate a way to evauluate contrasts within that model.

Initially, using the **BayesFactor** package of Morey et. al., we can use their default `anovaBF` approach. In order to set up the analysis for contrasts that are done below, the contrast for "factora" is respecified here as the standard orthogonal set used above. This BayesFactor approach is the alternative to the "fit.1" model used above.

The 'anovaBF' default approach is to compare the full model to the intercept-only model. A Jeffries prior is used for in the default approach of the `anovaBF` function for hypothetical population means and variances. Some control over

the characteristics of the prior distribution is possible in `anovaBF`, but that is beyond the scope of this document. Readers are urged to begin with the help pages for the **BayesFactor** package and the `anovaBF` function.

```
#library(BayesFactor)
#set contrasts for factora to the orthogonal set
contrasts.factora <- matrix(c(2,-1,-1, 0,1,-1),ncol=2)
contrasts(hays$factora) <- contrasts.factora
#contrasts(factora)
anovaBF(dv~factora, data=hays)
```

```
## Bayes factor analysis
## --------------
## [1] factora : 6.926757 ±0.01%
##
## Against denominator:
##   Intercept only
## ---
## Bayes factor type: BFlinearModel, JZS
```

This indicates some evidence against the null, rougly 7 times the evidence for the null. It is not a strikingly strong degress of evidence

9.2 Analytical Contrasts and BayesFactor

A direct BF approach to individual contrasts with the **BayesFactor** doesn't appear to be available. An alternative would be to use the 'lmBF' function on the analogous "lm" model. However, 'lmBF' won't work with IVs that are factors, so we will try to trick it. If we manually code the contrasts as new variables in the data frame we can name those as IVs. I've created a modified .csv file that has these new variables (ac1 and ac2). Read the new file and look at the first few and last few lines:

```
hays3 <- read.csv("data/hays3.csv", stringsAsFactors=T)
kable(headTail(hays3))
```

	factora	dv	ac1	ac2
1	control	27	2	0
2	control	28	2	0
3	control	33	2	0
4	control	19	2	0
...	NA	...	...	...
27	slow	22	-1	-1
28	slow	17	-1	-1
29	slow	20	-1	-1
30	slow	23	-1	-1

Now lets run the BF regression with those IVs to find the best model . Because of what we already know about this data set, I fit the full model with both contrasts, and a model that only had ac1 (the contrast that compares the control group to the average of both experimental groups).

The 'lmBF' function allows us to evaluate those models and to compare them. The substantially larger bayes Factor for the ac1-only model indicates it is a better fit. This is because (as we know), the means of the 2nd and 3rd groups are so close so we don't really need that second vector in the model.

```
lmbf.full <- lmBF(dv~ac1+ac2, data=hays3)
lmbf.ac1 <-  lmBF(dv~ac1, data=hays3)
#lmbf.ac2 <-  lmBF(dv~ac2, data=hays3)
#lmbf.full
#lmbf.ac1
ful_vs_ac1 <- c(lmbf.full,lmbf.ac1)
ful_vs_ac1
```

```
## Bayes factor analysis
## --------------
## [1] ac1 + ac2 : 6.959137 ±0%
## [2] ac1       : 20.4168  ±0%
##
## Against denominator:
##   Intercept only
## ---
## Bayes factor type: BFlinearModel, JZS
```

A different function from the same package permits direct comparison of all regression models for a given set of IVs. It gives results that mirror what we saw above.

```
generalTestBF(dv~ac1+ac2, data=hays3, whichModels="all")
```

```
## Bayes factor analysis
## --------------
## [1] ac1       : 20.4168   ±0%
## [2] ac2       : 0.3530796 ±0%
## [3] ac1 + ac2 : 6.959137  ±0%
##
## Against denominator:
##   Intercept only
## ---
## Bayes factor type: BFlinearModel, JZS
```

To use this approach, it is recommended that the student explore a good bit more background on use of Bayes Factors approaches. The detailed treatment of this method is beyond the bounds of the 511 class.

One can also examine considerable online help from Morey and others on the **BayesFactor** package:

[http://bayesfactor.blogspot.com/2015/01/multiple-comparisons-with-bayesfactor-1.html]

Chapter 10

Robust Methods and Resampling Methods

Violations of assumptions can wreak havoc with the standard methods of ANOVA, leading to increases in Type I error rate. Wilcox (2016) has reviewed the literature on this and concludes that researchers have been systematically taught, in error, that the standard parametric methods are often resistant to violations of assumptions. He argues, in a compelling fashion, that the standard approach of evaluating assumptions with inferential tests (e.g., levene test for homogeneity of variance, and various tests for non-normality) are underpowerd for sample sizes that are common in much research. Reliance on the "non-significance" of such tests leads to flawed application of the standard inferential methods such as the omnibus F test employed in ANOVA. Two general approaches, sometimes intertwined, are available in light of this perspective. Robust methods often employ alternative statistics for estimation done with essential descriptive statistics such as means and standard deviations and employ them in analyses. A large class of modified central tendency estimators can be employed. The second class of alternative methods are the resampling methods that we have covered elsewhere. In this chapter, a few examples will be shown for the application of both classes of these methods.

10.1 Robust statistics and 1-way ANOVA

Wilcox has implemented an easy to use suite of tools in an R package for Robust Methods (Mair and Wilcox, 2018). Some background in concepts of robust central tendency estimators is useful in fully understanding the approaches - such methods as trimming, and winsorizing. The function operates by doing 20% trimming for computation of the location indices, by default and then employs the Welch F method seen above. An effect size statistic is also obtained

```
t1way(formula = dv~factora, data = hays)
```

```
## Call:
## t1way(formula = dv ~ factora, data = hays)
##
## Test statistic: F = 8.1743
## Degrees of freedom 1: 2
## Degrees of freedom 2: 9.65
## p-value: 0.00838
##
## Explanatory measure of effect size: 0.68
## Bootstrap CI: [0.26; 1.07]
```

Pairwise multiple comparisons can also be performed, on these trimmed means, with a correction for error rate inflation built in.

```
lincon(formula = dv~factora, data = hays)
```

```
## Call:
## lincon(formula = dv ~ factora, data = hays)
##
##                    psihat ci.lower ci.upper p.value
## control vs. fast  7.16667  1.57906 12.75427 0.01221
## control vs. slow  5.33333  0.98750  9.67916 0.01221
## fast vs. slow    -1.83333 -7.15661  3.48994 0.34025
```

For both functions, control of the degree of trimming can be implemented with the "tr" argument (results not shown).

```
t1way(formula = dv~factora, data = hays, tr=.10)
```

```
## Call:
## t1way(formula = dv ~ factora, data = hays, tr = 0.1)
##
## Test statistic: F = 4.3376
## Degrees of freedom 1: 2
## Degrees of freedom 2: 12.02
## p-value: 0.03817
##
## Explanatory measure of effect size: 0.65
## Bootstrap CI: [0.36; 1.01]
```

```
lincon(formula = dv~factora, data = hays, tr=.10)
```

```
## Call:
## lincon(formula = dv ~ factora, data = hays, tr = 0.1)
##
##                   psihat ci.lower ci.upper p.value
```

```
## control vs. fast  6.500 -0.50481 13.50481 0.05124
## control vs. slow  5.375  0.04328 10.70672 0.05124
## fast vs. slow    -1.125 -7.21691  4.96691 0.60863
```

The `med1way` function from **WRS2** performs a 1-way

```
med1way(formula = dv~factora, data = hays, iter=1000)
```

```
## Call:
## med1way(formula = dv ~ factora, data = hays, iter = 1000)
##
## Test statistic F: 1.3439
## Critical value: 1.2377
## p-value: 0.046
```

10.2 Resampling methods: Permutation testing

I have been using the **lmPerm** package for permutation tests. Unfortunately the source code has been moved into the CRAN archive for unsupported packages. It is, nontheless, accessible. Follow these steps

1. Download the file lmPerm_1.1-2.tar.gz from http://cran.r-project.org/src/contrib/Archive/lmPerm/, and save it to a convenient place on your local machine.
2. MS Windows users may need to install RTools from http://cran.r-project.org/bin/windows/Rtools/. Mac and Linux users can skip this step.
3. Do install.packages(file.choose(), repos=NULL, type="source")

The `aovp` function is simple. Its defaults

```
#library(lmPerm)
#library(multcomp)
set.seed(15) # so that reruns of this produce the same outcome as seen here
fit <- aovp(dv~factora, data=hays, perm="Prob")
```

```
## [1] "Settings:  unique SS "
```

```
anova(fit)
```

```
## Analysis of Variance Table
##
## Response: dv
##           Df R Sum Sq R Mean Sq Iter Pr(Prob)
## factora    2   233.87   116.933 5000    0.002 **
## Residuals 27   535.60    19.837
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Alternatively…..

```
#library(coin)
independence_test(dv ~ factora,
                  data = hays)
```

```
##
##  Asymptotic General Independence Test
##
## data:  dv by factora (control, fast, slow)
## maxT = 2.9574, p-value = 0.008792
## alternative hypothesis: two.sided
```

Pairwise comparisons….

```
#library(rcompanion)
ppairs <- pairwisePermutationTest(dv ~ factora,
                                  data = hays,
                                  method="fdr")

ppairs
```

```
##           Comparison   Stat  p.value p.adjust
## 1 control - fast = 0  2.334  0.01958  0.02937
## 2 control - slow = 0  2.576 0.009982  0.02937
## 3    fast - slow = 0 -0.336   0.7369  0.73690
```

Or organized another way:

```
ppm <- pairwisePermutationMatrix(dv ~ factora,
                                 data = hays,
                                 method="fdr")

ppm
```

```
## $Unadjusted
##         control    fast     slow
## control      NA 0.01958 0.009982
## fast         NA      NA 0.736900
## slow         NA      NA       NA
##
## $Method
## [1] "fdr"
##
## $Adjusted
##         control    fast    slow
## control 1.00000 0.02937 0.02937
## fast    0.02937 1.00000 0.73690
## slow    0.02937 0.73690 1.00000
```

10.3 Resampling methods: Bootstrapping

This section briefly reviews implementation of bootstrapping capability from two different R packages, **WRS2** and **lmboot**.

10.3.1 Percentile t boostrapping along with robust mean estimation.

Wilcox recommends a percentile t method of bootstrapping for 1way ANOVA (Wilcox, 2016). This is implemented with the `t1waybt` function which also permits specification of the amount of trimming for means, thus adding a further robustness feature to the analysis. This method is appropriate when either (or both) heteroscedasticity and non-normality are present. The variance explained is a proportion of variance and the Effect size is a cohen's f value. These value will differ from what was obtained previously, because of the trimming and the bootstrapping for error terms.

```
t1waybt(dv~factora,tr=.2,nboot=800, data=hays)
```

```
## Warning in t1waybt(dv ~ factora, tr = 0.2, nboot = 800, data = hays): Some
## bootstrap estimates of the test statistic could not be computed.
```

```
## Call:
## t1waybt(formula = dv ~ factora, data = hays, tr = 0.2, nboot = 800)
##
## Effective number of bootstrap samples was 796.
##
## Test statistic: 8.1743
## p-value: 0.01382
## Variance explained: 0.466
## Effect size: 0.683
```

The user can employ another **WRS2** function, `mcppb20`, to do pairwise post hoc tests comparing pairs of groups within the bootstrapping and trimming framework:

```
mcppb20(dv~factora,tr=.2,nboot=800, data=hays)
```

```
## Call:
## mcppb20(formula = dv ~ factora, data = hays, tr = 0.2, nboot = 800)
##
##                    psihat ci.lower ci.upper p-value
## control vs. fast  7.16667  0.33333 12.33333  0.0150
## control vs. slow  5.33333  1.16667 10.16667  0.0025
## fast vs. slow    -1.83333 -6.16667  3.66667  0.4575
```

10.4 Residual/Wild Bootsrapping

The **lmboot** package provides a function to implement Residual/Wild bootstrapping, which is recommended for models with heteroscedasticity. The simple output is only the p-value for the test of the single IV in 1way ANOVA. The modeling is based on Type I SS by default and cannot be changed. I rerun the base model so that the results can be compared. The `ANOVA.boot` function permits specification of both a seed that helps with reproducibility within this document, and number of bootstrap samples ("B").

