Chapter 3

Two-Level Factorial Designs

3.1 Example 3.1, Table 3.5, p. 90

Read data and convert to long format.

```
#### 3.1
fn.data <- "http://statacumen.com/teach/RSM/data/RSM_EX_03-01.txt"
df.3.1 <- read.table(fn.data, header=TRUE)
str(df.3.1)
```

```
## 'data.frame': 8 obs. of  5 variables:
##  $ a : int  -1 1 -1 1 -1 1 -1 1
##  $ b : int  -1 -1 1 1 -1 -1 1 1
##  $ c : int  -1 -1 -1 -1 1 1 1 1
##  $ y1: int  247 470 429 435 837 551 775 660
##  $ y2: int  400 446 405 445 850 670 865 530
```

```
df.3.1
```

```
##    a  b  c  y1  y2
## 1 -1 -1 -1 247 400
## 2  1 -1 -1 470 446
## 3 -1  1 -1 429 405
## 4  1  1 -1 435 445
## 5 -1 -1  1 837 850
## 6  1 -1  1 551 670
## 7 -1  1  1 775 865
## 8  1  1  1 660 530
```

```
# reshape data into long format
library(reshape2)
df.3.1.long <- melt(df.3.1, id.vars = c("a", "b", "c")
                    , variable.name = "rep", value.name = "y")
df.3.1.long
```

```
##     a  b  c rep   y
## 1  -1 -1 -1  y1 247
## 2   1 -1 -1  y1 470
## 3  -1  1 -1  y1 429
## 4   1  1 -1  y1 435
## 5  -1 -1  1  y1 837
## 6   1 -1  1  y1 551
## 7  -1  1  1  y1 775
## 8   1  1  1  y1 660
## 9  -1 -1 -1  y2 400
## 10  1 -1 -1  y2 446
## 11 -1  1 -1  y2 405
## 12  1  1 -1  y2 445
## 13 -1 -1  1  y2 850
## 14  1 -1  1  y2 670
## 15 -1  1  1  y2 865
## 16  1  1  1  y2 530
```

Fit second-order linear model.

```
library(rsm)
rsm.3.1.y.SOabc <- rsm(y ~ SO(a, b, c), data = df.3.1.long)
```

```
## Warning:  Some coefficients are aliased - cannot use 'rsm' methods.
##  Returning an 'lm' object.
# externally Studentized residuals
rsm.3.1.y.SOabc$studres <- rstudent(rsm.3.1.y.SOabc)
summary(rsm.3.1.y.SOabc)
```

```
##
## Call:
## rsm(formula = y ~ SO(a, b, c), data = df.3.1.long)
##
## Residuals:
##    Min     1Q Median     3Q    Max
## -91.44 -27.47   5.69  27.72  79.94
##
## Coefficients: (3 not defined because of singularities)
##                 Estimate Std. Error t value Pr(>|t|)
## (Intercept)       563.44      15.72   35.85  5.1e-11 ***
## FO(a, b, c)a      -37.56      15.72   -2.39  0.04055 *
## FO(a, b, c)b        4.56      15.72    0.29  0.77815
## FO(a, b, c)c      153.81      15.72    9.79  4.3e-06 ***
## TWI(a, b, c)a:b   -12.94      15.72   -0.82  0.43165
## TWI(a, b, c)a:c   -76.94      15.72   -4.90  0.00085 ***
## TWI(a, b, c)b:c   -14.31      15.72   -0.91  0.38619
## PQ(a, b, c)a^2         NA         NA      NA       NA
## PQ(a, b, c)b^2         NA         NA      NA       NA
## PQ(a, b, c)c^2         NA         NA      NA       NA
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 62.9 on 9 degrees of freedom
## Multiple R-squared:  0.934,Adjusted R-squared:  0.89
## F-statistic: 21.2 on 6 and 9 DF,  p-value: 7.88e-05
```

Note that the quadratic terms can't be estimated because there are only two levels for each predictor variable.

Fit main-effect with three-way interaction linear model.

```
lm.3.1.y.3WIabc <- lm(y ~ (a + b + c)^3, data = df.3.1.long)
# externally Studentized residuals
lm.3.1.y.3WIabc$studres <- rstudent(lm.3.1.y.3WIabc)
summary(lm.3.1.y.3WIabc)
```

```
##
## Call:
## lm.default(formula = y ~ (a + b + c)^3, data = df.3.1.long)
##
## Residuals:
##    Min     1Q Median     3Q    Max
##  -76.5  -20.2    0.0   20.2   76.5
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)   563.44      15.81   35.64  4.2e-10 ***
```

```
## a             -37.56      15.81   -2.38   0.0448 *
## b               4.56      15.81    0.29   0.7802
## c             153.81      15.81    9.73  1.0e-05 ***
## a:b           -12.94      15.81   -0.82   0.4369
## a:c           -76.94      15.81   -4.87   0.0012 **
## b:c           -14.31      15.81   -0.91   0.3918
## a:b:c          14.94      15.81    0.94   0.3724
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 63.2 on 8 degrees of freedom
## Multiple R-squared:  0.94,Adjusted R-squared:  0.888
## F-statistic: 18.1 on 7 and 8 DF,  p-value: 0.00026
```

