# Blocking and Fractional Factorial Designs

Henry Scharf

MATH/STAT 571B

# Module Goals:

**Ch. 7 [DAE]: Blocking and Confounding in the $2^k$ Factorial Design**

Students will be able to:

1. Explain what confounding means in the context of experimental design.
2. Explain how to construct a blocked factorial design with specified confounding variables.

**Ch. 8 [DAE]: Two-Level Fractional Factorial Designs**

Students will be able to:

1. Explain how fractional factorial designs are constructed through the specification of generators.
2. Construct specific examples of fractional factorial designs.
3. Explain how fractional factorial designs can be used in sequential/adaptive experimentation.
4. Explain what the resolution of a design is.

# Blocking a Replicated $2^k$ Factorial Design

# Blocking a Replicated $2^k$ Factorial Design

- What to do when there are nuisance factors? **Blocking.**
- Recall: Randomization can *protect against bias* introduced from unmeasured predictors, but it is *less efficient than blocking* for controllable variables.
- When each block can accommodate one or more complete factorial designs then implementation is fairly clear.
- When each block cannot accommodate a complete design, something else required.
- Upshot: With modest additional assumptions, we can (carefully) block combinations of factors in ways that allow us to account for the nuisance factors.

# Example: Coffee

- Henry and the other faculty are interested in determining what combination of location (Luce, Snakes and Lattes, Starbucks, Slot Canyon) and time of day (AM, PM) result in the best cup of coffee.
- He and the other four experimenters are aware that the taster should be treated as a nuisance factor and would like to use blocking to better detect the effects of the other factors.
- However, each taster can only drink 4 cups of coffee, no the full 8 required to carry out a full factorial design.
- How can they block by taster while still considering all possible combinations of the two primary variables?

# Confounding in the $2^k$ Factorial Design

# Confounding

- **Confounding**: when the effects of two different factors on a response are not jointly estimable.
- Confounding happens when two factors are perfectly linearly dependent.
  - When you know the level of one factor, you know exactly the level of the other (pairwise confounding).
  - In a sense, the opposite of orthogonality among predictors.
- **Confounding as a design technique**: a way to introduce blocking in factorial designs that controls which effects are confounded with the blocking variable.

# Confounding in the $2^k$ Factorial Design

- If we have to confound a factor with a blocking variable, it would be great if we could assume its effect was negligible.
- There are good reasons to suspect that higher-order interactions are more likely to be negligible than lower-order ones.

# Two Blocks

- If each block can accommodate half the full design, we can use as few as 2 blocks.
- 5 unknowns but only 4 observations implies something is not estimable.

$$y_i = \beta_0 + \beta_A x_A + \beta_B x_B + \beta_{AB} x_A x_B + \beta_{\text{block}} x_{\text{block}} + \epsilon_i$$

| **Treatment** | **Factorial Effects** | | | | **Blocking Designs** | |
|---|---|---|---|---|---|---|
| **Combination** | ***I*** | ***A*** | ***B*** | ***AB*** | ***Option (i)*** | ***Option (ii)*** |
| $(1)$ | $+$ | $-$ | $-$ | $+$ | 1 | 1 |
| $a$ | $+$ | $+$ | $-$ | $-$ | 1 | 2 |
| $b$ | $+$ | $-$ | $+$ | $-$ | 2 | 2 |
| $ab$ | $+$ | $+$ | $+$ | $+$ | 2 | 1 |

(1) What would a blocking design confounded with the effect of factor A be? How many unique balanced blocking designs are there for two blocks?

# Two Blocks

```
levels <- c("-", "+")
df <- expand.grid(A = levels, B = levels)
df$block <- factor(c(1, 2, 2, 1))
X <- model.matrix(~ A * B + block, data = df,
                  contrasts.arg = list(A = contr.helmert,
                                       B = contr.helmert,
                                       block = contr.helmert))
X
##   (Intercept) A1 B1 block1 A1:B1
## 1           1 -1 -1     -1     1
## 2           1  1 -1      1    -1
## 3           1 -1  1      1    -1
## 4           1  1  1     -1     1
```