```
#library(lmboot)
anova(aov(dv~factora, data=hays))
```

```
## Analysis of Variance Table
##
## Response: dv
##           Df Sum Sq Mean Sq F value   Pr(>F)
## factora    2 233.87 116.933  5.8947 0.007513 **
## Residuals 27 535.60  19.837
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
boot1 <- ANOVA.boot(dv~factora, data=hays, seed=1234, B=5000)
boot1$`p-values` #bootstrap p-values for 1-way model
```

```
## [1] 0.0058
```

Chapter 11

Nonparametric approaches to 1-way ANOVA problems

Non-parametric statistics had, for many decades, been a preferred choice when distributional assumptions are not met for parametric tests (normality assumption), along with scale transformation methods. Nonparametric methods employ a transformation of the DV to ranks and then inferences are made using the ranking values. For tests of group differences such as the independent samples situation (2 sample t-test), or a multiway oneway layout such as the 1way ANOVA models, the parametric tests are understood to be tests of hypotheses about population means. It is important to remember that for the non-parametric analogs, the hypothese are "location" hypotheses, not narrowly tied to means and also not explicit hypothese about medians, a mistake that is often made. These tests may also still be sensitive to assumptions regarding dispersion, so should not be considered as alternatives when the homogeneity of within-group variance assumptions are violated.

More recent developments such as permutation tests and bootstrapping are alternative ways of doing inferences when normality assumptions are violated. Their usage has increased, relative to the nonparametric tests, because of the change in computational power of modern computers. Robust methods are often combined with bootstrapping (Wilcox, 2016).

A good general reference for non-parametric methods is the textbook by Hollander, Wolfe, and Chicken (2013).

11.1 The Kruskal-Wallis Test for the 1way layout

The Kruskall-Wallis test is a well-accepted method for doing an omnibus 1way nonparametric analysis. It is implemented in several places in R, including this base system function, `kruskal.test`. The implementation uses the standard model formula approach and produces a test statistic that uses the Chi-squared distribution, where df are #groups minus one.

```
kruskal.test(hays$dv~hays$factora)
```

```
##
##  Kruskal-Wallis rank sum test
##
## data:  hays$dv by hays$factora
## Kruskal-Wallis chi-squared = 8.1309, df = 2, p-value = 0.01716
```

11.2 Follow ups to Kruskal

One method of following up an omnibus Kruskal-Wallis test is to compare pairs of groups using the Wilcoxon rank sum test (same as Mann-Whitney U test). This is somewhat like using multiple t-tests to follow up a 1way ANOVA, a method that is not recommended. However in the non-parametric situation this is one recommended approach, as long as corrections for error-rate inflation are employed. The `pairwise.wilcox.test` will do just such analyses of all possible pairs of groups. It implements the `p.adjust` method seen in an earlier section of this document, permitting p value adjustments for multiple comparison based Type I error inflation. The warning just means that the p values are approximated since the algorithm for handling ties produces an inability to produce exact calculations - and is an accepted outcome.

```
pairwise.wilcox.test(hays$dv,hays$factora, p.adjust.method="holm")
```

```
##
##  Pairwise comparisons using Wilcoxon rank sum test with continuity correction
##
## data:  hays$dv and hays$factora
##
##      control fast
## fast 0.056   -
## slow 0.030   0.447
##
## P value adjustment method: holm
```

```
                          # Adjusts p-values for multiple comparisons;
                          # See ?p.adjust for options
```

11.3 A test by Dunn for comparing pairs of groups

Alternatively the `dunn.test` function does both the omnibus test as a kruskal-wallis test and uses Dunn's method of pairwise comparisons that also permits p-value adjustment for multiple comparisons. Note that the p values are not the same for the three comparisons as found above with `pairwise.wilcox.test`. This is because the Dunn test takes a slightly different approach. One tricky part of using `dunn.test` is that the returned p values are one-tailed and should be compared to the chosen alpha rate/2.

```
#library(dunn.test)
dunn.test(hays$dv, hays$factora, method="holm")
```

```
##   Kruskal-Wallis rank sum test
##
## data: x and group
## Kruskal-Wallis chi-squared = 8.1309, df = 2, p-value = 0.02
##
##
##                             Comparison of x by group
##                                      (Holm)
## Col Mean-|
## Row Mean |    control       fast
## ---------+----------------------
##     fast |   2.647794
##          |    0.0122*
##          |
##     slow |   2.240441  -0.407353
##          |     0.0251     0.3419
##
## alpha = 0.05
## Reject Ho if p <= alpha/2
```

```
          # Adjusts p-values for multiple comparisons
          # See ?dunnTest for other options for p value adjustments
```

Chapter 12

A Larger Design and Trend Analysis

In order to expand the 1way ANOVA illustrations in this document, this chapter uses a different data set that permits implementation of trend analysis and incorporates an unequal sample size design. The data set comes from a laboratory study of ethanol effects on behavioral activity of mice (Data from B. Dudek). It is a dose-response study that is a completely randomized design with four doses of alcohol plus a control group. This chapter does a full analysis of this data set, employing many of the methods outlined in earlier chapters. Evaluation of the shape of the does response function is addressed with the use of Orthogonal Polynomial contrasts.

12.1 Import and process the data

In this data set, the DV is average mouse running speed (cm/sec, called speed15) in a fifteen minute test in an enclosed activity monitoring apparatus. The independent variable is called dose and has five levels: saline/control, 1.0, 1.5, 2.0, and 2.5 g/kg ethanol doses. The expection was that lower doses would disinhibit behavior (more/faster running in the test apparatus) and then higher doses would begin to depress or sedate behavior, producing a biphasic dose response function. Total sample size was N=234 mice randomly assigned to the five conditions.

```
# read data from mouse alcohol/activity data set used earlier for the SPSS illustration
# data file is mouse_alcohol_doseresponse2b.csv
#mouse <- read.csv(file.choose(), stringsAsFactors=T)
mouse <- read.csv("data/mouse_alcohol_doseresponse2b.csv", stringsAsFactors = TRUE)
str(mouse)
```

```
## 'data.frame': 234 obs. of  2 variables:
##  $ dose   : Factor w/ 5 levels "1 g/kg","1.5 g/kg",..: 5 1 5 1 3 3 3 3 4 5 ...
##  $ speed15: num  9.76 13.35 10.72 13.56 14.22 ...
```

```
head(mouse)
```

```
##     dose   speed15
## 1 SALINE  9.762282
## 2 1 g/kg 13.351472
## 3 SALINE 10.724117
## 4 1 g/kg 13.560831
## 5 2 g/kg 14.222366
## 6 2 g/kg 15.052335
```

The characteristics of the "dose" variable create an issue for use as a factor in `aov` and `lm` models. This is because the nominal labels are alphabetically ordered and the control group is the last in that order (numbers come first in alphabetical ordering)

```
levels(mouse$dose)
```

```
## [1] "1 g/kg"   "1.5 g/kg" "2 g/kg"   "2.5 g/kg" "SALINE"
```

This ordering is not relevant for finding the omnibus F test in the `aov` or `lm` models but it will create a problem for evaluation of the trend components which will assume increasing order of the levels. Those levels were zero (called saline), 1, 1.5, 2, and 2.5 g/kg. We can change the order of those levels for the dose factor with the `ordered` function. This will enable proper use of the `contr.poly` specification for trend contrasts.

```
# force dose to be an ordered factor so that plots and tables maintain
# the correct order of the levels
mouse$dose <- ordered(mouse$dose,
                      levels=c("SALINE","1 g/kg","1.5 g/kg","2 g/kg","2.5 g/kg"))
levels(mouse$dose)
```

```
## [1] "SALINE"   "1 g/kg"   "1.5 g/kg" "2 g/kg"   "2.5 g/kg"
```

The original code for dose as a nominal variable/factor also creates problems for drawing line graphs with dose as the X axis variable. So, a new variable, "edose" is created to be explicitly numeric.

```
# create a new variable that is the quantitative scale that dose is expressed on
# this will be needed to draw line graphs
mouse[which(mouse$dose == "SALINE"),"edose"] <- 0
mouse[which(mouse$dose == "1 g/kg"),"edose"] <- 1
mouse[which(mouse$dose == "1.5 g/kg"),"edose"] <- 1.5
mouse[which(mouse$dose == "2 g/kg"),"edose"] <- 2
mouse[which(mouse$dose == "2.5 g/kg"),"edose"] <- 2.5
class(mouse$edose)
```

```
## [1] "numeric"
```

The attach function is used to simplify our code even though current best practices in R recommend against using it. The downside of using it is not encountered in the illustrations in this chapter. It makes code for the graphing functions simpler.

```
attach(mouse)
```

12.2 Exploratory Data Analysis: Numeric Summaries

Characteristics of the data set are initially explored with the `describeBy` function from **psych**. We can see that the sample sizes per group are nearly equal. One additional characteristic should be noted from this table. The standard deviations (and thus the variances) of the five groups differ over two-fold. Close examination of the homogeneity assumption will further evalauate this pattern.

```
# obtain descriptive statistics on the three groups using the psych package
require(psych)
tab1 <- describeBy(speed15,dose,mat=T, digits=3,type=2)
row.names(tab1) <- NULL
gt(tab1[2:15])
```

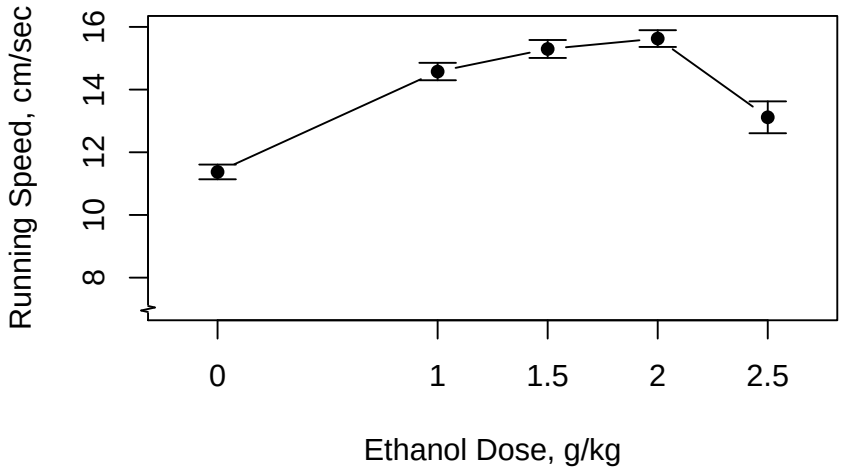
group1	vars	n	mean	sd	median	trimmed	mad	min	max	range	skew	kur
SALINE	1	46	11.373	1.592	11.191	11.296	1.255	8.477	15.319	6.842	0.493	0
1 g/kg	1	47	14.576	1.906	13.824	14.420	1.429	11.539	19.826	8.288	0.861	0
1.5 g/kg	1	47	15.296	1.963	15.442	15.320	2.182	10.824	18.960	8.136	-0.185	-0
2 g/kg	1	47	15.629	1.832	15.766	15.649	1.499	10.735	20.594	9.858	-0.060	1
2.5 g/kg	1	47	13.116	3.487	13.422	13.098	4.711	6.997	20.088	13.091	0.021	-1

12.3 Exploratory Data Analysis: Graphical exploration

A line graph with standard error bars is the expected summary figure for evaluation of dose-response curves. Note that the code uses the "edose" variable that is numerically coded rather than the original "dose" variable.

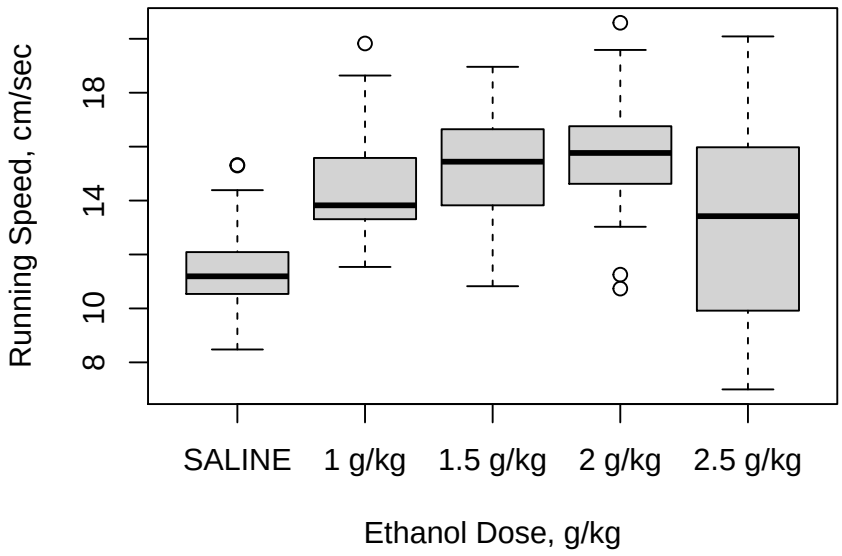
```
# draw line graph using sciplot package
# difficult to get correct line graph with other R functions
# plotmeans and plotCI give nice graphs but X axis is not scaled as continuous
# lineplot.CI defaults to using std error bars, and the X axis is scaled properly
# error bars could be switched to CI's if desired.  See the help function on lineplot.CI
lineplot.CI(edose,speed15,x.cont=T,
```

```
            xlab="Ethanol Dose, g/kg",
                        ylab="Running Speed, cm/sec",
                        ylim=c(7,15.99))
# add axis break to indicate truncated axis
# axis.break, in the plotrix package, can add a double slash or zigzag
#    axis break indicator to any open plot
library(plotrix)
axis.break(2,7,style="zigzag")
```



A boxplot can also be informative. The numeric "edose" variable is not needed here since the X axis in the box plot is not scaled. Recall that the correct ordering of levels of the dose variable occur because of the reording done above.