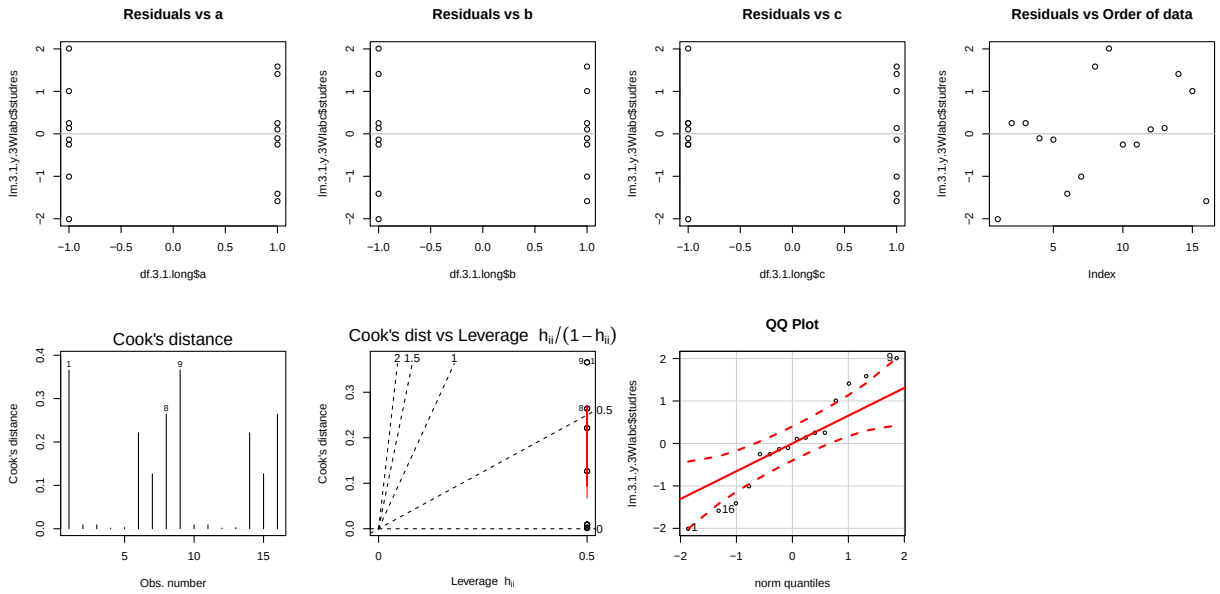
Here are the residual diagnostic plots, but we'll skip over them since our interest is in the interaction plots below.

```
# plot diagnistics
par(mfrow=c(2,4))

plot(df.3.1.long$a, lm.3.1.y.3WIabc$studres, main="Residuals vs a")
  # horizontal line at zero
  abline(h = 0, col = "gray75")

plot(df.3.1.long$b, lm.3.1.y.3WIabc$studres, main="Residuals vs b")
  # horizontal line at zero
  abline(h = 0, col = "gray75")

plot(df.3.1.long$c, lm.3.1.y.3WIabc$studres, main="Residuals vs c")
  # horizontal line at zero
  abline(h = 0, col = "gray75")

# residuals vs order of data
plot(lm.3.1.y.3WIabc$studres, main="Residuals vs Order of data")
  # horizontal line at zero
  abline(h = 0, col = "gray75")

plot(lm.3.1.y.3WIabc, which = c(4,6))

# Normality of Residuals
library(car)
qqPlot(lm.3.1.y.3WIabc$studres, las = 1, id.n = 3, main="QQ Plot")
```

```
##  1  9 16
##  1 16  2
```

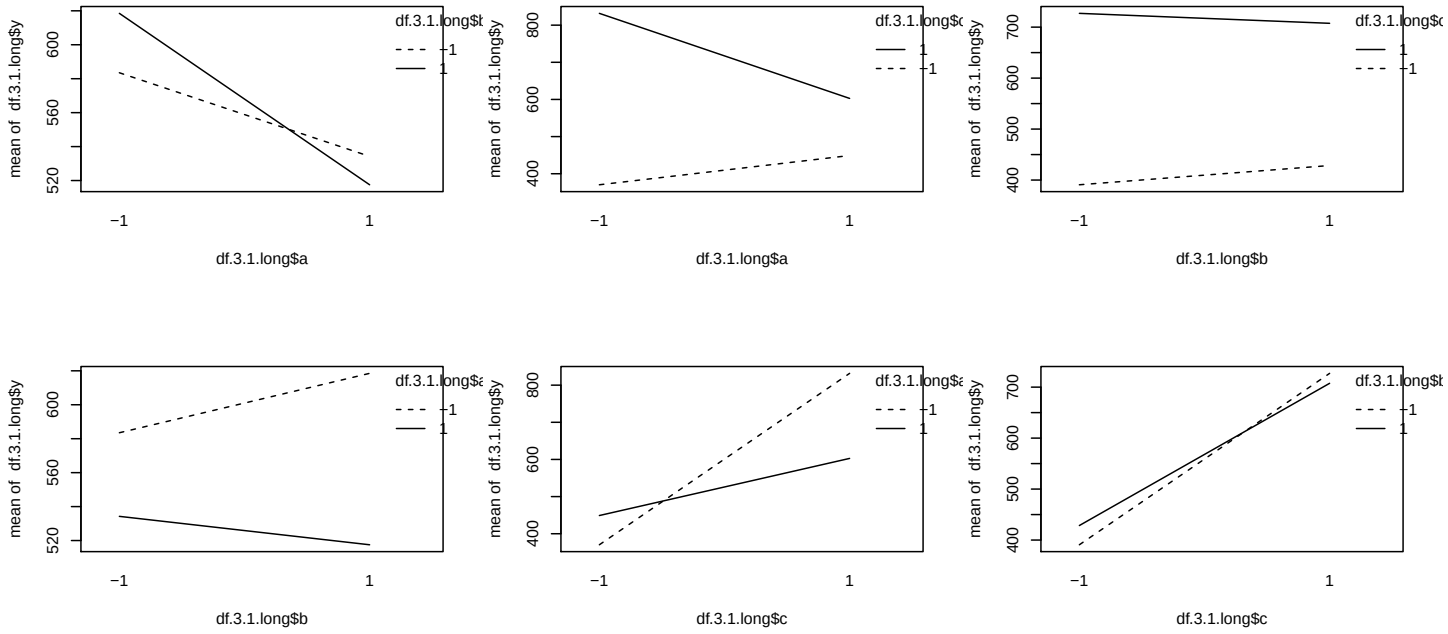
```
cooks.distance(lm.3.1.y.3WIabc)
```

```
##        1        2        3        4        5        6        7
## 0.365817 0.009001 0.009001 0.001563 0.002641 0.221297 0.126580
##        8        9       10       11       12       13       14
## 0.264100 0.365817 0.009001 0.009001 0.001563 0.002641 0.221297
##       15       16
## 0.126580 0.264100
```