# Two Blocks

- Sample correlation function reveals linear dependence (caution: these predictors are *not* random variables).

```
cor(X[, -1])
##        A1 B1 block1 A1:B1
## A1      1  0      0     0
## B1      0  1      0     0
## block1  0  0      1    -1
## A1:B1   0  0     -1     1
```

# Two Blocks

- Recall: For 0-1 coding, average effect of AB was $\beta_{AB} - \frac{1}{2}\beta_A - \frac{1}{2}\beta_B$.

```
levels <- c("-", "+")
df <- expand.grid(A = levels, B = levels)
df$block <- factor(c(1, 2, 2, 1))
X01 <- model.matrix(~ A * B + block, data = df)
cor(X01[, -1])
##                A+        B+     block2         A+:B+
## A+      1.0000000 0.0000000  0.0000000  0.5773503
## B+      0.0000000 1.0000000  0.0000000  0.5773503
## block2  0.0000000 0.0000000  1.0000000 -0.5773503
## A+:B+   0.5773503 0.5773503 -0.5773503  1.0000000
cor(X01[, 'A+:B+'] - 0.5 * X01[, 'A+'] - 0.5 * X01[, 'B+'], X01[, 'block2'])
## [1] -1
eigen(cor(X01[, -1]))$values
## [1] 2.000000e+00 1.000000e+00 1.000000e+00 2.220446e-15
```

- Same fundamental underlying problem: too many unknowns for the number of observations.

# Two Blocks

Table of Plus and Minus Signs for the $2^3$ Design

| Treatment Combination | Factorial Effect | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | $I$ | $A$ | $B$ | $AB$ | $C$ | $AC$ | $BC$ | $ABC$ | Block |
| (1) | + | − | − | + | − | + | + | − | 1 |
| $a$ | + | + | − | − | − | − | + | + | 2 |
| $b$ | + | − | + | − | − | + | − | + | 2 |
| $ab$ | + | + | + | + | − | − | − | − | 1 |
| $c$ | + | − | − | + | + | − | − | + | 2 |
| $ac$ | + | + | − | − | + | + | − | − | 1 |
| $bc$ | + | − | + | − | + | − | + | − | 1 |
| $abc$ | + | + | + | + | + | + | + | + | 2 |

- See DAE p.308–309 for notation/perspectives for defining blocks based on specified confounding.

# Two Blocks

- Generally a good idea to confound blocks with highest-order interaction terms.
- If possible, implement more blocks,
  - Confound different factors with block for each pair of blocks (DAE Ch. 7.8)

# More Than Two Blocks

## Four Blocks

- If we can fit 1/4 of the full factorial design, then we can use as few as 4 blocks.
- We will have $2^k + 3$ unknowns and $2^k$ observations.
- Confounding two factors with the block instead of one defines 4 blocks.
- Third confounded factor is defined implicitly by the choice of the first two.
- $\mathsf{L}_{\mathrm{ABC}} = (\mathbf{1}_{\{x_A=\mathrm{high}\}} + \mathbf{1}_{\{x_B=\mathrm{high}\}} + \mathbf{1}_{\{x_C=\mathrm{high}\}}) \mod 2.$

| **Treatment Combination** | **Factorial Effects** | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | ***I*** | ***A*** | ***B*** | ***C*** | $\boxed{\textit{\textbf{AB}}}$ | ***AC*** | ***BC*** | $\boxed{\textit{\textbf{ABC}}}$ | $\mathsf{L}_{\mathrm{ABC}}$ | $\mathsf{L}_{\mathrm{AB}}$ | ***Block*** |
| $(1)$ | $+$ | $-$ | $-$ | $-$ | $+$ | $+$ | $+$ | $-$ | 0 | 0 | 1 |
| $a$ | $+$ | $+$ | $-$ | $-$ | $-$ | $-$ | $+$ | $+$ | 1 | 1 | 4 |
| $b$ | $+$ | $-$ | $+$ | $-$ | $-$ | $+$ | $-$ | $+$ | 1 | 1 | 4 |
| $c$ | $+$ | $-$ | $-$ | $+$ | $+$ | $-$ | $-$ | $+$ | 1 | 0 | 2 |
| $ab$ | $+$ | $+$ | $+$ | $-$ | $+$ | $-$ | $-$ | $-$ | 0 | 0 | 1 |
| $ac$ | $+$ | $+$ | $-$ | $+$ | $-$ | $+$ | $-$ | $-$ | 0 | 1 | 3 |
| $bc$ | $+$ | $-$ | $+$ | $+$ | $-$ | $-$ | $+$ | $-$ | 0 | 1 | 3 |
| $abc$ | $+$ | $+$ | $+$ | $+$ | $+$ | $+$ | $+$ | $+$ | 1 | 0 | 2 |