```
boxplot(speed15~dose,ylab="Running Speed, cm/sec",xlab=("Ethanol Dose, g/kg"))
```



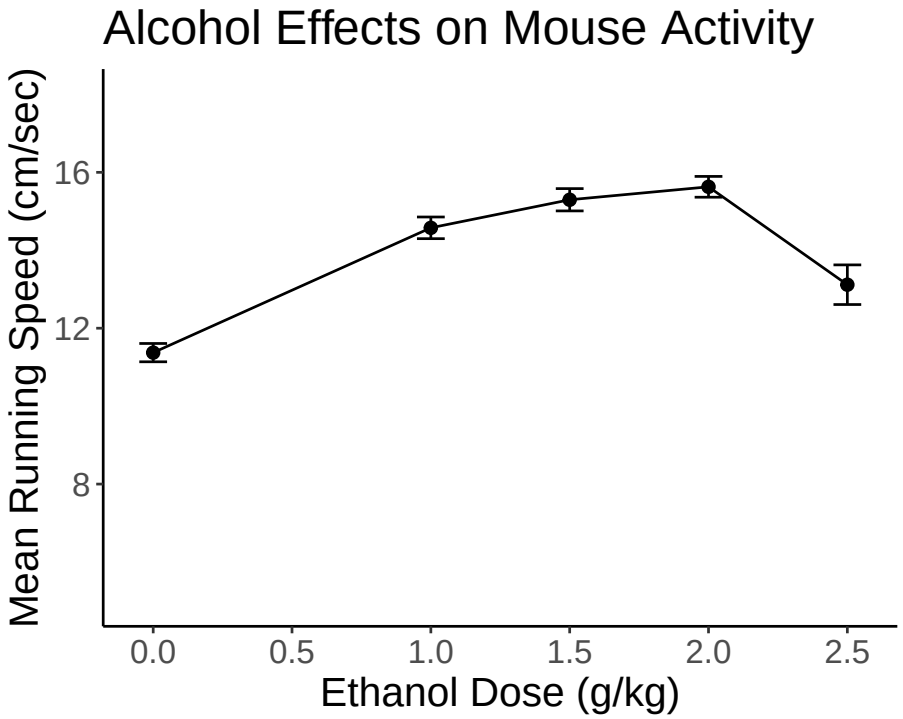
The **ggplot2** package provides capabilities to produce plots that are publication quality. For the type of graph desired here, a line graph with Std Error bars, it takes some work to generate the plot. The `ggplot` function will not permit drawing the line graph of means +/- SEM's directly from the data frame containing the raw data. Instead, a new data frame must be created, containing the summary statistics for each group. This is done with the `summarySE` function described in the code chunk. Once created, the summary data frame can be used by `ggplot`. Here, we use the "edose" variable since it is a numeric variable.

```
#   GGPLOT2 can draw line graphs with error bars
#   this approach requires some preliminary work to establish the means and std errors to b
#library(ggplot2)
#library(plyr)
# the summarySE function produces a data frame that has the summary stats by group
# it comes from http://www.cookbook-r.com/Graphs/Plotting_means_and_error_bars_(ggplot2)/
source("summarySE.R")
# now use summarySE on our data
mouse_summ <- summarySE(mouse,measurevar="speed15", groupvars="edose")
#str(mouse_summ)
# rename the column that contains the mean to something more less confusing
colnames(mouse_summ) <- c("edose", "N", "mean", "sd", "se", "95%ci" )
kable(mouse_summ)
```

edose	N	mean	sd	se	95%ci
0.0	46	11.37323	1.592286	0.2347698	0.4728507
1.0	47	14.57600	1.905680	0.2779720	0.5595286
1.5	47	15.29618	1.962592	0.2862734	0.5762384
2.0	47	15.62853	1.831644	0.2671728	0.5377909
2.5	47	13.11634	3.487325	0.5086786	1.0239169

The critical elements in creating the `ggplot` graph are the first line, the `geom_line` specification and the `geom_errorbar` specification. The remaining code controls text attributes and aesthetic properties of the graph. Note that in the `ggplot` figure, there is not a simple way to give the visual indicator that the Y axis has a break - the `axis.break` function does not work in the `ggplot` environment. The break symbols could be manually added, but I didn't do that work here.

```
# now draw the plot
#win.graph() # or quartz() or x11()
ggplot(mouse_summ, aes(x=edose, y=mean)) +
  expand_limits(y=c(5,18)) +
  expand_limits(x=c(0,2.5)) +
  scale_y_continuous(breaks=0:4*4) +
  scale_x_continuous(breaks=0:5*.5) +
  geom_line() + geom_point(size=2) +
  geom_errorbar(aes(ymin=mean-se, ymax=mean+se), colour="black", width=.1)+
  xlab("Ethanol Dose (g/kg)")+
  ylab("Mean Running Speed (cm/sec)")+
  ggtitle("Alcohol Effects on Mouse Activity")+
  theme_classic()+
  theme(text = element_text(size=16))
```




```
# code for bare minimum drawing of the line graph
#ggplot(mouse_summ, aes(x=edose, y=mean)) +
#  geom_line() +
#  geom_errorbar(aes(ymin=mean-se, ymax=mean+se), colour="black", width=.1)
```

12.4 Fit the base 1way aov model

The `aov` and `lm` functions require use of factors as IVs when performing analyses of variance. Therefore, the original "dose" variable is used throughout. It is important that we ordered the levels of that factor so that the orthononal polynomial trend contrasts properly match the levels.

First, we fit the base omnibus mode.

```
# fit basic analysis of variance model using aov
fit1t.aov <- aov(speed15~dose, data=mouse)
#summary(fit1.aov)
anova(fit1t.aov)
```

```
## Analysis of Variance Table
##
## Response: speed15
##            Df  Sum Sq Mean Sq F value    Pr(>F)
```

```
## dose        4  573.28 143.321  28.002 < 2.2e-16 ***
## Residuals 229 1172.08   5.118
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
# to show that for a 1way design, SS TYPE changes don't
# affect the omnibus BG SS
Anova(fit1t.aov,type=3)
```

```
## Anova Table (Type III tests)
##
## Response: speed15
##             Sum Sq  Df  F value    Pr(>F)
## (Intercept)  45848   1 8957.719 < 2.2e-16 ***
## dose           573   4   28.002 < 2.2e-16 ***
## Residuals     1172 229
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

12.5 Find the effect size indicators for the dose effect.

Initially, use the `anova_stats` function to obtain several effect size indicators.

```
anova_stats(fit1t.aov)
```

```
##           term  df    sumsq  meansq statistic p.value etasq partial.etasq
## dose      dose   4  573.283 143.321    28.002       0 0.328         0.328
## ...2 Residuals 229 1172.080   5.118        NA      NA    NA            NA
##      omegasq partial.omegasq epsilonsq cohens.f power
## dose   0.316           0.316     0.317    0.699     1
## ...2      NA              NA        NA       NA    NA
```

Next, use the `etaSquared` function from **lsr**.

```
#library(lsr)
#etaSquared(fit.1, type=1)
#etaSquared(fit.1, type=2)
etaSquared(fit1t.aov, type=3)
```

```
##         eta.sq eta.sq.part
## dose 0.3284605   0.3284605
```

12.6 Implement orthogonal trend analysis

Once we learn how to create the orthogonal polynomial trend coefficients, we can use the same method as previously seen with the Hays data set to obtain

SS and tests of those contrasts, using 'split' inside the `summary` function. Or we can use the `summary.lm` function. Fortunately, R has a built-in capability for orthogonal polynomials with the `contr.poly` function. It also permits exact specification of the numerical levels/spacing of the IV in the instance where levels of the quantitative IV are not equally spaced (as is our case here).

```
# now create the contrasts for trend analysis
# we need to define the contrasts based on an IV that
# has unequally spaced intervals
# by using the contr.poly function
# 5 levels with the IV values listed here
contrasts.dose <- contr.poly(5,scores=c(0,1,1.5,2,2.5))
contrasts(dose) <- contrasts.dose
contrasts(dose)
```

```
##                    .L         .Q         .C          ^4
## SALINE   -0.72782534  0.4907292 -0.1676525  0.03671115
## 1 g/kg   -0.20795010 -0.4728845  0.6311625 -0.36711155
## 1.5 g/kg  0.05198752 -0.4595010 -0.2169621  0.73422310
## 2 g/kg    0.31192515 -0.1159905 -0.6213006 -0.55066732
## 2.5 g/kg  0.57186277  0.5576469  0.3747527  0.14684462
```

Parenthetically, note that if the levels of the dose variable were equally spaced the could could simply have been the following. The `contr.poly` specifier does not even require parentheses in this case.

```
# code not used since the levels in our illustration were not equally spaced.
contrasts.dose <- contr.poly
contrasts(dose) <- contrasts.dose
contrasts(dose)
```

Next, we can fit the omnibus model again and then partition the trend SS and F tests inside the summary function. This time, the orthogonal polynomial trend coefficients will be used for the contrasts and tests of those contrasts will be provided using the `split` argument. Caution: This example data set has unequal sample sizes, so thse F tests are tests of TYPE I SS.

```
# note:  summary with split on an aov object yields type I SS
# for these F tests
# use summary.lm, as below, to obtain Type III tests on either an
# aov object or directly on an lm object with summary.
fit2t.aov <- aov(speed15~dose)
summary(fit2t.aov,
        split=list(dose=list(linear=1, quadratic=2, cubic=3,quartic=4)))
```

```
##                     Df Sum Sq Mean Sq F value   Pr(>F)
## dose                 4  573.3   143.3  28.002  < 2e-16 ***
##   dose: linear       1  157.7   157.7  30.808 7.85e-08 ***
##   dose: quadratic    1  377.1   377.1  73.679 1.42e-15 ***
```

```
##   dose: cubic       1   31.6    31.6   6.175   0.0137 *
##   dose: quartic     1    6.9     6.9   1.345   0.2473
## Residuals         229 1172.1     5.1
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

The alternative approach to providing inferences on the `aov` model object is to use the `summary.lm` function. This will produce t-tests, but they are equivalent to Type III SS F tests. Note that the squares of the t values do not exactly equal the F values produced above by the split argument in `summary` because of this Type I vs Type III distinction.

```
# or
summary.lm(fit2t.aov)
```

```
##
## Call:
## aov(formula = speed15 ~ dose)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -6.1194 -1.4152 -0.0894  1.3701  6.9716
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  13.9981     0.1479  94.645  < 2e-16 ***
## dose.L        1.8621     0.3319   5.610 5.80e-08 ***
## dose.Q       -2.8387     0.3309  -8.580 1.45e-15 ***
## dose.C       -0.8203     0.3301  -2.485   0.0137 *
## dose^4       -0.3827     0.3300  -1.160   0.2473
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 2.262 on 229 degrees of freedom
## Multiple R-squared:  0.3285,Adjusted R-squared:  0.3167
## F-statistic:    28 on 4 and 229 DF,  p-value: < 2.2e-16
```

Alternatively, we can do the trend analysis with `lm` and obtain the parameter estimates instead of variance partitioning as was the case with applying `summary.lm` to the `aov` object. Note that the orthogonal polynomial contrasts are still in effect from above. The results produce t values that are close, but not identical, to the square roots of the F tests seen just above but they match the t values from `summary.lm`. If the design has equal sample size then these three different ways of obtaining inferences on the contrasts will all match.

```
fit3t.lm <- lm(speed15~dose)
summary(fit3t.lm)
```

```
##
## Call:
## lm(formula = speed15 ~ dose)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -6.1194 -1.4152 -0.0894  1.3701  6.9716
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  13.9981     0.1479  94.645  < 2e-16 ***
## dose.L        1.8621     0.3319   5.610 5.80e-08 ***
## dose.Q       -2.8387     0.3309  -8.580 1.45e-15 ***
## dose.C       -0.8203     0.3301  -2.485   0.0137 *
## dose^4       -0.3827     0.3300  -1.160   0.2473
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 2.262 on 229 degrees of freedom
## Multiple R-squared:  0.3285,Adjusted R-squared:  0.3167
## F-statistic:    28 on 4 and 229 DF,  p-value: < 2.2e-16
```

12.7 Effect size proportion of variance estimates for contrasts.

A manual approach is used since I am still looking for an efficient way to generate effect size estimates for analytical contrasts. Find SStotal to be used as the denominator for eta squared computations for each contrast. Then create a vector of the SS for those four trend components. Then Divide the vector by SStotal for the four eta squared estimates.

A second useful descriptive statistic takes each trend component SS and divides by the SSbetween. This yields a proportion statistic that characterizes the percentage of the between-group effect that each orthogonal trend component accounts for. I used the SS values from the `summary` function application to the `aov` object.