Interaction plots, main effects vs main effects.

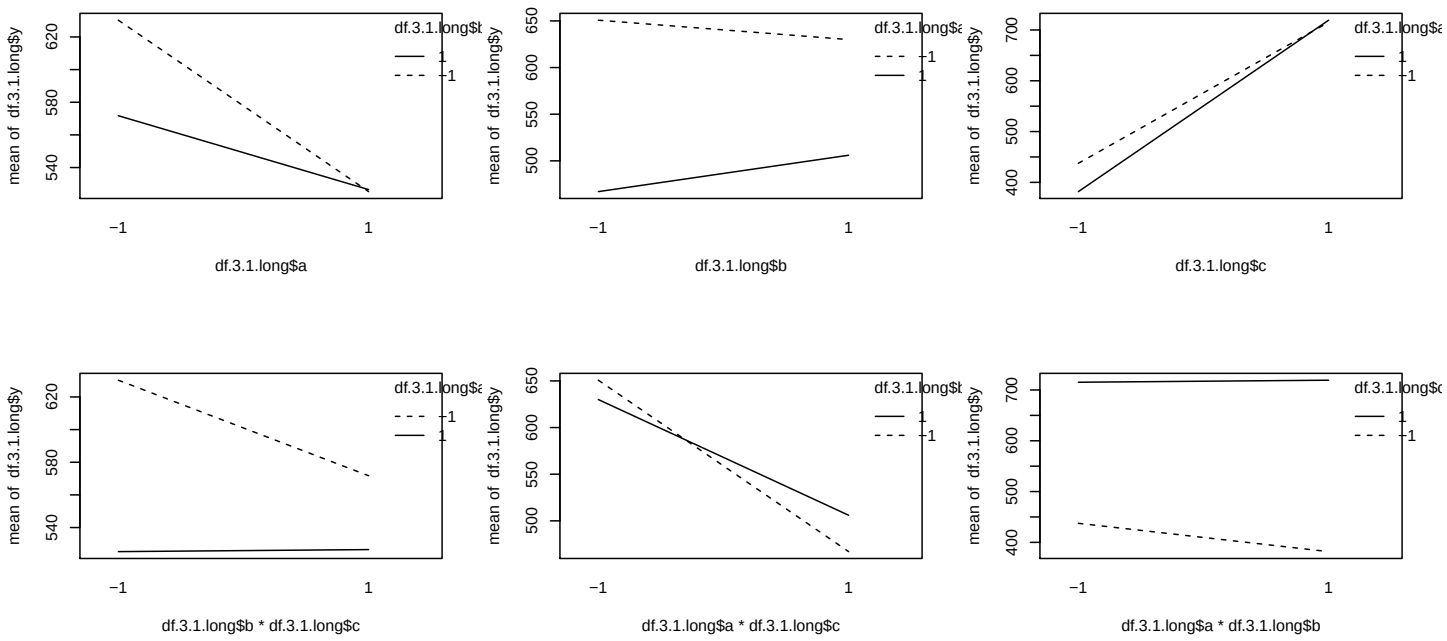
```
# interaction plot
par(mfcol=c(2,3))
interaction.plot(df.3.1.long$a, df.3.1.long$b, df.3.1.long$y)
interaction.plot(df.3.1.long$b, df.3.1.long$a, df.3.1.long$y)
interaction.plot(df.3.1.long$a, df.3.1.long$c, df.3.1.long$y)
interaction.plot(df.3.1.long$c, df.3.1.long$a, df.3.1.long$y)
interaction.plot(df.3.1.long$b, df.3.1.long$c, df.3.1.long$y)
interaction.plot(df.3.1.long$c, df.3.1.long$b, df.3.1.long$y)
```



Interaction plots, main effects vs two-way interactions.

```
# interaction plot
par(mfcol=c(2,3))
interaction.plot(df.3.1.long$a, df.3.1.long$b * df.3.1.long$c, df.3.1.long$y)
interaction.plot(df.3.1.long$b * df.3.1.long$c, df.3.1.long$a, df.3.1.long$y)
interaction.plot(df.3.1.long$b, df.3.1.long$a * df.3.1.long$c, df.3.1.long$y)
```