## Four Blocks

| **Treatment Combination** | **Factorial Effects** | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | ***I*** | ***A*** | ***B*** | ***C*** | $\boxed{AB}$ | $\boxed{AC}$ | ***BC*** | ***ABC*** | $L_{\mathrm{AB}}$ | $L_{\mathrm{AC}}$ | ***Block*** |
| $(1)$ | + | − | − | − | + | + | + | − | 0 | 0 | 1 |
| $a$ | + | + | − | − | − | − | + | + | 1 | 1 | 4 |
| $b$ | + | − | + | − | − | + | − | + | 1 | 0 | 2 |
| $c$ | + | − | − | + | + | − | − | + | 0 | 1 | 3 |
| $ab$ | + | + | + | − | + | − | − | − | 0 | 1 | 3 |
| $ac$ | + | + | − | + | − | + | − | − | 1 | 0 | 2 |
| $bc$ | + | − | + | + | − | − | + | − | 1 | 1 | 4 |
| $abc$ | + | + | + | + | + | + | + | + | 0 | 0 | 1 |

(2) What is the third confounded factor? Which of these two designs do you think is a better choice? Why?

# Four Blocks

- Tables can get unwieldy for $k > 3$, so alternative notation/math required (see DAE).
- To determine implicitly defined confounded factors, use mod 2 arithmetic and specialized product operation:
  - $(ABC)(AB) = A^2B^2C = C$
  - $(A)(AB) = A^2B = B$

(3) $(AB)(BC) = ?$

- Be careful not to accidentally confound factors you care about with the blocks.

# $2^p$ Blocks

■ **TABLE 7.9**

**Suggested Blocking Arrangements for the $2^k$ Factorial Design**

| Number of Factors, $k$ | Number of Blocks, $2^p$ | Block Size, $2^{k-p}$ | Effects Chosen to Generate the Blocks | Interactions Confounded with Blocks |
|---|---|---|---|---|
| 3 | 2 | 4 | *ABC* | *ABC* |
| | 4 | 2 | *AB*, *AC* | *AB*, *AC*, *BC* |
| 4 | 2 | 8 | *ABCD* | *ABCD* |
| | 4 | 4 | *ABC*, *ACD* | *ABC*, *ACD*, *BD* |
| | 8 | 2 | *AB*, *BC*, *CD* | *AB*, *BC*, *CD*, *AC*, *BD*, *AD*, *ABCD* |
| 5 | 2 | 16 | *ABCDE* | *ABCDE* |
| | 4 | 8 | *ABC*, *CDE* | *ABC*, *CDE*, *ABDE* |
| | 8 | 4 | *ABE*, *BCE*, *CDE* | *ABE*, *BCE*, *CDE*, *AC*, *ABCD*, *BD*, *ADE* |
| | 16 | 2 | *AB*, *AC*, *CD*, *DE* | All two- and four-factor interactions (15 effects) |
| 6 | 2 | 32 | *ABCDEF* | *ABCDEF* |

- A $2^k$ factorial design split over $2^p$ blocks would have $2^k + (2^p - 1)$ unknown parameters in the full factorial model.
- $p$ generator confounders required for blocks of size $2^p$, which will implicitly define a total of $2^p - 1$ confounders.