```
# first define SStotal
sst <- var(speed15)*(length(speed15)-1)
sst
```

```
## [1] 1745.362
```

```
#Then create a vector of the SS components for trend
sstrend <- c(157.7,377.1,31.6,6.9)
names(sstrend) <- c("linear", "quadratic", "cubic", "quadratic")
```

```
sstrend
```

```
##    linear quadratic     cubic quadratic
##     157.7     377.1      31.6       6.9
```

```
sstrend/sst
```

```
##      linear   quadratic       cubic   quadratic
## 0.090353733 0.216058293 0.018105123 0.003953334
```

```
sstrend/573.3
```

```
##     linear  quadratic      cubic  quadratic
## 0.27507413 0.65777080 0.05511948 0.01203558
```

12.8 using emmeans for trend analysis

In order to use `emmeans` to perform the orthogonal trend decomposition, we need to be able to pass the exact trend coefficients that we saw earlier. Since they were based on unequally spaced intervals and since R chooses a scale such that the sum of the squared coefficients equals 1 (orthonormalizing), the coefficients are decimal quantities. For accuracy we need to extract those exact values from the `contrasts(dose)` object and then pass them to the `emmeans` function. Initially, these coefficients are in a matrix, and I found it easier to convert that matrix to a dataframe so that the columns of coefficients can be extracted. Notice the possibility of adjusting the pvalues and/or CIs for the multiple testing situation. Just for demonstration, I did not adjust the t-test inferences but I did adjust the CIs.

```
fit2.emm.a <- emmeans(fit3t.lm, "dose", data=mouse)
trend <- as.data.frame(contrasts(mouse$dose))
# check with, e.g., trend[,4]
lincombs <- contrast(fit2.emm.a,
               list(linear=trend$.L,
                    quadratic=trend$.Q,
                    cubic=trend$.C,
                    quartic=trend$'^4'
                    ))
test(lincombs, adjust="none")
```

```
##  contrast  estimate    SE  df t.ratio p.value
##  linear       1.435 0.331 229  4.331  <.0001
##  quadratic   -3.158 0.331 229 -9.541  <.0001
##  cubic       -0.114 0.330 229 -0.346  0.7293
##  quartic     -0.544 0.330 229 -1.648  0.1007
```

```
confint(lincombs, adjust="sidak")
```

```
##  contrast  estimate    SE  df lower.CL upper.CL
##  linear       1.435 0.331 229    0.603    2.267
##  quadratic   -3.158 0.331 229   -3.990   -2.327
##  cubic       -0.114 0.330 229   -0.944    0.715
##  quartic     -0.544 0.330 229   -1.373    0.285
##
## Confidence level used: 0.95
## Conf-level adjustment: sidak method for 4 estimates
```

These t test values match the `lm` model output (and application of `summary.lm` to the `aov` object) where regression coefficients were tested and thus relate to Type III SS. This is probably the most appropriate way to do trend analysis when there is unequal N, although it is more cumbersome than using the "split" argument.

12.9 Summary to this point in the analysis of the dose response data set

At this point, we have information from the analysis that largely fits what our eye saw from the dose response curve figures. Low doses tended to increase running speed relative to the control condition but at the highest dose, the curve began to take a downward trajectory, reflecting the expectation outlined above. In terms of the trend analysis, the shape of the curve was largely influenced by linear and quadratic components, with the quadratic bend accounting for the majority of the Between group effect of the IV (the 66% value just calculated).

12.10 Post Hoc tests

The full ANOVA and trend decomposition didn't tell us about one interesting question that emerges from examination of the dose response curve. We might as whether the 2.5 dose mean is actually different from the 2.0 dose. I.e., is the drop "significant". This is explicitly a post hoc question. The most direct way of evaluating this is with a Tukey test. I'll opt for doing both the standard Tukey HSD test and the DTK test here. DTK is applicable when there is unequal sample size and heterogeneity of variance (we had a hint of that when we examined the sd's for the five groups). Since that one pairwise comparison (2 vs 2.5) was derived from visual examination of the figure in a post hoc way, it can be argued that all pairwise comparisons were visually taken in to account. The results of the DTK test function will evaluate all of those pairwise comparisons and the alpha rate adjustment properly takes this into account.

```
# first, use the Tukey HSD test procedure
TukeyHSD(fit1t.aov)
```

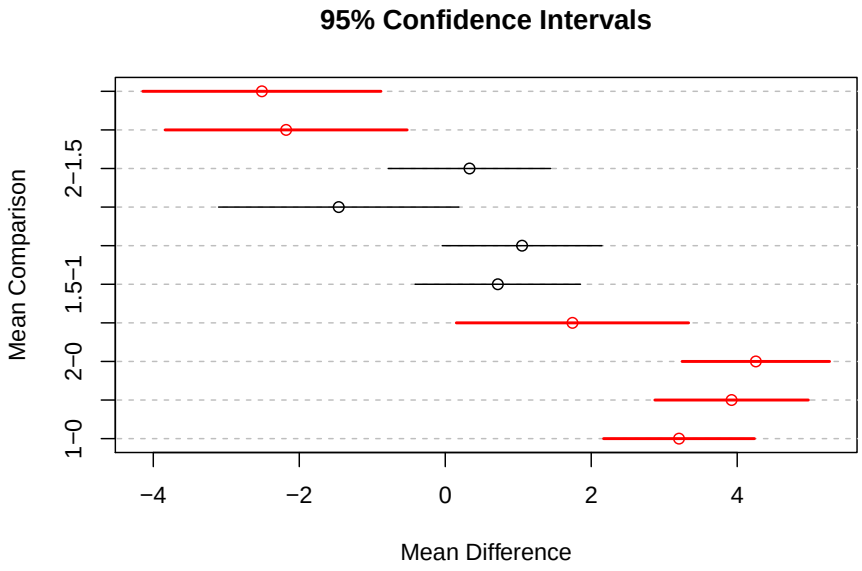
```
##   Tukey multiple comparisons of means
```

```
##     95% family-wise confidence level
##
## Fit: aov(formula = speed15 ~ dose, data = mouse)
##
## $dose
##                           diff        lwr        upr     p adj
## 1 g/kg-SALINE      3.2027633  1.9125852  4.4929414 0.0000000
## 1.5 g/kg-SALINE    3.9229500  2.6327718  5.2131281 0.0000000
## 2 g/kg-SALINE      4.2552967  2.9651186  5.5454748 0.0000000
## 2.5 g/kg-SALINE    1.7431045  0.4529263  3.0332826 0.0023542
## 1.5 g/kg-1 g/kg    0.7201867 -0.5630363  2.0034096 0.5355178
## 2 g/kg-1 g/kg      1.0525334 -0.2306895  2.3357563 0.1634975
## 2.5 g/kg-1 g/kg   -1.4596588 -2.7428818 -0.1764359 0.0168655
## 2 g/kg-1.5 g/kg    0.3323467 -0.9508762  1.6155697 0.9535766
## 2.5 g/kg-1.5 g/kg -2.1798455 -3.4630685 -0.8966226 0.0000500
## 2.5 g/kg-2 g/kg   -2.5121922 -3.7954152 -1.2289693 0.0000018
```

```
# now request the Dunnett-Tukey-Kramer test:
DTK.result <- DTK.test(speed15,edose)
DTK.result
```

```
## [[1]]
## [1] 0.05
##
## [[2]]
##                Diff    Lower CI   Upper CI
## 1-0       3.2027633  2.16944629  4.2360803
## 1.5-0     3.9229500  2.87151157  4.9743884
## 2-0       4.2552967  3.24521665  5.2653768
## 2.5-0     1.7431045  0.15203497  3.3341740
## 1.5-1     0.7201867 -0.41301751  1.8533909
## 2-1       1.0525334 -0.04240628  2.1474731
## 2.5-1    -1.4596588 -3.10589543  0.1865778
## 2-1.5     0.3323467 -0.77974361  1.4444370
## 2.5-1.5  -2.1798455 -3.83756613 -0.5221249
## 2.5-2    -2.5121922 -4.14361391 -0.8807706
```

```
DTK.plot(DTK.result)
```



Of course there other ways to do post hoc testing that could fit this data set. We might want to use the Dunnett test to compare each treatment group with control. It turns out that none of the four CIs overlaps zero, so we conclude each dose significantly raised DV scores.

```
### The DUNNETT TYPE OF TEST FOR COMPARING TREATMENTS TO A COMMON CONTROL GROUP
require(multcomp)
fit1.dunnett <- glht(fit1t.aov,linfct=mcp(dose="Dunnett"))
# obtain CI's do test each difference
confint(fit1.dunnett, level = 0.95)
```

```
##
##   Simultaneous Confidence Intervals
##
## Multiple Comparisons of Means: Dunnett Contrasts
##
##
## Fit: aov(formula = speed15 ~ dose, data = mouse)
##
## Quantile = 2.4572
## 95% family-wise confidence level
##
##
## Linear Hypotheses:
##                        Estimate lwr    upr
## 1 g/kg - SALINE == 0   3.2028   2.0498 4.3557
```

```
## 1.5 g/kg - SALINE == 0 3.9229   2.7700 5.0759
## 2 g/kg - SALINE == 0   4.2553   3.1023 5.4083
## 2.5 g/kg - SALINE == 0 1.7431   0.5901 2.8961
```

We might also wish to do pairwise comparisons with methods other than the Tukey or DTK test. We can use the `pairwise.t.test` function.

```
# This function provides several of the bonferroni style corrections
# for pairwise multiple comparisons.  Note that it permits use of
# the pooled within cell variance as the core error term.
# first, just do pairwise comparisons with bonferroni corrections for
# having done a set of three "contrasts".  should match results seen
# above in the BonferroniCI function.
pairwise.t.test(speed15,dose,pool.sd=TRUE,p.adj="bonf")
```

```
##
##  Pairwise comparisons using t tests with pooled SD
##
## data:  speed15 and dose
##
##          SALINE  1 g/kg 1.5 g/kg 2 g/kg
## 1 g/kg   7.8e-10 -      -        -
## 1.5 g/kg 6.1e-14 1.0000 -        -
## 2 g/kg   5.6e-16 0.2506 1.0000   -
## 2.5 g/kg 0.0026  0.0199 5.1e-05  1.8e-06
##
## P value adjustment method: bonferroni
```

```
# We can change the approach to the Holm, Hochberg, Hommel,
# Benjamini and Hochberg, Benjamini&Yekutieli, and fdr corrections,
# as well as "none" which will give the same thing as the LSD test.
# Choice of these depends on several factors, including whether
# the contrasts examined are independent (and they are not since they
# are all of the pairwise comparisons,
# Of these modified bonferroni type of approaches, the "BY" and "fdr"
# approaches are probably the most appropriate here, since some of our
# comparisons are correlated and BY permits that correlation to be either
# positive or negative
#pairwise.t.test(dv,factora,pool.sd=TRUE,p.adj="holm")
#pairwise.t.test(dv,factora,pool.sd=TRUE,p.adj="hochberg")
#pairwise.t.test(dv,factora,pool.sd=TRUE,p.adj="hommel")
#pairwise.t.test(dv,factora,pool.sd=TRUE,p.adj="BH")
pairwise.t.test(speed15,dose,pool.sd=TRUE,p.adj="BY")
```

```
##
##  Pairwise comparisons using t tests with pooled SD
##
## data:  speed15 and dose
```

```
##
##          SALINE  1 g/kg 1.5 g/kg 2 g/kg
## 1 g/kg   7.6e-10 -      -        -
## 1.5 g/kg 8.9e-14 0.4041 -        -
## 2 g/kg   1.6e-15 0.0917 1.0000   -
## 2.5 g/kg 0.0012  0.0083 3.0e-05  1.3e-06
##
## P value adjustment method: BY
```

```
pairwise.t.test(speed15,dose,pool.sd=TRUE,p.adj="fdr")
```

```
##
##  Pairwise comparisons using t tests with pooled SD
##
## data:  speed15 and dose
##
##          SALINE  1 g/kg  1.5 g/kg 2 g/kg
## 1 g/kg   2.6e-10 -       -        -
## 1.5 g/kg 3.0e-14 0.13796 -        -
## 2 g/kg   5.6e-16 0.03132 0.47710  -
## 2.5 g/kg 0.00043 0.00284 1.0e-05  4.5e-07
##
## P value adjustment method: fdr
```

```
#pairwise.t.test(dv,factora,pool.sd=TRUE,p.adj="none")
# note that setting pool.sd to FALSE changes the outcome since
# it employs a Welch or Fisher-Behrens type of approach
pairwise.t.test(speed15,dose,pool.sd=FALSE,p.adj="bonf")
```