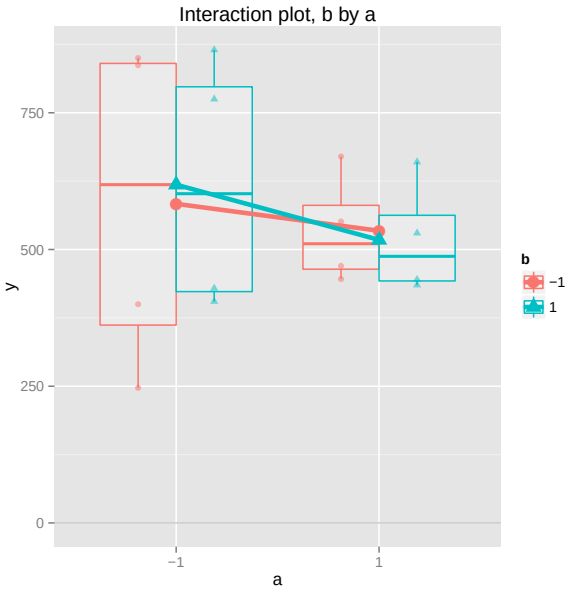
```
interaction.plot(df.3.1.long$a * df.3.1.long$c, df.3.1.long$b, df.3.1.long$y)
interaction.plot(df.3.1.long$c, df.3.1.long$a * df.3.1.long$b, df.3.1.long$y)
interaction.plot(df.3.1.long$a * df.3.1.long$b, df.3.1.long$c, df.3.1.long$y)
```

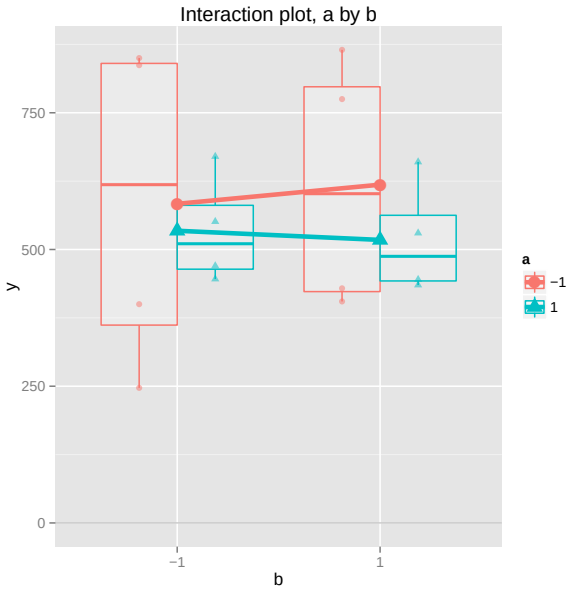


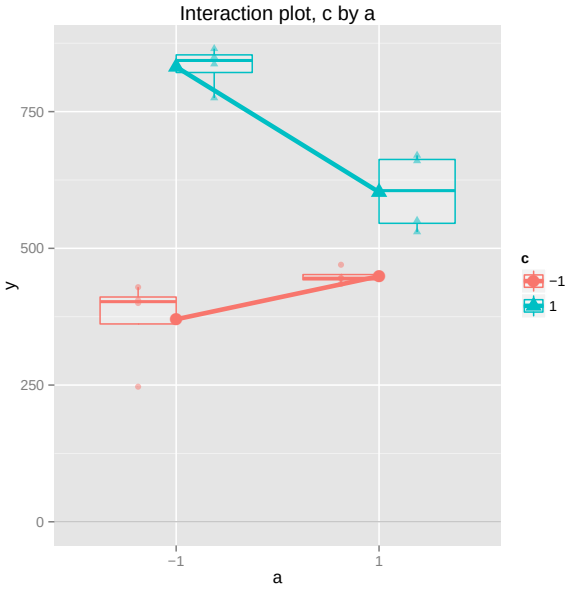
Interaction plots, main effects vs main effects in `ggplot()`.