## A Practical Point

- Suppose you would like to implement a $2^3$ factorial design for 3 factors (A, B, C).
- You are willing to assume an additive model.
- There is a nuisance variable you'd like to control for, but each block will only be able to accommodate 3 factor combinations.
- One possibility: Use 4 blocks, each of size 2, generated by AB & AC.
- Another possibility: Use 3 blocks, collapsing block 4 into 1 and 2.

| **Treatment Combination** | **Factorial Effects** | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | ***I*** | ***A*** | ***B*** | ***C*** | $\boxed{AB}$ | $\boxed{AC}$ | ***BC*** | ***ABC*** | ***4 Blocks*** | ***3 Blocks*** |
| $(1)$ | $+$ | $-$ | $-$ | $-$ | $+$ | $+$ | $+$ | $-$ | 1 | 1 |
| $a$ | $+$ | $+$ | $-$ | $-$ | $-$ | $-$ | $+$ | $+$ | 4 | 1 |
| $b$ | $+$ | $-$ | $+$ | $-$ | $-$ | $+$ | $-$ | $+$ | 2 | 2 |
| $c$ | $+$ | $-$ | $-$ | $+$ | $+$ | $-$ | $-$ | $+$ | 3 | 3 |
| $ab$ | $+$ | $+$ | $+$ | $-$ | $+$ | $-$ | $-$ | $-$ | 3 | 3 |
| $ac$ | $+$ | $+$ | $-$ | $+$ | $-$ | $+$ | $-$ | $-$ | 2 | 2 |
| $bc$ | $+$ | $-$ | $+$ | $+$ | $-$ | $-$ | $+$ | $-$ | 4 | 2 |
| $abc$ | $+$ | $+$ | $+$ | $+$ | $+$ | $+$ | $+$ | $+$ | 1 | 1 |

# A Practical Point

- 4 blocks are balanced, orthogonal to primary factors.
- 3 blocks involve 1 fewer parameter.

(4) Which design do you think has more power to detect a non-zero effect for A?

```
df_alt <- expand.grid(A = levels, B = levels, C = levels)
df_alt$block4 <- factor(c(1, 4, 2, 3, 3, 2, 4, 1))
df_alt$block3 <- factor(c(1, 1, 2, 3, 3, 2, 2, 1))
X3 <- model.matrix(~ A * B * C + block3, data = df_alt)
X4 <- model.matrix(~ A * B * C + block4, data = df_alt)
beta <- c(0, 2, 0, 0, 0, 0, 0, 0, 0, 0)
names(beta) <- colnames(X3)
p_values <- rowMeans(sapply(1:500, function(rep){
  y <- X3 %*% beta + rnorm(nrow(X3))
  c("3 Blocks" = anova(aov(y ~ A + B + C + block3, data = df_alt))['A', 'Pr(>F)'],
    "4 Blocks" = anova(aov(y ~ A + B + C + block4, data = df_alt))['A', 'Pr(>F)'])
}))
```

# A Practical Point

```
p_values
##  3 Blocks  4 Blocks
## 0.1262140 0.1949921
```

# Two-Level Fractional Factorial Designs

# Two-Level Fractional Factorial Designs

- When the number of combinations in a factorial design is large, it can be costly/challenging to implement the entire design.
- If we are willing to assume some of effects are negligible, then we can use a subset of factor combinations for inference.
- Motivation is very similar to blocking: Specify confounding such that effects of interest are confounded with effects that we can assume are 0.
- Smaller designs can be implemented sequentially/adaptively.

# One-Half Fraction of $2^k$: $2^{k-1}$ Fractional Factorial

- Defining relation between $I$ and generator: E.g., $I = ABC$
- Aliases:
  - $A * I = A * (ABC) \implies A = BC$
  - $B = AC$
  - $C = AB$
- Generator with opposite sign defines complementary design from same **family**

| A | B | C |
|---|---|---|
| $-$ | $-$ | $+$ |
| $+$ | $-$ | $-$ |
| $-$ | $+$ | $-$ |
| $+$ | $+$ | $+$ |

| A | B | C |
|---|---|---|
| $-$ | $-$ | $-$ |
| $+$ | $-$ | $+$ |
| $-$ | $+$ | $+$ |
| $+$ | $+$ | $-$ |

# One-Quarter Fraction of $2^k$: $2^{k-2}$ Fractional Factorial

- Two generators now required, which implicitly define three relationships.
- E.g., $I = ABCE = BCDF \implies I = AB^2C^2DEF = ADEF$
- Analogous to using four blocks with a factorial design, be careful about implied aliases.
- Choosing highest-order effects can backfire: E.g., $I = ABCD = ABC \implies I = D$
- Other members of fractional factorial family are defined by signs on relationships.
  - $I = P = Q$
  - $I = -P = Q$
  - $I = P = -Q$
  - $I = -P = -Q$
- Combine any two family members to get a $2^{k-1}$ design based on shared generator.
  - $I = P = Q$ and $I = P = -Q$ is equivalent to $I = P$.