```
##
##  Pairwise comparisons using t tests with non-pooled SD
##
## data:  speed15 and dose
##
##          SALINE  1 g/kg  1.5 g/kg 2 g/kg
## 1 g/kg   9.8e-13 -       -        -
## 1.5 g/kg < 2e-16 0.74370 -        -
## 2 g/kg   < 2e-16 0.07593 1.00000  -
## 2.5 g/kg 0.02771 0.14049 0.00372  0.00042
##
## P value adjustment method: bonferroni
```

```
# or
pairwise.t.test(speed15,dose,pool.sd=FALSE,p.adj="fdr")
```

```
##
##  Pairwise comparisons using t tests with non-pooled SD
##
```

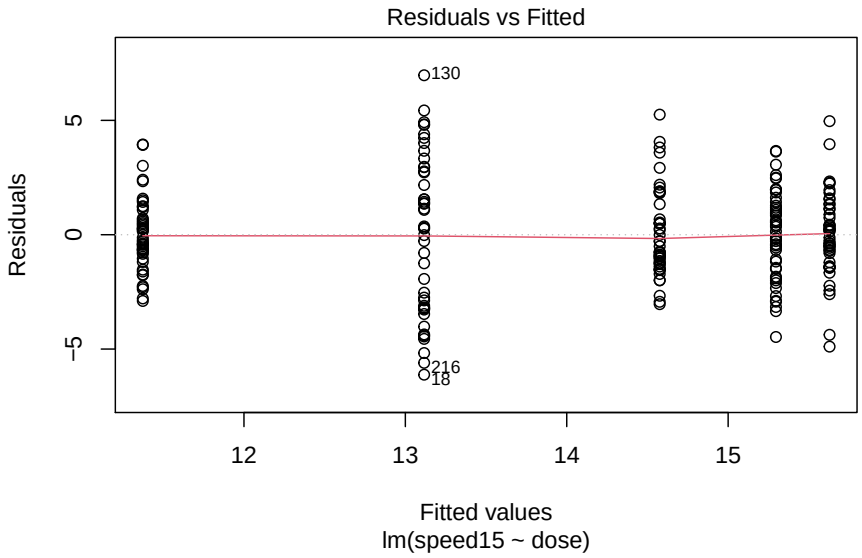
```
## data:  speed15 and dose
##
##          SALINE  1 g/kg  1.5 g/kg 2 g/kg
## 1 g/kg   3.3e-13 -       -        -
## 1.5 g/kg < 2e-16 0.08263 -        -
## 2 g/kg   < 2e-16 0.01085 0.39824  -
## 2.5 g/kg 0.00462 0.01756 0.00074  0.00011
##
## P value adjustment method: fdr
```

12.11 Evaluation of Assumptions: graphical and inferential evaluation

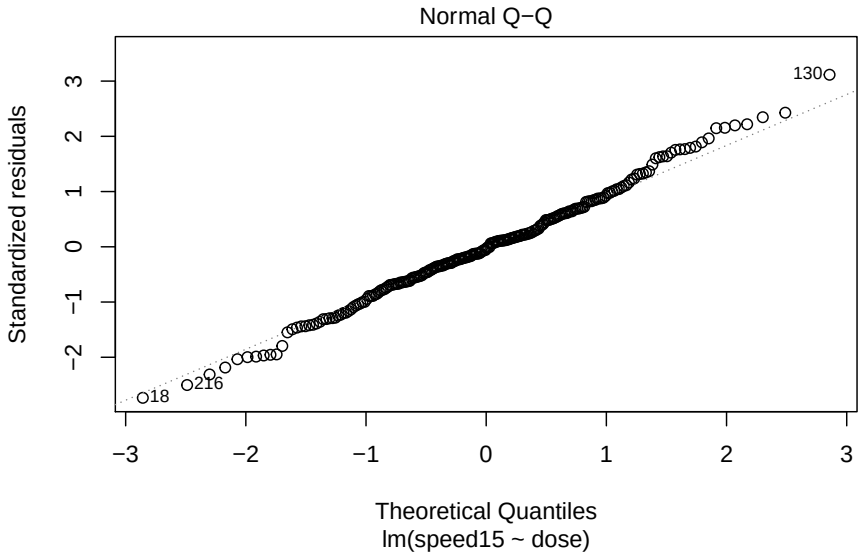
Perhaps this section is placed too late in the flow of the analyses since violations of the assumption(s) would have implications for most of the tests above.

First, we can obtain the standard pair of plots for evaluating normality and homoscedasticity of the residuals from the ANOVAs done above. It is easiest to work with the `lm` fit object. The first plot reveals the heteroscedasticity that we suspected. The group that has the largest residual spread is the group that had a mean near 13 (the 2.5 g/kg dose group), and this group had the highest standard deviation, as seen with the initial descriptive statistics. The second plot, at first glance gives a reasonable impression of a fit to normality for the residuals but closer inspection suggests a pattern where the largest outliers deviate a bit from expectation.

```
plot(fit3t.lm,which=1)
```

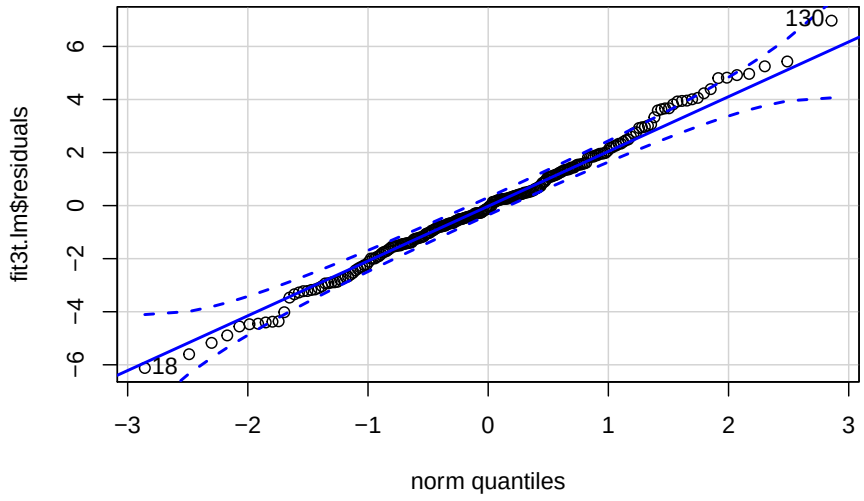



```
plot(fit3t.lm,which=2)
```



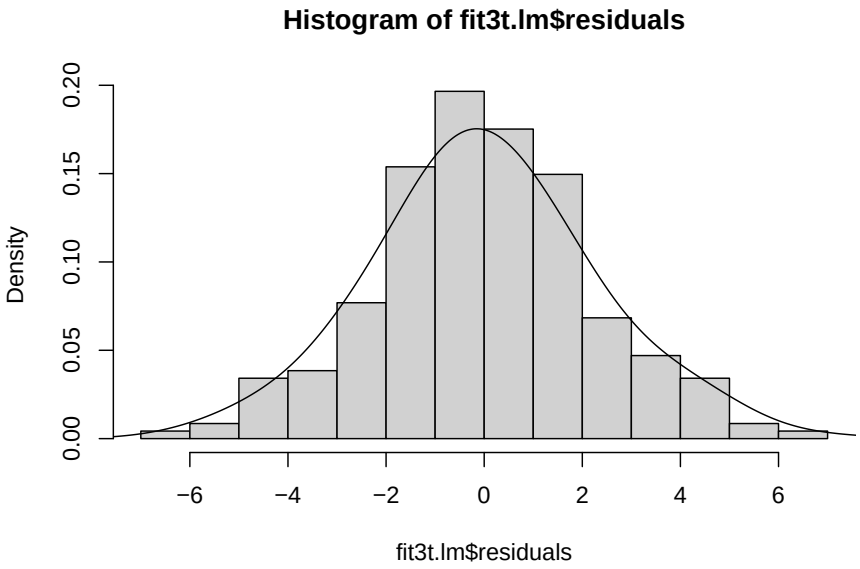
Lets pursue this potential non-normality a bit further. If we extract the residuals from the `lm` object, we can draw both the `qqPlot` figure and a frequency histogram.

```
qqPlot(fit3t.lm$residuals)
```




```
## [1] 130  18
```

```
hist(fit3t.lm$residuals, prob=TRUE, col="gray82",breaks=16)
lines(density(fit3t.lm$residuals, bw=1))
```



Both figures suggest that the normality assumption may not be violated here, but we can do a test. We find that the Anderson-Darling test does not permit rejection of the null hypothesis that normality is present.

```
#library(nortest)
ad.test(residuals(fit3t.lm)) #get Anderson-Darling test for normality (nortest package must
```

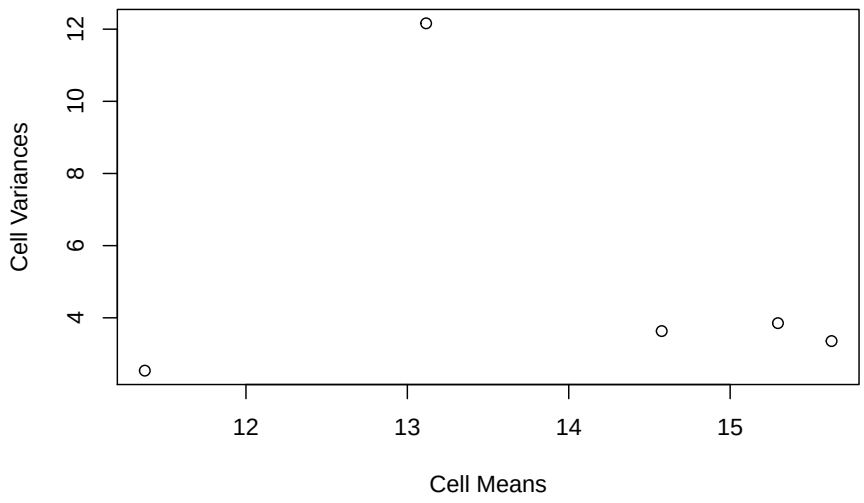
```
##
##  Anderson-Darling normality test
##
## data:  residuals(fit3t.lm)
## A = 0.42988, p-value = 0.3059
```

Now let's evaluate the Homogeneity of Variance Assumption. Among the several ways we reviewed to do this, the median-centered Levene test is probably adequate:

```
levene.test(speed15,edose,location="median")
```

```
##
##  Modified robust Brown-Forsythe Levene-type test based on the absolute
##  deviations from the median
##
## data:  speed15
## Test Statistic = 13.477, p-value = 7.011e-10
```

```
##########################################
## Plot of cell means vs cell variances ##
##########################################
#   recall that this type of plot helps evaluate whether
#   heterogeneity of variance might have arisen from a simple
#   scaling issue.  If so, then scale transformations may help.
#   E.g,, a postive mean-variance correlation reflects a situation where
#   a log transformation or a fractional exponent transformation of the DV
#   might produce homoscedasticity.
#   I'm also using the tapply function here in ways that we have not covered.
#   Tapply is an important function in dealing with factors.
plot(tapply(speed15, dose, mean), tapply(speed15, dose, var), xlab = "Cell Means",
             ylab = "Cell Variances", pch = levels(edose))
```




```
# examination of the plot shows that the heterogeneity is not simply due to a
# mean variance correlation.  the group with the highest variance is the highest
# dose group and no simple transformation can take care of the problem.
# the dispersion difference may come from a sex difference that was not examined here.
# thus the model may be underspecified and this may have produced the heterogeneity

#  recall that we can do the extension of the Welch test to 1-way anova
```

12.12 Alternate analyses when faced with Heteroscadisticity

The omnibus F test can be re-examined, using the Welch f test approach and we already reviewed that capability with the `oneway.test` function. Unsurprisingly, the omnibus F remains significant (with this large N design and strong treatment effects) at the .05 level.

```
oneway.test(speed15~dose,var.equal=F)
```

```
##
##  One-way analysis of means (not assuming equal variances)
##
## data:  speed15 and dose
## F = 46.616, num df = 4.00, denom df = 113.58, p-value < 2.2e-16
```

Wilcox argues for use of the percentile t approach to bootstrapping whenver either/or heteroscadisticy or non-normality are present. That method was simple to implement and also yields a rejection of the omnibus null hypothesis. The "tr" argument specifies the degree of trimming. This choice is not to be taken lightly and users should be familiar with the literature on trimmed means, Winsorized means, and other M-estimators. The abundant writings of Wilcox (e.g., (Wilcox, 2016)) are a good starting point.

```
# from the WRS2 package
t1waybt(speed15~edose,tr=.2,nboot=2000, data=mouse)
```

```
## Call:
## t1waybt(formula = speed15 ~ edose, data = mouse, tr = 0.2, nboot = 2000)
##
## Effective number of bootstrap samples was 2000.
##
## Test statistic: 49.8329
## p-value: 0
## Variance explained: 0.375
## Effect size: 0.613
```

Pairwise comparisons using bootstrapping also yield similar conclusions to the traditional tests done above.