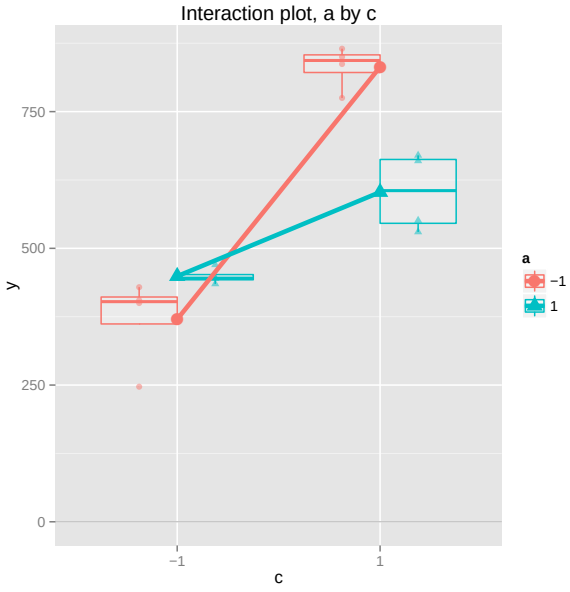
```
# Interaction plots, ggplot
library(plyr)
# Calculate the cell means for each (a, b) combination
# create factor version for ggplot categories
df.3.1.factor <- df.3.1.long
df.3.1.factor$a <- factor(df.3.1.factor$a)
df.3.1.factor$b <- factor(df.3.1.factor$b)
df.3.1.factor$c <- factor(df.3.1.factor$c)

#mean(df.3.1.factor[, "y"])
df.3.1.factor.mean    <- ddply(df.3.1.factor, .(), summarise, m = mean(y))
#df.3.1.factor.mean
df.3.1.factor.mean.a  <- ddply(df.3.1.factor, .(a), summarise, m = mean(y))
#df.3.1.factor.mean.a
df.3.1.factor.mean.b  <- ddply(df.3.1.factor, .(b), summarise, m = mean(y))
#df.3.1.factor.mean.b
df.3.1.factor.mean.c  <- ddply(df.3.1.factor, .(c), summarise, m = mean(y))
#df.3.1.factor.mean.c
df.3.1.factor.mean.ab <- ddply(df.3.1.factor, .(a,b), summarise, m = mean(y))
#df.3.1.factor.mean.ab
df.3.1.factor.mean.ac <- ddply(df.3.1.factor, .(a,c), summarise, m = mean(y))
#df.3.1.factor.mean.ac
df.3.1.factor.mean.bc <- ddply(df.3.1.factor, .(b,c), summarise, m = mean(y))
#df.3.1.factor.mean.bc
df.3.1.factor.mean.abc <- ddply(df.3.1.factor, .(a,b,c), summarise, m = mean(y))
#df.3.1.factor.mean.abc

library(ggplot2)
## (a, b)
p <- ggplot(df.3.1.factor, aes(x = a, y = y, colour = b, shape = b))
p <- p + geom_hline(aes(yintercept = 0), colour = "black"
                    , linetype = "solid", size = 0.2, alpha = 0.3)
p <- p + geom_boxplot(alpha = 0.25, outlier.size=0.1)
p <- p + geom_point(alpha = 0.5, position=position_dodge(width=0.75))
p <- p + geom_point(data = df.3.1.factor.mean.ab, aes(y = m), size = 4)
```

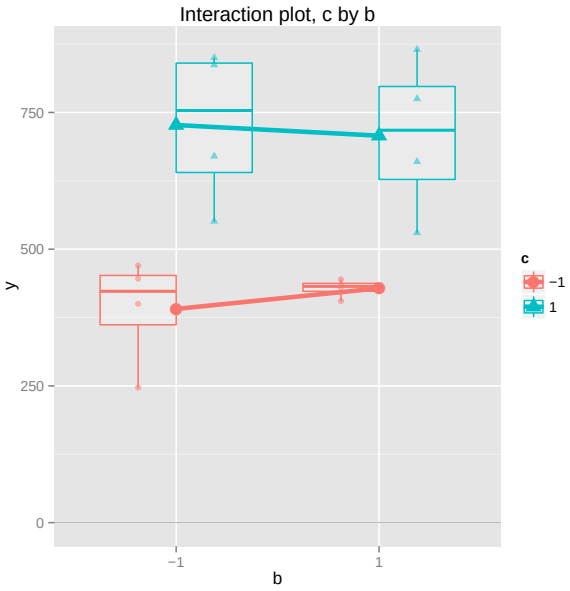
```
p <- p + geom_line(data = df.3.1.factor.mean.ab, aes(y = m, group = b), size = 1.5)
p <- p + labs(title = "Interaction plot, b by a")
print(p)
## ymax not defined:  adjusting position using y instead
p <- ggplot(df.3.1.factor, aes(x = b, y = y, colour = a, shape = a))
p <- p + geom_hline(aes(yintercept = 0), colour = "black"
                    , linetype = "solid", size = 0.2, alpha = 0.3)
p <- p + geom_boxplot(alpha = 0.25, outlier.size=0.1)
p <- p + geom_point(alpha = 0.5, position=position_dodge(width=0.75))
p <- p + geom_point(data = df.3.1.factor.mean.ab, aes(y = m), size = 4)
p <- p + geom_line(data = df.3.1.factor.mean.ab, aes(y = m, group = a), size = 1.5)
p <- p + labs(title = "Interaction plot, a by b")
print(p)
## ymax not defined:  adjusting position using y instead
## (a, c)
p <- ggplot(df.3.1.factor, aes(x = a, y = y, colour = c, shape = c))
p <- p + geom_hline(aes(yintercept = 0), colour = "black"
                    , linetype = "solid", size = 0.2, alpha = 0.3)
p <- p + geom_boxplot(alpha = 0.25, outlier.size=0.1)
p <- p + geom_point(alpha = 0.5, position=position_dodge(width=0.75))
p <- p + geom_point(data = df.3.1.factor.mean.ac, aes(y = m), size = 4)
p <- p + geom_line(data = df.3.1.factor.mean.ac, aes(y = m, group = c), size = 1.5)
p <- p + labs(title = "Interaction plot, c by a")
print(p)
## ymax not defined:  adjusting position using y instead
p <- ggplot(df.3.1.factor, aes(x = c, y = y, colour = a, shape = a))
p <- p + geom_hline(aes(yintercept = 0), colour = "black"
                    , linetype = "solid", size = 0.2, alpha = 0.3)
p <- p + geom_boxplot(alpha = 0.25, outlier.size=0.1)
p <- p + geom_point(alpha = 0.5, position=position_dodge(width=0.75))
p <- p + geom_point(data = df.3.1.factor.mean.ac, aes(y = m), size = 4)
p <- p + geom_line(data = df.3.1.factor.mean.ac, aes(y = m, group = a), size = 1.5)
p <- p + labs(title = "Interaction plot, a by c")
print(p)
## ymax not defined:  adjusting position using y instead
## (b, c)
p <- ggplot(df.3.1.factor, aes(x = b, y = y, colour = c, shape = c))
p <- p + geom_hline(aes(yintercept = 0), colour = "black"
                    , linetype = "solid", size = 0.2, alpha = 0.3)
p <- p + geom_boxplot(alpha = 0.25, outlier.size=0.1)
p <- p + geom_point(alpha = 0.5, position=position_dodge(width=0.75))
p <- p + geom_point(data = df.3.1.factor.mean.bc, aes(y = m), size = 4)
p <- p + geom_line(data = df.3.1.factor.mean.bc, aes(y = m, group = c), size = 1.5)
p <- p + labs(title = "Interaction plot, c by b")
print(p)
## ymax not defined:  adjusting position using y instead
p <- ggplot(df.3.1.factor, aes(x = c, y = y, colour = b, shape = b))
p <- p + geom_hline(aes(yintercept = 0), colour = "black"
                    , linetype = "solid", size = 0.2, alpha = 0.3)
p <- p + geom_boxplot(alpha = 0.25, outlier.size=0.1)
p <- p + geom_point(alpha = 0.5, position=position_dodge(width=0.75))
p <- p + geom_point(data = df.3.1.factor.mean.bc, aes(y = m), size = 4)
p <- p + geom_line(data = df.3.1.factor.mean.bc, aes(y = m, group = b), size = 1.5)
p <- p + labs(title = "Interaction plot, b by c")
print(p)
## ymax not defined:  adjusting position using y instead
```