# Example: Cooking for Neko

# Example: Cooking for Neko

- Neko is my 5 year old kid. As her parents, her mom and I are always trying to get her to eat more vegetables.
- She likes pizza. So, I'm interested in figuring out a way to maximize her vegetable intake through the medium of pizza.
- Four factors:
  - (A) Pepperoni: yes or no
  - (B) Vegetable: tomatoes or spinach
  - (C) Cheese: yes or no
  - (D) Sauce: red or pesto
- Response: Mass of pizza eaten (grams)
- Constraint: I can only cook 4 small pizzas a night.

# Example: Cooking for Neko

- Neko is my 5 year old kid. As her parents, her mom and I are always trying to get her to eat more vegetables.
- She likes pizza. So, I'm interested in figuring out a way to maximize her vegetable intake through the medium of pizza.
- Four factors:
  - (A) Pepperoni: yes or no
  - (B) Vegetable: tomatoes or spinach
  - (C) Cheese: yes or no
  - (D) Sauce: red or pesto
- Response: Mass of pizza eaten (grams)
- Constraint: I can only cook 4 small pizzas a night.

# Example: Cooking for Neko

- I can cook 1/4 of a full factorial design in one evening.
- Plan: Use a $2^{2-2}$ design on week 1 and record grams of pizza consumed for each combination (random order of course).
- To define design, need two (and implied third) generators.
  - $I = ABC = ABD \implies I = A^2B^2CD = CD$

# Example: Cooking for Neko

| A | B | C | D |
|---|---|---|---|
| $-$ | $-$ | $+$ | $+$ |
| $+$ | $-$ | $-$ | $-$ |
| $-$ | $+$ | $-$ | $-$ |
| $+$ | $+$ | $+$ | $+$ |

```r
df_1
##   pepperoni     veg cheese sauce     grams
## 1        no  tomato    yes pesto  57.48285
## 2       yes  tomato     no   red 152.03073
## 3        no spinach     no   red  53.38043
## 4       yes spinach    yes pesto 152.77834
```

```r
fit_1 <- aov(grams ~ pepperoni + veg + cheese + sauce, data = df_1)
summary(lm(fit_1))
## ALL 4 residuals are 0: no residual degrees of freedom!
##
## Coefficients: (1 not defined because of singularities)
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept)    55.058        NaN     NaN      NaN
## pepperoniyes   96.973        NaN     NaN      NaN
## vegspinach     -1.677        NaN     NaN      NaN
## cheeseyes       2.425        NaN     NaN      NaN
## saucepesto         NA         NA      NA       NA
```

# Example: Cooking for Neko

| A | B | C | D |
|---|---|---|---|
| − | − | + | + |
| + | − | − | − |
| − | + | − | − |
| + | + | + | + |

```
anova(fit_1)
## Warning in anova.lm(fit_1): ANOVA F-tests on an essentially perfe
## unreliable
## Analysis of Variance Table
##
## Response: grams
##           Df Sum Sq Mean Sq F value Pr(>F)
## pepperoni  1 9403.7  9403.7     NaN    NaN
## veg        1    2.8     2.8     NaN    NaN
## cheese     1    5.9     5.9     NaN    NaN
## Residuals  0    0.0     NaN
```

# Example: Cooking for Neko

- For week 2, I decide to complete one of the 1/2 fractional designs.

| A | B | C | D |
|---|---|---|---|
| $-$ | $-$ | $-$ | $+$ |
| $+$ | $-$ | $+$ | $-$ |
| $-$ | $+$ | $+$ | $-$ |
| $+$ | $+$ | $-$ | $+$ |

```
df_2
##   pepperoni     veg cheese sauce      grams
## 1        no  tomato     no pesto  -3.373131
## 2       yes  tomato    yes   red 194.385323
## 3        no spinach    yes   red  93.019707
## 4       yes spinach     no pesto 132.033769
```

```
fit_2 <- aov(grams ~ pepperoni + veg + cheese + sauce, data = df_2)
summary(lm(fit_2))
## Coefficients: (1 not defined because of singularities)
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept)    -3.373        NaN     NaN      NaN
## pepperoniyes  118.386        NaN     NaN      NaN
## vegspinach     17.021        NaN     NaN      NaN
## cheeseyes      79.372        NaN     NaN      NaN
## saucepesto         NA         NA      NA       NA
```