```
# from the WRS2 package
mcppb20(speed15~edose,tr=.2,nboot=2000, data=mouse)
```

```
## Call:
## mcppb20(formula = speed15 ~ edose, data = mouse, tr = 0.2, nboot = 2000)
##
##               psihat ci.lower ci.upper p-value
## 0 vs. 1     -2.97790 -4.08606 -2.02812   0.000
```

```
## 0 vs. 2     -4.07103 -5.21790 -3.00591   0.000
## 0 vs. 2.5   -4.32550 -5.25236 -3.44927   0.000
## 0 vs. 1.5   -1.78139 -3.70158  0.26297   0.006
## 1 vs. 2     -1.09314 -2.22172  0.16915   0.019
## 1 vs. 2.5   -1.34761 -2.39651 -0.29189   0.001
## 1 vs. 1.5    1.19650 -0.72582  3.13702   0.081
## 2 vs. 2.5   -0.25447 -1.36754  0.86564   0.524
## 2 vs. 1.5    2.28964  0.47334  4.33964   0.000
## 2.5 vs. 1.5  2.54411  0.49023  4.60210   0.000
```

I am not yet confident of a way to implement bootstrapping for analytical contrasts, such as trend analysis, in R. It should be possible to do it by bootstrapping the `lm` model fit. One approach is found in the Resampling chapter.

Chapter 13

Unequal Sample Sizes

The presence of unequal samples sizes has major implications in factorial designs that require care in choice of SS decomposition types (e.g., Type I vs II, vs III). For 1-way layouts, the impact is more minimal. However, there are some things to be aware of, particularly with regard to use of contrasts, and this chapter addresses those issues.

The critical starting point in understanding why the issue arises with contrasts (even "orthogonal" ones), is to know that when a design is unbalanced (unequal n's), the coding vectors (when applied to all cases) are not completely uncorrelated. Thus the regression modeling is not clean. It has to cope with correlated IVs and this is where the Type I, II, and III SS issues come into play.

13.1 Import the data and Describe

We can use the same original data set from earlier parts of this tutorial, the "hays" data set. However, I randomly deleted five cases from that data set, two from the control group, one from the fast group, and three from the slow group. The data are in a .csv file that is read here.

```
hays_unbal <- read.csv("data/hays_unequal.csv",stringsAsFactors=TRUE)
```

The descriptive statistics show that the three means differ a small amount from their original values in the equal-sample-size illustrations above, but not radically so.

```
describeBy(hays_unbal$dv, group=hays_unbal$factora, type=2)
```

```
##
##  Descriptive statistics by group
## group: control
##    vars n  mean   sd median trimmed  mad min max range skew kurtosis   se
```

```
## X1    1 8 26.62 5.32   26.5   26.62 4.45  19  36    17 0.32     0.37 1.88
## ------------------------------------------------------------
## group: fast
##    vars n  mean   sd median trimmed  mad min max range skew kurtosis   se
## X1    1 9 20.33 4.69     19   20.33 4.45  15  30    15 1.03     1.05 1.56
## ------------------------------------------------------------
## group: slow
##    vars n  mean   sd median trimmed  mad min max range  skew kurtosis   se
## X1    1 7 22.43 1.51     23   22.43 1.48  20  24     4 -0.62    -0.81 0.57
```

We can visualize the data set with this multiple-panel graph produced in **ggplot**.

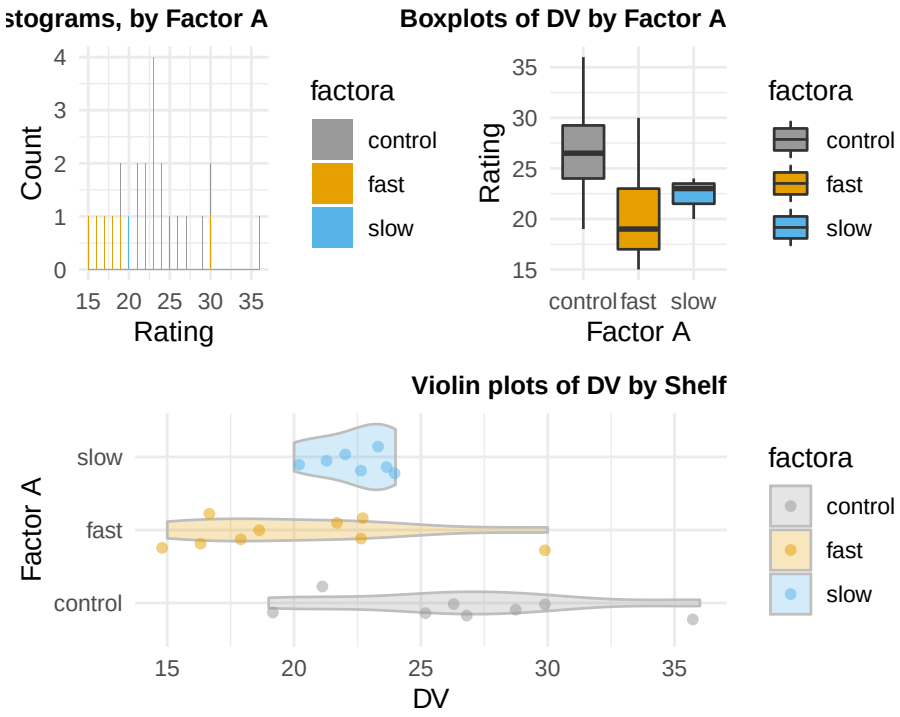
```
# Modeled after https://dmyee.files.wordpress.com/2016/03/advancedggplot.pdf
#Histograms of  DV by Shelf
#library("RColorBrewer")
cbPalette <- c("#999999", "#E69F00", "#56B4E9", "#009E73",
               "#F0E442", "#0072B2", "#D55E00", "#CC79A7")
p1<-ggplot(data = hays_unbal, aes(x = dv, fill=factora)) +
  geom_histogram(binwidth = .1) +
  scale_fill_manual(values=cbPalette) +
  #scale_fill_brewer(palette="Paired") +
  #scale_colour_grey() + scale_fill_grey() +
  xlab("Rating") + ylab("Count") +
  ggtitle("Histograms, by Factor A") +
  theme_minimal() +
  theme(plot.title = element_text(size=10, face = "bold", hjust = 1))

# Boxplots of "Healthiness" Rating" by factora
p2<-ggplot(data = hays_unbal, aes(x = factora, y = dv, fill=factora)) +
  geom_boxplot() +xlab("Factor A") + ylab("Rating") +
  scale_fill_manual(values=cbPalette) +
  #scale_fill_brewer(palette="Paired") +
  #scale_colour_grey() + scale_fill_grey() +
  ggtitle("Boxplots of DV by Factor A") +
  theme_minimal() +
  theme(plot.title = element_text(size=10, face = "bold", hjust = 1))

# Violin plots of "Healthiness" Rating" by factora
p3<-ggplot(data = hays_unbal, aes(x = factora, y = dv, fill=factora)) +
  geom_violin(alpha=.25, color="gray") +
  geom_jitter(alpha=.5, aes(color=factora), position=position_jitter(width=0.3)) +
  scale_fill_manual(values=cbPalette) + scale_colour_manual(values=cbPalette) +
  #scale_fill_brewer(palette="Paired") +
    #scale_colour_grey() + scale_fill_grey() +
  coord_flip() +
  xlab("Factor A") + ylab("DV") +
```

```
  ggtitle("Violin plots of DV by Shelf") +
  theme_minimal() +theme(plot.title = element_text(size=10, face = "bold", hjust = 1))

# Creating a matrix that defines the layout
# (not all graphs need to take up the same space)
lay <- rbind(c(1,2),c(3,3))# Plotting the plots on a grid
grid.arrange(p1, p2, p3, ncol=2, layout_matrix=lay)
```



13.2 Fit the model with aov

The omnibus ANOVA model can be fit with `aov` in the previously defined manner, and an omnibus F test is returned.

```
fit1_unbal <- aov(dv~factora, data=hays_unbal)
anova(fit1_unbal)
```

```
## Analysis of Variance Table
##
## Response: dv
##            Df Sum Sq Mean Sq F value Pr(>F)
## factora     2 171.37  85.685  4.6425 0.0214 *
```

```
## Residuals 21 387.59  18.457
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

It is useful to recall that `aov` is a wrapper for the `lm` function and thus requires the IV to be a factor so that coding vectors (contrasts) can be produced. Let's recall what the default contrasts are for a factor. The are dummy or indicator codes. Thus two coding vectors are available for this three-level factor.

```
contrasts(hays_unbal$factora)
```

```
##         fast slow
## control    0    0
## fast       1    0
## slow       0    1
```

Next we can change the coding scheme to effect, or deviation, coding.

```
contrasts(hays_unbal$factora) <- contr.sum
contrasts(hays_unbal$factora)
```

```
##         [,1] [,2]
## control    1    0
## fast       0    1
## slow      -1   -1
```

A rerun of the omnibus F test reveals that this change in coding scheme does not produce a different F value. This was what should have been expected.

```
fit2_unbal <- aov(dv~factora, data=hays_unbal)
anova(fit2_unbal)
```

```
## Analysis of Variance Table
##
## Response: dv
##           Df Sum Sq Mean Sq F value Pr(>F)
## factora    2 171.37  85.685  4.6425 0.0214 *
## Residuals 21 387.59  18.457
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

If we want to change our coding scheme to an orthogonal contrast set, we need to define those contrasts first in a matrix, and then assign that matrix to the contrasts used for the factor. This process is identical to that used in earlier sections of this document. This would be what we called the "helmert" set. (But note that R has a built-in helmert set that if we had used it would have actually looked like what we called reverse helmert, so I did not use it here.)

```
contr_special <- matrix(c(1,-.5,-.5,0,1,-1), ncol=2)
contrasts(hays_unbal$factora) <- contr_special
```

```
contrasts(hays_unbal$factora)
```

```
##         [,1] [,2]
## control  1.0    0
## fast    -0.5    1
## slow    -0.5   -1
```

Next, a refit of the omnibus model will use these analytical/orthogonal contrasts. This change in coding scheme also resulted in the same omnibus F as the prior two model fits. It should be reassuring that all three approaches produce the same outcome - they are just different (but effective) ways of coding for group membership.

```
fit3_unbal <- aov(dv~factora, data=hays_unbal)
anova(fit3_unbal)
```

```
## Analysis of Variance Table
##
## Response: dv
##           Df Sum Sq Mean Sq F value Pr(>F)
## factora    2 171.37  85.685  4.6425 0.0214 *
## Residuals 21 387.59  18.457
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Now that the basics of the omnibus F test are established to behave the same way in unbalanced designs as is the case with equal-N designs, we can move on to the question of partitioning the SSBG into contrast components and doing inferences about those contrasts.

13.3 Evaluate Analytical/Orthogonal/SingleDF contrasts

This illustration will utilized the analytical/orthogonal/singleDF contrasts just put in place above, but the message will be the same for the two other coding schemes.

The first way we have approached the assessment of contrasts is to use the "split" argument in the `summary` function, as applied to the `aov` fit object. This produces the SS partitioning and provides F tests for the two single df contrasts as well as the omnibus 2df effect. Note that the SS for the two contrasts sum to the SSBG value.

```
summary(fit3_unbal,
        split = list(factora = list(ac1 = 1, ac2 = 2)))
```

```
##                Df Sum Sq Mean Sq F value  Pr(>F)
```

```
## factora          2  171.4   85.68   4.642 0.02140 *
##   factora: ac1   1  154.1  154.08   8.348 0.00877 **
##   factora: ac2   1   17.3   17.29   0.937 0.34418
## Residuals       21  387.6   18.46
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Another way we saw to obtain inferential tests for contrasts was to test the regression coefficients created by the `aov` function. They can be obtained with the `summary.lm` function applied to the `aov` object since `aov` is merely a wrapper for `lm`.

In the work with the balanced design analyzed in earlier chapters of this document, we saw that we could square the t values for the tests of the two orthogonal contrasts and obtain the F values produced by the "split argument" used just above.

```
coefficients <- summary.lm(fit3_unbal)
coefficients
```

```
##
## Call:
## aov(formula = dv ~ factora, data = hays_unbal)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -7.6250 -2.3571 -0.0268  1.8437  9.6667
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  23.1290     0.8816  26.236   <2e-16 ***
## factora1      3.4960     1.2435   2.812   0.0105 *
## factora2     -1.0476     1.0825  -0.968   0.3442
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 4.296 on 21 degrees of freedom
## Multiple R-squared:  0.3066,Adjusted R-squared:  0.2405
## F-statistic: 4.642 on 2 and 21 DF,  p-value: 0.0214
```

So, let's extract the t value from the coefficients table and square them. We find that the squared t for the second contrast does equal the F from the "split" table above, but the first does not - the first is smaller than the one tabled in the "split" derived F table.