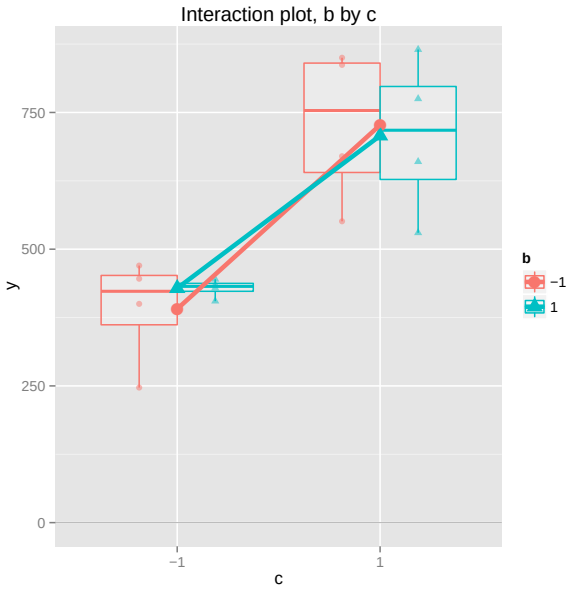

Interaction plot, b by a
y
750
500
250
0
−1
1
a
b
−1
1


Interaction plot, a by b
y
750
500
250
0
−1
1
b
a
−1
1


Interaction plot, c by a
y
750
500
250
0
−1
1
a
c
−1
1


Interaction plot, a by c
y
750
500
250
0
−1
1
c
a
−1
1


Interaction plot, c by b
y
750
500
250
0
−1
1
b
c
−1
1


Interaction plot, b by c
y
750
500
250
0
−1
1
c
b
−1
1

3.2 Example 3.2, Table 3.7, p. 97

Here's a quick way to create a design matrix for a factorial design.

```
design <- expand.grid("a" = c(-1, 1)
                      , "b" = c(-1, 1)
                      , "c" = c(-1, 1)
                      , "d" = c(-1, 1)
                      , "y" = NA)
design
```

```
##     a  b  c  d  y
## 1  -1 -1 -1 -1 NA
## 2   1 -1 -1 -1 NA
## 3  -1  1 -1 -1 NA
## 4   1  1 -1 -1 NA
## 5  -1 -1  1 -1 NA
## 6   1 -1  1 -1 NA
## 7  -1  1  1 -1 NA
## 8   1  1  1 -1 NA
## 9  -1 -1 -1  1 NA
## 10  1 -1 -1  1 NA
## 11 -1  1 -1  1 NA
## 12  1  1 -1  1 NA
## 13 -1 -1  1  1 NA
## 14  1 -1  1  1 NA
## 15 -1  1  1  1 NA
## 16  1  1  1  1 NA
```

Read data.

```
#### 3.2
fn.data <- "http://statacumen.com/teach/RSM/data/RSM_EX_03-02.txt"
df.3.2 <- read.table(fn.data, header=TRUE)
str(df.3.2)
```

```
## 'data.frame': 16 obs. of  5 variables:
##  $ a: int  -1 1 -1 1 -1 1 -1 1 -1 1 ...
##  $ b: int  -1 -1 1 1 -1 -1 1 1 -1 -1 ...
##  $ c: int  -1 -1 -1 -1 1 1 1 1 -1 -1 ...
##  $ d: int  -1 -1 -1 -1 -1 -1 -1 -1 1 1 ...
##  $ y: int  45 71 48 65 68 60 80 65 43 100 ...
```

```
df.3.2
```

```
##     a  b  c  d   y
## 1  -1 -1 -1 -1  45
## 2   1 -1 -1 -1  71
## 3  -1  1 -1 -1  48
## 4   1  1 -1 -1  65
## 5  -1 -1  1 -1  68
## 6   1 -1  1 -1  60
## 7  -1  1  1 -1  80
## 8   1  1  1 -1  65
## 9  -1 -1 -1  1  43
## 10  1 -1 -1  1 100
```

```
## 11 -1  1 -1  1  45
## 12  1  1 -1  1 104
## 13 -1 -1  1  1  75
## 14  1 -1  1  1  86
## 15 -1  1  1  1  70
## 16  1  1  1  1  96
```

Fit first-order with four-way interaction linear model.