# Example: Cooking for Neko

```
df_12 <- cbind(rbind(df_1, df_2), week = rep(1:2, rep(4, 2)))
head(df_12, 2)
##   pepperoni    veg cheese sauce     grams week
## 1        no tomato    yes pesto  57.48285    1
## 2       yes tomato     no   red 152.03073    1
```

```
fit_12 <- aov(grams ~ pepperoni * veg * cheese * sauce, data = df_12)
summary(lm(fit_12))
## Coefficients: (8 not defined because of singularities)
##                                             Estimate Std. Error t value
## (Intercept)                                  35.0021        NaN     NaN
## pepperoniyes                                117.0286        NaN     NaN
## vegspinach                                   18.3783        NaN     NaN
## cheeseyes                                    61.0526        NaN     NaN
## saucepesto                                  -38.3753        NaN     NaN
## pepperoniyes:vegspinach                           NA         NA      NA
## pepperoniyes:cheeseyes                      -18.6980        NaN     NaN
## vegspinach:cheeseyes                        -21.4134        NaN     NaN
## pepperoniyes:saucepesto                           NA         NA      NA
## vegspinach:saucepesto                             NA         NA      NA
## cheeseyes:saucepesto                         -0.1967        NaN     NaN
```

# Example: Cooking for Neko

- After two weeks, do I know enough to predict which combination of factors will result in the most amount eaten?

```
fit_add_block <- aov(grams ~ pepperoni + veg + cheese + sauce + as.factor(week),
                     data = df_12)
summary(lm(fit_add_block))
## Coefficients:
##                    Estimate Std. Error t value Pr(>|t|)
## (Intercept)        45.02998   12.30967   3.658   0.0673 .
## pepperoniyes      107.67958   10.05080  10.714   0.0086 **
## vegspinach          7.67162   10.05080   0.763   0.5250
## cheeseyes          40.89861   10.05080   4.069   0.0554 .
## saucepesto        -38.47359   10.05080  -3.828   0.0620 .
## as.factor(week)2    0.09833   10.05080   0.010   0.9931
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 14.21 on 2 degrees of freedom
```

# Example: Cooking for Neko

(5) Based on an additive model, what combination of factors is best for getting Neko to eat?

```
fit_add <- aov(grams ~ pepperoni + veg + cheese + sauce, data = df_12)
summary(lm(fit_add))
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept)    45.079      9.175   4.913 0.016149 *
## pepperoniyes  107.680      8.207  13.121 0.000956 ***
## vegspinach      7.672      8.207   0.935 0.418847
## cheeseyes      40.899      8.207   4.984 0.015532 *
## saucepesto    -38.474      8.207  -4.688 0.018346 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 11.61 on 3 degrees of freedom
```

# Example: Cooking for Neko

- Confirmation experiment: Did the model predict the result well?

```
predict(fit_add, newdata = data.frame(pepperoni = "yes", veg = "spinach",
                                       cheese = "yes", sauce = "red"),
        interval = "prediction")
##        fit      lwr      upr
## 1 201.3289 154.2455 248.4124
df_confirm
##   pepperoni     veg cheese sauce    grams week
## 8       yes spinach    yes   red 133.0945    3
```

# Example: Cooking for Neko

- Something's not quite right with the additive model.
- Try including some second-order interaction terms.

```r
df_123 <- rbind(df_12, df_confirm)
fit_int <- aov(grams ~ pepperoni * veg + pepperoni * cheese + pepperoni * sauce +
                 veg * cheese + veg * sauce + cheese * sauce, data = df_123)
summary(lm(fit_int))
## Coefficients: (2 not defined because of singularities)
##                           Estimate Std. Error t value Pr(>|t|)
## (Intercept)               -23.2537        NaN     NaN      NaN
## pepperoniyes              175.2844        NaN     NaN      NaN
## vegspinach                 76.6341        NaN     NaN      NaN
## cheeseyes                  61.0526        NaN     NaN      NaN
## saucepesto                 19.8805        NaN     NaN      NaN
## pepperoniyes:vegspinach  -116.5116        NaN     NaN      NaN
## pepperoniyes:cheeseyes    -18.6980        NaN     NaN      NaN
## pepperoniyes:saucepesto         NA         NA      NA       NA
## vegspinach:cheeseyes      -21.4134        NaN     NaN      NaN
## vegspinach:saucepesto           NA         NA      NA       NA
## cheeseyes:saucepesto       -0.1967        NaN     NaN      NaN
```