```
t_ac1 <- coefficients$coefficients[2,3]
t_ac2 <- coefficients$coefficients[3,3]
tvals <- c(t_ac1, t_ac2)
```

```
tvals
```

```
## [1]  2.8115374 -0.9677595
```

```
tvals^2
```

```
## [1] 7.9047423 0.9365584
```

At this point, we have two competing approaches to inference and we will need to understand them a bit better. But first, we should obtain the tests of the contrasts provided by the **emmeans** package and function, and compare. We see that the t-tests produced by the `contrast` function applied to an `emmeans` object are identical to those produced the the `summary.lm` function above and thus different from the F test of the first contrast found in the table produced by the "split" version of the `summary` function.

```
fit3_unbal.emm.a <- emmeans(fit3_unbal, "factora", data=hays_unbal)
lincombs2 <- contrast(fit3_unbal.emm.a,
               list(ac1=c(1,-.5,-.5), ac2=c(0,1,-1))) # second one not changed
test(lincombs2, adjust="none")
```

```
##  contrast estimate   SE df t.ratio p.value
##  ac1          5.24 1.87 21  2.812  0.0105
##  ac2         -2.10 2.17 21 -0.968  0.3442
```

```
#confint(lincombs2, adjust="none")
```

13.4 What is the source of the discrepancy?

The short answer to the question of the origin of this discrepancy is the difference between type I and type III SS that are employed by the two different approaches. But to illustrate this more complete, lets examine two other variables that were placed in the "unbalanced" data frame. The same two orthogonal contrasts were manually created and added as variables into the data frame.

```
gt(hays_unbal)
```

id	dv	factora	ac1	ac2
1	27	control	1.0	0
4	19	control	1.0	0
5	25	control	1.0	0
6	29	control	1.0	0
7	36	control	1.0	0
8	30	control	1.0	0
9	26	control	1.0	0
10	21	control	1.0	0
11	23	fast	-0.5	1

12	22	fast	-0.5	1
13	18	fast	-0.5	1
14	15	fast	-0.5	1
16	30	fast	-0.5	1
17	23	fast	-0.5	1
18	16	fast	-0.5	1
19	19	fast	-0.5	1
20	17	fast	-0.5	1
21	23	slow	-0.5	-1
22	24	slow	-0.5	-1
23	21	slow	-0.5	-1
26	24	slow	-0.5	-1
27	22	slow	-0.5	-1
29	20	slow	-0.5	-1
30	23	slow	-0.5	-1

Now that we have the fully instantiated coding vectors at our disposal, we can verfy the non-orthogonality of the set simply by examining the Pearson product moment correlation between the two vectors. The non-zero correlation means that we do not have a fully orthogonal set, even though the correlation is small. The redundant/overlapping/shared variance that the two IVs have in the DV must be rectified in our analyses.

```
cor(hays_unbal$ac1, hays_unbal$ac2)
```

```
## [1] -0.07254763
```

Now, we can fit the object using `lm` and control the order of entry of the coding vectors into the regression model. Note that the same omnibus F and multiple R squared are returned as with the `aov` modeling above. In addition, the regression coefficients and their t-tests are also identical to those produced with `summary.lm` on the `aov` object above and identical to the t-tests produced with the **emmeans** approach used above. These inferential tests are utilizing a method that is equivalent to employing type III SS in F tests. We can explore that below.

```
fit4_unbal.lm <- lm(dv~ac1+ac2, data=hays_unbal)
summary(fit4_unbal.lm)
```

```
##
## Call:
## lm(formula = dv ~ ac1 + ac2, data = hays_unbal)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -7.6250 -2.3571 -0.0268  1.8437  9.6667
##
```

```
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  23.1290     0.8816  26.236   <2e-16 ***
## ac1           3.4960     1.2435   2.812   0.0105 *
## ac2          -1.0476     1.0825  -0.968   0.3442
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 4.296 on 21 degrees of freedom
## Multiple R-squared:  0.3066,Adjusted R-squared:  0.2405
## F-statistic: 4.642 on 2 and 21 DF,  p-value: 0.0214
```

```
fit5_unbal.lm <- lm(dv~ac2+ac1, data=hays_unbal)
summary(fit5_unbal.lm)
```

```
##
## Call:
## lm(formula = dv ~ ac2 + ac1, data = hays_unbal)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -7.6250 -2.3571 -0.0268  1.8437  9.6667
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  23.1290     0.8816  26.236   <2e-16 ***
## ac2          -1.0476     1.0825  -0.968   0.3442
## ac1           3.4960     1.2435   2.812   0.0105 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 4.296 on 21 degrees of freedom
## Multiple R-squared:  0.3066,Adjusted R-squared:  0.2405
## F-statistic: 4.642 on 2 and 21 DF,  p-value: 0.0214
```

In order to understand the distinctions between type I and type III SS, lets begin with the `Anova` function from the **car** package. Initially, we obtain F values based on Type III SS. Notice that the F values, for both orders of IV entry, are identical to the squares of the t values from the tests of the regression coefficients seen above.

```
Anova(fit4_unbal.lm, type=3)
```

```
## Anova Table (Type III tests)
##
## Response: dv
##              Sum Sq Df  F value  Pr(>F)
```

```
## (Intercept) 12704.3  1 688.3348 < 2e-16 ***
## ac1           145.9  1   7.9047 0.01046 *
## ac2            17.3  1   0.9366 0.34418
## Residuals     387.6 21
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
Anova(fit5_unbal.lm, type=3)
```

```
## Anova Table (Type III tests)
##
## Response: dv
##              Sum Sq Df  F value  Pr(>F)
## (Intercept) 12704.3  1 688.3348 < 2e-16 ***
## ac2            17.3  1   0.9366 0.34418
## ac1           145.9  1   7.9047 0.01046 *
## Residuals     387.6 21
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Now we can switch the "type" specification to type I SS by using the `anova` function from base R.

```
anova(fit4_unbal.lm)
```

```
## Analysis of Variance Table
##
## Response: dv
##           Df Sum Sq Mean Sq F value   Pr(>F)
## ac1        1 154.08 154.083  8.3484 0.008774 **
## ac2        1  17.29  17.286  0.9366 0.344179
## Residuals 21 387.59  18.457
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
anova(fit5_unbal.lm)
```

```
## Analysis of Variance Table
##
## Response: dv
##           Df Sum Sq Mean Sq F value  Pr(>F)
## ac2        1  25.47  25.474  1.3802 0.25321
## ac1        1 145.89 145.895  7.9047 0.01046 *
## Residuals 21 387.59  18.457
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

The first thing to notice from these analyses is that the two F test are both different when the models with opposite entry orders of the vectors are compared.

For Type I SS, the effect of each IV is evaluated at the point at which it enters the equation, not adjusted for the presence of correlated IVs that are entered later.

A second thing to notice is that the F value for the second term to enter these two models match what was obtained by Type III SS just above, using `Anova`. This is because in this simple 2-IV model, the second vector entered is always adjusted for "all" other IVs, and this defines the Type III SS methodology: treat each vector as if it were the last to enter the equation.

A final comparison to make is to examine which F tests from the above four analyses match the F tests produced with the "split" approach used above, and repeated here for convenience. Notice here, that the F values match what we just examined above with the model where ac1 was entered first, but not the model where ac2 was ordered first. This implies that the "split" approach takes the first defined contrast as the first vector to enter the equation and the second, second, and so forth if there are additional vectors. We have reached the conclusion that the "split" approach utilizes TYPE I SS.

```
summary(fit3_unbal,
        split = list(factora = list(ac1 = 1, ac2 = 2)))
```

```
##                Df Sum Sq Mean Sq F value  Pr(>F)
## factora         2  171.4   85.68   4.642 0.02140 *
##   factora: ac1  1  154.1  154.08   8.348 0.00877 **
##   factora: ac2  1   17.3   17.29   0.937 0.34418
## Residuals      21  387.6   18.46
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Can we force the opposite ordering by a trick of what we call what on the "split" argument? No. Switching this does not change the outcome, it just changes to order of listing in the table.

```
summary(fit3_unbal,
        split = list(factora = list(ac1 = 2, ac2 = 1)))
```

```
##                Df Sum Sq Mean Sq F value  Pr(>F)
## factora         2  171.4   85.68   4.642 0.02140 *
##   factora: ac1  1   17.3   17.29   0.937 0.34418
##   factora: ac2  1  154.1  154.08   8.348 0.00877 **
## Residuals      21  387.6   18.46
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

In order to change the F values to the Type I SS seen with the opposite ordering model and using `aov`, we would have to switch the order of contrasts in the original definition.

```
contr_special2 <- matrix(c(0,1,-1,1,-.5,-.5), ncol=2)
contrasts(hays_unbal$factora) <- contr_special2
contrasts(hays_unbal$factora)
```

```
##         [,1] [,2]
## control    0  1.0
## fast       1 -0.5
## slow      -1 -0.5
```

Now, refitting the model should produce the second table of F values seen above with the `anova` function applied to the `lm` object.

```
fit3b_unbal.aov <- aov(dv~factora, data=hays_unbal)
summary(fit3b_unbal.aov,
        split = list(factora = list(ac1 = 1, ac2 = 2)))
```

```
##                Df Sum Sq Mean Sq F value Pr(>F)
## factora         2  171.4   85.68   4.642 0.0214 *
##   factora: ac1  1   25.5   25.47   1.380 0.2532
##   factora: ac2  1  145.9  145.89   7.905 0.0105 *
## Residuals      21  387.6   18.46
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

13.5 Commentary and Conclusions

This demonstration reflects many of the perspectives developed on Type I and III SS in the tutorial document on dong Basic Multiple Regression and Linear modeling. It reinforces the notion that the researcher must be careful in delineating why certain methods are preferred and understand the implications of varying choices among those methods. This document focused only on Type I and III SS distinctions because the Type II SS is only a relevant concept for factorial designs.

I have not seen any textbooks that emphasize the impact of non-orthogonality on contrast analysis in a one-way design. The exposition here may be unique. Textbooks most commonly focus on the impact of unequal N in factorial designs and examine the Type I, II and III distinctions in regard to lower order effects such as main effects. We will have to return to this conversation at that point.

A final point is the reiteration that the **emmeans** approach to executing analytical contrasts is employing a Type III SS approach for orthogonal sets. This is the most commonly used method for experimental designs and is an important aspect of the application of **emmeans** to larger/factorial designs.

Recommendation:

The common recommendation for use of Type III SS in factorial designs relates

to the fact that testing lower order effects with Type III SS is tantamount to testing null hypotheses about unweighted marginal means. This is a defensible approach for true experiments where the unequalness of sample sizes does not reflect population stratification. However, it oneway designs, this issue does not occur, except partially when groups are "combined" with contrats which pit their values against other groups. But there is a, perhaps, more important way of looking at the Type I/III distinction for contrasts in oneway designs. That is the idea that even within orthogonal sets, there are probably a-priori hypotheses that drive contrast choice for some but not all contrasts. The idea would be that the most important, a-priori, contrasts could be tested first by creating the design matrix with the where order of the vectors is determined by their a-priori importance. Then, a Type I SS approach might be more appropriate. Allow the most important hypotheses to absorb SS for the DV in a manner unadjusted by other less important contrasts. This kind of discussion is also absent in textbooks.

Chapter 14

Reproducibility

Version 1.4 Oct 16, 2020

Added a chapter on unequal sample size implications.

Updated some syntax styles and improved some wording.

Version 1.3 Oct 13, 2020

```
updated some syntax styles and emphasized using summary.lm on `aov` objects

Cleaned up some wording and layout issues.
```

Version 1.2 Sept. 16, 2020

updated for some R version 4 compatibility issues, including stringsAsFactors in read.csv.

Also edited some wording.

Version 1.1 April 9, 2020

converted to bookdown format, added multiple sections:

new graphs, contrast analysis details, nonparametrics, permutation, bootstrapping, etc.