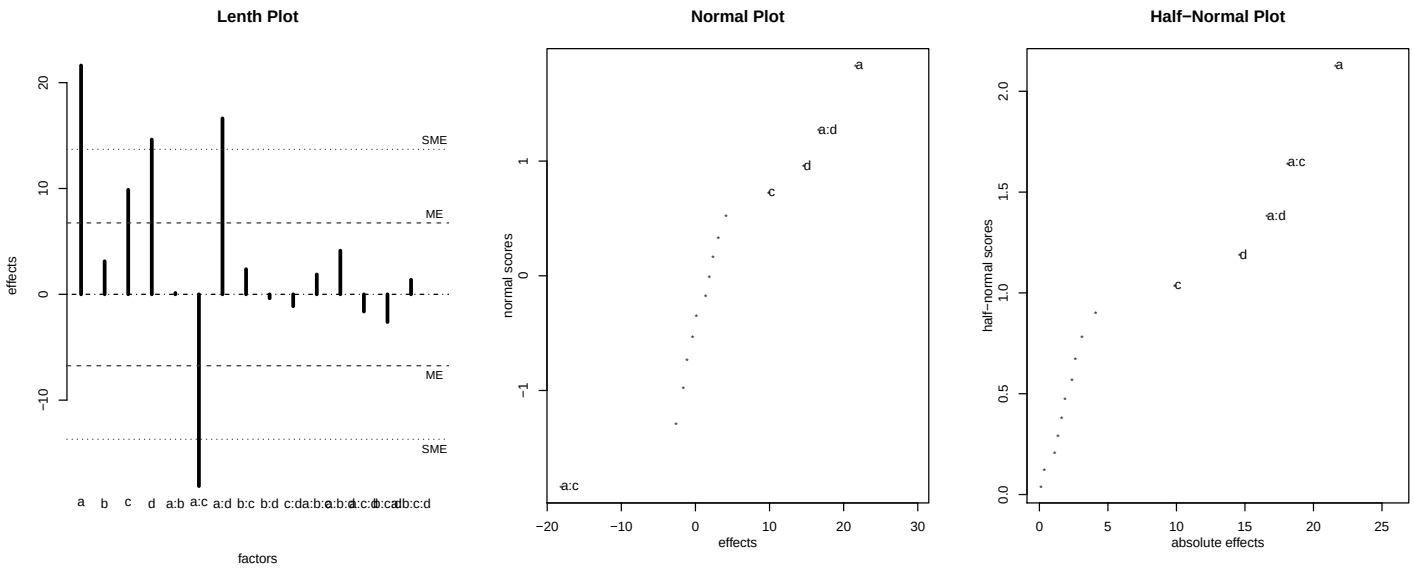
```
lm.3.2.y.4WIabcd <- lm(y ~ (a + b + c + d)^4, data = df.3.2)
## externally Studentized residuals
#lm.3.2.y.4WIabcd£studres <- rstudent(lm.3.2.y.4WIabcd)
#summary(lm.3.2.y.4WIabcd)
```

Use Lenth procedure and normal plots to assess effects in unreplicated design.

```
# BsMD package has unreplicated factorial tests (Daniel plots (aka normal), and Lenth)
library(BsMD)
par(mfrow=c(1,3))
LenthPlot(lm.3.2.y.4WIabcd, alpha = 0.05, main = "Lenth Plot") # , adj = 0.2
```

```
##   alpha    PSE     ME    SME
##   0.050  2.625  6.748 13.699
```

```
DanielPlot(lm.3.2.y.4WIabcd, main = "Normal Plot")
DanielPlot(lm.3.2.y.4WIabcd, half = TRUE, main = "Half-Normal Plot")
```



In example 3.2, all terms with B were not significant. So, project design to a 2^3 in factors A, C, and D (exploritory, not confirmatory). Now we have two replicates in each term (A, C, D), so now have error terms. So can do ANOVA, etc. Then, we can run additional confirmatory experiments.

Fit first-order with three-way interaction linear model.

```
lm.3.2.y.4WIacd <- lm(y ~ (a + c + d)^3, data = df.3.2)
# externally Studentized residuals
lm.3.2.y.4WIacd$studres <- rstudent(lm.3.2.y.4WIacd)
summary(lm.3.2.y.4WIacd)
```

```
##
## Call:
## lm.default(formula = y ~ (a + c + d)^3, data = df.3.2)
##
## Residuals:
##    Min     1Q Median     3Q    Max
##   -6.0   -2.5    0.0    2.5    6.0
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)   70.062      1.184   59.16  7.4e-12 ***
## a             10.812      1.184    9.13  1.7e-05 ***
## c              4.938      1.184    4.17  0.00312 **
## d              7.313      1.184    6.18  0.00027 ***
## a:c           -9.063      1.184   -7.65  6.0e-05 ***
## a:d            8.312      1.184    7.02  0.00011 ***
## c:d           -0.563      1.184   -0.48  0.64748
## a:c:d         -0.813      1.184   -0.69  0.51203
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 4.74 on 8 degrees of freedom
## Multiple R-squared:  0.969,Adjusted R-squared:  0.941
## F-statistic: 35.3 on 7 and 8 DF,  p-value: 2.12e-05
```