## Example: Cooking for Neko

- I decide to implement another 1/4 fraction, now based on $I = ABC = -ABD = -CD$.

(6) What factor combinations are in this design? What are the aliases for $I = ABC$?

# Example: Cooking for Neko

| A | B | C | D |
|---|---|---|---|
| $-$ | $-$ | $+$ | $-$ |
| $+$ | $-$ | $-$ | $+$ |
| $-$ | $+$ | $-$ | $+$ |
| $+$ | $+$ | $+$ | $-$ |

- I already did row 4 in my confirmation experiment.

```
df_4
##    pepperoni     veg cheese sauce     grams week
## 5         no  tomato    yes   red  82.37148    4
## 10       yes  tomato     no pesto 108.17973    4
## 11        no spinach     no pesto  49.92112    4
```

# Example: Cooking for Neko

- I have a hunch that maybe (B) Vegetable and (D) Sauce might also interact.

```
df_1234 <- rbind(df_123, df_4)
fit_1234 <- aov(grams ~ pepperoni * veg + cheese + veg * sauce, data = df_1234)
summary(lm(fit_1234))
## -10.8635  -3.2372  -3.2372 -10.8635
##
## Coefficients:
##                           Estimate Std. Error t value Pr(>|t|)
## (Intercept)                 44.710     13.573   3.294 0.021619 *
## pepperoniyes               106.430     11.755   9.054 0.000275 ***
## vegspinach                   2.609     13.814   0.189 0.857625
## cheeseyes                   40.899      8.886   4.603 0.005826 **
## saucepesto                 -39.723     11.755  -3.379 0.019689 *
```

# Example: Cooking for Neko

- One more confirmation experiment: Pepperoni, Tomato, Cheese, Red Sauce
- Already looked at this combination on Week 2.

```
predict(fit_1234, newdata = data.frame(pepperoni = "yes", veg = "tomato",
                                       cheese = "yes", sauce = "red"),
        interval = "prediction")
##        fit      lwr      upr
## 1 192.0387 151.9299 232.1475
df_2
##   pepperoni     veg cheese sauce      grams
## 1        no  tomato     no pesto  -3.373131
## 2       yes  tomato    yes   red 194.385323
## 3        no spinach    yes   red  93.019707
## 4       yes spinach     no pesto 132.033769
```

# Example: Cooking for Neko

```
fit_all <- aov(grams ~ pepperoni * veg * cheese * sauce, data = df_1234)
summary(lm(fit_all))
## Coefficients: (4 not defined because of singularities)
##                                          Estimate Std. Error t value
## (Intercept)                               42.5683        NaN     NaN
## pepperoniyes                             109.4625        NaN     NaN
## vegspinach                                10.8122        NaN     NaN
## cheeseyes                                 39.8032        NaN     NaN
## saucepesto                               -45.9414        NaN     NaN
## pepperoniyes:vegspinach                  -29.4402        NaN     NaN
## pepperoniyes:cheeseyes                     2.5514        NaN     NaN
## vegspinach:cheeseyes                      -0.1639        NaN     NaN
## pepperoniyes:saucepesto                    2.0904        NaN     NaN
## vegspinach:saucepesto                     42.4821        NaN     NaN
## cheeseyes:saucepesto                      21.0528        NaN     NaN
## pepperoniyes:vegspinach:cheeseyes        -42.4989        NaN     NaN
## pepperoniyes:vegspinach:saucepesto             NA         NA      NA
## pepperoniyes:cheeseyes:saucepesto              NA         NA      NA
## vegspinach:cheeseyes:saucepesto                NA         NA      NA
## pepperoniyes:vegspinach:cheeseyes:saucepesto   NA         NA      NA
```