Version 1.0 Feb 25, 2019

```
sessionInfo()
```

```
## R version 4.0.2 (2020-06-22)
## Platform: x86_64-w64-mingw32/x64 (64-bit)
## Running under: Windows 10 x64 (build 18363)
##
## Matrix products: default
##
```

```
## locale:
## [1] LC_COLLATE=English_United States.1252
## [2] LC_CTYPE=English_United States.1252
## [3] LC_MONETARY=English_United States.1252
## [4] LC_NUMERIC=C
## [5] LC_TIME=English_United States.1252
##
## attached base packages:
## [1] parallel  tcltk     stats     graphics  grDevices utils     datasets
## [8] methods   base
##
## other attached packages:
##  [1] dplyr_1.0.2               WRS2_1.1-0
##  [3] userfriendlyscience_0.7.2 sjstats_0.18.0
##  [5] sciplot_1.2-0             rcompanion_2.3.25
##  [7] pwr_1.3-0                 psych_2.0.7
##  [9] plyr_1.8.6                plotrix_3.7-8
## [11] pgirmess_1.6.9            outliers_0.14
## [13] nortest_1.0-4             mutoss_0.1-12
## [15] multtest_2.44.0           Biobase_2.48.0
## [17] BiocGenerics_0.34.0       multcomp_1.4-13
## [19] TH.data_1.0-10            MASS_7.3-53
## [21] mvtnorm_1.1-1             lsr_0.5
## [23] lmPerm_2.1.0              lmboot_0.0.1
## [25] lawstat_3.4               lattice_0.20-41
## [27] knitr_1.29                KScorrect_1.4.0
## [29] gridExtra_2.3             gt_0.2.2
## [31] granova_2.1               ggthemes_4.2.0
## [33] ggplot2_3.3.2             DTK_3.5
## [35] ez_4.4-0                  emmeans_1.5.0
## [37] dunn.test_1.3.5           coin_1.3-1
## [39] survival_3.2-3            car_3.0-9
## [41] carData_3.0-4             beeswarm_0.2.3
## [43] BayesFactor_0.9.12-4.2    coda_0.19-3
## [45] asbio_1.6-5               afex_0.27-2
## [47] lme4_1.1-23               Matrix_1.2-18
##
## loaded via a namespace (and not attached):
##   [1] tidyselect_1.1.0      htmlwidgets_1.5.1     grid_4.0.2
##   [4] combinat_0.0-8        maptools_1.0-1        munsell_0.5.0
##   [7] codetools_0.2-16      effectsize_0.3.2      units_0.6-7
##  [10] statmod_1.4.34        withr_2.2.0           colorspace_1.4-1
##  [13] rstudioapi_0.11       stats4_4.0.2          DescTools_0.99.37
##  [16] gbRd_0.4-11           labeling_0.3          Rdpack_1.0.0
##  [19] gWidgets2tcltk_1.0-6  splancs_2.01-40       mnormt_2.0.1
##  [22] farver_2.0.3          LearnBayes_2.15.1     vctrs_0.3.2
```

```
##  [25] generics_0.0.2       xfun_0.16             diptest_0.75-7
##  [28] R6_2.4.1             doParallel_1.0.15     Kendall_2.2
##  [31] reshape_0.8.8        scales_1.1.1          rgeos_0.5-3
##  [34] gtable_0.3.0         tkrplot_0.0-24        multcompView_0.1-8
##  [37] sandwich_2.5-1       rlang_0.4.7           MatrixModels_0.4-1
##  [40] EMT_1.1              scatterplot3d_0.3-41  splines_4.0.2
##  [43] rgdal_1.5-16         checkmate_2.0.0       broom_0.7.0.9000
##  [46] yaml_2.2.1           reshape2_1.4.4        abind_1.4-5
##  [49] modelr_0.1.8         backports_1.1.8       DiagrammeR_1.0.6.1
##  [52] tools_4.0.2          lavaan_0.6-7          bookdown_0.20
##  [55] spData_0.3.8         ellipsis_0.3.1        RColorBrewer_1.1-2
##  [58] raster_3.3-13        ggridges_0.5.2        Rcpp_1.0.5
##  [61] visNetwork_2.0.9     BiasedUrn_1.07        classInt_0.4-3
##  [64] purrr_0.3.4          deldir_0.1-28         viridis_0.5.1
##  [67] pbapply_1.4-3        deSolve_1.28          zoo_1.8-8
##  [70] ggrepel_0.8.2        haven_2.3.1           magrittr_1.5
##  [73] data.table_1.13.0    pixmap_0.4-11         openxlsx_4.1.5
##  [76] lmerTest_3.1-2       gmodels_2.18.1        lmtest_0.9-37
##  [79] tmvnsim_1.0-2        sjmisc_2.8.5          matrixStats_0.56.0
##  [82] hms_0.5.3            evaluate_0.14         xtable_1.8-4
##  [85] XML_3.99-0.5         rio_0.5.16            mclust_5.4.6
##  [88] readxl_1.3.1         compiler_4.0.2        tibble_3.0.3
##  [91] KernSmooth_2.23-17   crayon_1.3.4          minqa_1.2.4
##  [94] htmltools_0.5.0      mc2d_0.1-18           mgcv_1.8-32
##  [97] spdep_1.1-5          tidyr_1.1.1           libcoin_1.0-6
## [100] expm_0.999-5         Exact_2.0             SuppDists_1.1-9.5
## [103] MBESS_4.8.0          DBI_1.1.0             sjlabelled_1.1.6
## [106] gWidgets2_1.0-8      sf_0.9-5              boot_1.3-25
## [109] data.tree_1.0.0      GPArotation_2014.11-1 gdata_2.18.0
## [112] SCRT_1.3.1           insight_0.9.0         forcats_0.5.0
## [115] pkgconfig_2.0.3      numDeriv_2016.8-1.1   foreign_0.8-80
## [118] sp_1.4-2             foreach_1.5.0         pbivnorm_0.6.0
## [121] minpack.lm_1.2-1     estimability_1.3      bibtex_0.4.2.2
## [124] stringr_1.4.0        digest_0.6.25         parameters_0.8.2
## [127] rmarkdown_2.3        cellranger_1.1.0      curl_4.3
## [130] gtools_3.8.2         modeltools_0.2-23     nloptr_1.2.2.2
## [133] jsonlite_1.7.0       lifecycle_0.2.0       nlme_3.1-149
## [136] viridisLite_0.3.0    pillar_1.4.6          GGally_2.0.0
## [139] glue_1.4.1           bayestestR_0.7.2      zip_2.1.0
## [142] iterators_1.0.12     pander_0.6.3          ufs_0.3.1
## [145] class_7.3-17         stringi_1.4.6         performance_0.4.8
## [148] memoise_1.1.0        e1071_1.7-3
```

Bibliography

Aho, K. (2019). *asbio: A Collection of Statistical Tools for Biologists*. R package version 1.5-5.

Allaire, J., Xie, Y., McPherson, J., Luraschi, J., Ushey, K., Atkins, A., Wickham, H., Cheng, J., Chang, W., and Iannone, R. (2018). *rmarkdown: Dynamic Documents for R*. R package version 1.11.

Auguie, B. (2017). *gridExtra: Miscellaneous Functions for "Grid" Graphics*. R package version 2.3.

Champely, S. (2018). *pwr: Basic Functions for Power Analysis*. R package version 1.2-2.

Dinno, A. (2017). *dunn.test: Dunn's Test of Multiple Comparisons Using Rank Sums*. R package version 1.3.5.

Eklund, A. (2016). *beeswarm: The Bee Swarm Plot, an Alternative to Stripchart*. R package version 0.2.3.

Fox, J., Weisberg, S., and Price, B. (2018). *car: Companion to Applied Regression*. R package version 3.0-2.

Gastwirth, J. L., Gel, Y. R., Hui, W. L. W., Lyubchich, V., Miao, W., and Noguchi, K. (2017). *lawstat: Tools for Biostatistics, Public Policy, and Law*. R package version 3.2.

Giraudoux, P. (2018). *pgirmess: Spatial Analysis and Data Mining for Field Ecologists*. R package version 1.6.9.

Gross, J. and Ligges, U. (2015). *nortest: Tests for Normality*. R package version 1.0-4.

Hays, W. L. (1994). *Statistics*. Harcourt College Publishers, Fort Worth, 5th edition.

Hollander, M., Wolfe, D. A., and Chicken, E. (2013). *Nonparametric statistical methods*. John Wiley and Sons, Inc., Hoboken, New Jersey, 3rd edition.

Hothorn, T., Bretz, F., and Westfall, P. (2017a). *multcomp: Simultaneous Inference in General Parametric Models*. R package version 1.4-8.

Hothorn, T., Hornik, K., van de Wiel, M. A., Winell, H., and Zeileis, A. (2017b). *coin: Conditional Inference Procedures in a Permutation Test Framework*. R package version 1.2-2.

Iannone, R., Cheng, J., and Schloerke, B. (2019). *gt: Easily Create Presentation-Ready Display Tables*. R package version 0.1.0.

Komsta, L. (2011). *outliers: Tests for outliers*. R package version 0.14.

Lau, M. K. (2013). *DTK: Dunnett-Tukey-Kramer Pairwise Multiple Comparison Test Adjusted for Unequal Variances and Unequal Sample Sizes*. R package version 3.5.

Lawrence, M. A. (2016). *ez: Easy Analysis and Visualization of Factorial Experiments*. R package version 4.4-0.

Lemon, J., Bolker, B., Oom, S., Klein, E., Rowlingson, B., Wickham, H., Tyagi, A., Eterradossi, O., Grothendieck, G., Toews, M., Kane, J., Turner, R., Witthoft, C., Stander, J., Petzoldt, T., Duursma, R., Biancotto, E., Levy, O., Dutang, C., Solymos, P., Engelmann, R., Hecker, M., Steinbeck, F., Borchers, H., Singmann, H., Toal, T., Ogle, D., Baral, D., and Groemping, U. (2018). *plotrix: Various Plotting Functions*. R package version 3.7-4.

Lenth, R. (2019). *emmeans: Estimated Marginal Means, aka Least-Squares Means*. R package version 1.3.2.

Lüdecke, D. (2019). *sjstats: Collection of Convenient Functions for Common Statistical Computations*. R package version 0.17.3.

Mair, P. and Wilcox, R. (2018). *WRS2: A Collection of Robust Statistical Methods*. R package version 0.10-0.

Mangiafico, S. (2019). *rcompanion: Functions to Support Extension Education Program Evaluation*. R package version 2.0.10.

Maxwell, S. E., Delaney, H. D., and Kelley, K. (2017). *Designing experiments and analyzing data : a model comparison perspective*. Routledge, New York, NY, third edition / edition.

Morales, M., with code developed by the R Development Core Team, with general advice from the R-help listserv community, and especially Duncan Murdoch. (2017). *sciplot: Scientific Graphing Functions for Factorial Designs*. R package version 1.1-1.

Morey, R. D. and Rouder, J. N. (2018). *BayesFactor: Computation of Bayes Factors for Common Designs*. R package version 0.9.12-4.2.

Navarro, D. (2015). *lsr: Companion to "Learning Statistics with R"*. R package version 0.5.

Novack-Gottshall, P. and Wang, S. C. (2018). *KScorrect: Lilliefors-Corrected Kolmogorov-Smirnov Goodness-of-Fit Tests*. R package version 1.2.4.

Peters, G.-J. Y. (2017). Diamond plots: a tutorial to introduce a visualisation tool that facilitates interpretation and comparison of multiple sample estimates while respecting their inaccuracy. *PsyArXiv*.

Pollard, K. S., Gilbert, H. N., Ge, Y., Taylor, S., and Dudoit, S. (2018). *multtest: Resampling-based multiple hypothesis testing*. R package version 2.38.0.

Pruzek, R. M. and Helmreich, J. E. (2014). *granova: Graphical Analysis of Variance*. R package version 2.1.

R Core Team (2018). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria.

Revelle, W. (2019). *psych: Procedures for Psychological, Psychometric, and Personality Research*. R package version 1.8.12.

RStudio Team (2015). *RStudio: Integrated Development Environment for R*. RStudio, Inc., Boston, MA.

Sarkar, D. (2018). *lattice: Trellis Graphics for R*. R package version 0.20-38.

Singmann, H., Bolker, B., Westfall, J., and Aust, F. (2018). *afex: Analysis of Factorial Experiments*. R package version 0.22-1.

Team, M. C., Blanchard, G., Dickhaus, T., Hack, N., Konietschke, F., Rohmeyer, K., Rosenblatt, J., Scheer, M., and Werft, W. (2017). *mutoss: Unified Multiple Testing Procedures*. R package version 0.1-12.

Wheeler, B. and Torchiano, M. (2016). *lmPerm: Permutation Tests for Linear Models*. R package version 2.1.0.

Wickham, H. (2016). *plyr: Tools for Splitting, Applying and Combining Data*. R package version 1.8.4.

Wickham, H., Chang, W., Henry, L., Pedersen, T. L., Takahashi, K., Wilke, C., and Woo, K. (2018). *ggplot2: Create Elegant Data Visualisations Using the Grammar of Graphics*. R package version 3.1.0.

Wickham, H., François, R., Henry, L., and Müller, K. (2019). *dplyr: A Grammar of Data Manipulation*. R package version 0.8.0.1.

Wilcox, R. R. (2016). *Introduction to robust estimation and hypothesis testing*. Elsevier, Waltham, MA, 4th edition. edition.

Xie, Y. (2015). *Dynamic Documents with R and knitr*. Chapman and Hall/CRC, Boca Raton, Florida, 2nd edition. ISBN 978-1498716963.

Xie, Y. (2018a). *bookdown: Authoring Books and Technical Documents with R Markdown*. R package version 0.9.

Xie, Y. (2018b). *knitr: A General-Purpose Package for Dynamic Report Generation in R*. R package version 1.21.