3.3 Creating Fractional Factorial designs with blocks and aliasing

The `rsm` package can generate blocked designs.

```
library(rsm)

# help for cube, see examples
?cube

# create the first block
block1 <- cube( basis = ~ x1 + x2 + x3 + x4
              , n0 = 0
              , blockgen = ~ c(x1 * x2 * x3, x1 * x3 * x4)
              , randomize = FALSE
              , bid = 1)
block1
```

```
##    run.order std.order x1.as.is x2.as.is x3.as.is x4.as.is
## 1          1         1       -1       -1       -1       -1
## 6          2         2        1       -1        1       -1
## 12         3         3        1        1       -1        1
## 15         4         4       -1        1        1        1
##
## Data are stored in coded form using these coding formulas ...
## x1 ~ x1.as.is
## x2 ~ x2.as.is
## x3 ~ x3.as.is
## x4 ~ x4.as.is
```

```
# populate a list of all the blocks and print them all
block <- list()
for (i.block in 1:4) {
  block[[i.block]] <- cube( basis = ~ x1 + x2 + x3 + x4
                          , n0 = 0
                          , blockgen = ~ c(x1 * x2 * x3, x1 * x3 * x4)
                          , randomize = FALSE
                          , bid = i.block)
}
block
```

```
## [[1]]
##    run.order std.order x1.as.is x2.as.is x3.as.is x4.as.is
## 1          1         1       -1       -1       -1       -1
## 6          2         2        1       -1        1       -1
## 12         3         3        1        1       -1        1
## 15         4         4       -1        1        1        1
##
## Data are stored in coded form using these coding formulas ...
## x1 ~ x1.as.is
## x2 ~ x2.as.is
## x3 ~ x3.as.is
## x4 ~ x4.as.is
```

```
##
## [[2]]
##    run.order std.order x1.as.is x2.as.is x3.as.is x4.as.is
## 3          1         1       -1        1       -1       -1
## 8          2         2        1        1        1       -1
## 10         3         3        1       -1       -1        1
## 13         4         4       -1       -1        1        1
##
## Data are stored in coded form using these coding formulas ...
## x1 ~ x1.as.is
## x2 ~ x2.as.is
## x3 ~ x3.as.is
## x4 ~ x4.as.is
##
## [[3]]
##    run.order std.order x1.as.is x2.as.is x3.as.is x4.as.is
## 4          1         1        1        1       -1       -1
## 7          2         2       -1        1        1       -1
## 9          3         3       -1       -1       -1        1
## 14         4         4        1       -1        1        1
##
## Data are stored in coded form using these coding formulas ...
## x1 ~ x1.as.is
## x2 ~ x2.as.is
## x3 ~ x3.as.is
## x4 ~ x4.as.is
##
## [[4]]
##    run.order std.order x1.as.is x2.as.is x3.as.is x4.as.is
## 2          1         1        1       -1       -1       -1
## 5          2         2       -1       -1        1       -1
## 11         3         3       -1        1       -1        1
## 16         4         4        1        1        1        1
##
## Data are stored in coded form using these coding formulas ...
## x1 ~ x1.as.is
## x2 ~ x2.as.is
## x3 ~ x3.as.is
## x4 ~ x4.as.is
```

Alternatively, package `FrF2` is the most comprehensive R package for creating fractional factorial 2-level designs, as well as Plackett-Burman type screening designs. Regular fractional factorials default to maximum resolution minimum aberration designs and can be customized in various ways. It can also show the alias structure for regular fractional factorials of 2-level factors.

```
library(FrF2)

k = 4  # number of factors, defaults to names = A, B, ...
```

```
FrF2( nruns = 2^k
    , nfactors = k
    , blocks = 4
    , alias.block.2fis=TRUE
    , randomize = FALSE
    )
```

```
##   run.no run.no.std.rp Blocks  A  B  C  D
## 1      1         1.1.1      1 -1 -1 -1 -1
## 2      2         4.1.2      1 -1 -1  1  1
## 3      3        14.1.3      1  1  1 -1  1
## 4      4        15.1.4      1  1  1  1 -1
##   run.no run.no.std.rp Blocks  A  B  C  D
## 5      5         5.2.1      2 -1  1 -1 -1
## 6      6         8.2.2      2 -1  1  1  1
## 7      7        10.2.3      2  1 -1 -1  1
## 8      8        11.2.4      2  1 -1  1 -1
##    run.no run.no.std.rp Blocks  A  B  C  D
## 9       9         6.3.1      3 -1  1 -1  1
## 10     10         7.3.2      3 -1  1  1 -1
## 11     11         9.3.3      3  1 -1 -1 -1
## 12     12        12.3.4      3  1 -1  1  1
##    run.no run.no.std.rp Blocks  A  B  C  D
## 13     13         2.4.1      4 -1 -1 -1  1
## 14     14         3.4.2      4 -1 -1  1 -1
## 15     15        13.4.3      4  1  1 -1 -1
## 16     16        16.4.4      4  1  1  1  1
## class=design, type= FrF2.blocked
## NOTE: columns run.no and run.no.std.rp are annotation, not part of the data frame
```