ANOVA Screens

Neil J. Hatfield

Mar 14, 2024

The focus of this guide will be on how we can use R to visualize pass our through a series of screens for the purpose of ANOVA. The main tool that we'll leverage is a function that developed that may be applied to One-way ANOVA situations as well as One-Way + Block ANOVA situations. At the time of writing, my function only works for these two types of ANOVA models.

Getting Started

We will need two things: 1) load all necessary tools and 2) have some data to work with.

Loading Our Tools

The notion of ANOVA screens is not something that is built into R (or any other statistical software system). However, we can easily expand out R's tools through packages and custom functions.

For this guide, we will need to draw upon the following packages: `{tidyverse}`, `{hasseDiagram}`, `{openxlsx}`, `{knitr}`, and `{kableExtra}`. We can load them into our current session of R using the following commands. If you don't have these packages installed, you will get an error message along the lines of "there is no package called […]".

> **Installing Packages**
>
> Remember, I wrote a script that can check your R set up and install all the packages we use in this class for you. I highly recommend that run this script, even if you are an experienced R user. All you need to do is run the following commands in your R console:
>
> ```
> source("https://raw.github.com/neilhatfield/STAT461/master/rScripts/checkSetup.R")
> checkSetup()
> ```

```
# Load Packages ----
packages <- c("tidyverse", "hasseDiagram", "openxlsx", "knitr", "kableExtra")   ①
lapply(                                                                         ②
  X = packages,
  FUN = library,
  character.only = TRUE,
  quietly = TRUE
)
```

① This is the list of packages I want to load.

② The `lapply` function lets me repeatedly apply the `library` function–which actually loads the package whose name is listed.

Once you have loaded your packages, a next step would be to load any additional tools. Since I have yet to create a package containing all of the tools I've built for Stat 461, we'll need to load them by using the `source` function as shown below.

```
# Load extra tools ----
source("https://raw.github.com/neilhatfield/STAT461/main/rScripts/ANOVATools.R")
```

The `anovaScreens` function that we'll use is part of the suite of tools I've created. There are multiple other tools that are part of this suite that we'll use at other points during the semester.

> **ANOVA Screens in JMP**
>
> Several years ago I wrote a tool for use in the JMP point-and-click statistical software (free to PSU students; available on PSU lab machines). You'll need to download and install my StatsTools JMP Addin. This addin will then let you get the screen results in that software similar to what you'll see in this guide.

Getting Our Data

In order to pass data through a set of screens, we need to have data in our R environment. There are multiple ways in which we can load data into R. For additional examples, please check out the Getting Started guides in Canvas (specifically, the Stat461.GettingStarted.pdf).

Many of the data sets that we'll be working with in Stat 461 will be accessible online either as CSV, DAT, TXT, or XLSX files. Provided you have internet access, R can directly read online data files into your current R session/environment; you don't need to download the data to your computer first. While there is a certain attraction to this method (allows for quick data file updates), there are risks (e.g., what happens when you don't have internet access?). The methods that I'll use in the guides work equally well with a file located online or on your computer–the only change you'll need to make is the path to the file.

With the exception of XLSX files (and extremely large data files), my go-to function for reading data into R is `read.table` from base R. For XLSX files, I like to use the `readWorkbook` function from the `{openxlsx}` package. The following code demonstrates reading in various data files.

```
# Load Paper Airplane Data ----
centralPath <- "https://raw.github.com/neilhatfield/STAT461/main/dataFiles/"
## Section 2 ----
plane2 <- readWorkbook(
  xlsxFile = paste0(centralPath, "PaperAirplanes_Sp24_Sec2.xlsx")
)
## Section 3 ----
plane3 <- readWorkbook(
  xlsxFile = paste0(centralPath, "PaperAirplanes_Sp24_Sec3.xlsx")
)

# Load Barley Data ----
## Example for the One-way ANOVA + Block Section
barleyData <- read.table(
  file = "https://raw.github.com/neilhatfield/STAT461/master/dataFiles/barley.dat",
  header = TRUE,
  sep = ","
)
```

I'm using this `centralPath` to help ensure that you can read the full path to the paper airplane data is. If the path can fit on the line without getting cut off, you don't need to take this step.

No matter what route you take, you'll always need to know the path to the data file. Additionally, knowing whether the first row contains column headers (i.e., `header = TRUE`) as well as what character (comma, semicolon, whitespace) separates columns from each other (i.e., `sep = ","`) are incredibly useful.

> **Check Your Import**
>
> After you load your data, I highly recommend that you check the result. You can do this by using the `View` function (e.g., typing `View(plane2)` in the console) OR by clicking on the data frame in the Environment list of RStudio. If things don't look correct, try adjusting the arguments of your import command and re-importing the data. The most common issue is the wrong separator character.

Clean your Data

After you load your data, you should always clean your data. What this step entails will differ from data collection to data collection as well as your goals. An important step for ANOVA models is to ensure that R is thinking about our response quantitatively (either as `int` or `num` data type) and our factors (and blocks) as the `factor` data type.

```
# Example Cleaning ----
## Section 2 Paper Airplane ----
### Simplify column names (i.e., variables)
names(plane2) <- c("design", "folder", "thrower", "distance")
### Set factor data type to columns involving character data
plane2 <- plane2 %>%
  mutate(
    across(where(is.character), factor)
  )

## Section 3 Paper Airplane ----
### Simplify column names
names(plane3) <- c("design", "distance", "notes")
### Clean the designs to remove the asterisks
plane3$design <- gsub(pattern = "\\*", replacement = "", x = plane3$design)
### Set design to factor data type
#### NOTE: this has to be done AFTER the removal of asterisks
plane3$design <- as.factor(plane3$design)

## Barley Data ----
### Use a contextual name for the factor and set data type
names(barleyData)[which(names(barleyData) == "Treatment")] <- "Varietal"
barleyData$Varietal <- as.factor(barleyData$Varietal)
### Set factor data type for field
barleyData$Field <- as.factor(barleyData$Field)
### Use a simpler name for order
names(barleyData)[which(names(barleyData) == "Planting.Harvesting.Order")] <- "Order"
```

ANOVA Screens for One-way ANOVA Models

Our first encounter with ANOVA screens has been with One-way ANOVA models like that of the Paper Airplane studies. These are models that have one factor and other other terms beyond the main action and the residuals.

To identify our screens, we'll draw upon our model. Each node in the Hasse diagram will be a screen. Further, each screen will be a separate term in our algebraic formula. For the Paper Airplane studies, we can turn our attention to the following Hasse diagrams (see Figure 1).

(a) Section 2 (b) Section 3

Figure 1: Hasse Diagrams for the Paper Airplane Studies

The `anovaScreens` function will not produce a visualization like what we saw in class for the ANOVA screens. Rather, this function will return a new, tidy data frame that will show the original data values as well as the screens for each measurement unit. Thus, the Hasse diagrams serve an extra purpose of visually reminding us that we are passing our data through the sequence of screens.

To use the `anovaScreens` function, we simply need to call the function by name and pass along three inputs: the data frame, the response name, and the factor name. I will quickly note that if there are any additional columns in the data frame you pass, they will not be returned by `anovaScreens`. Thus, I highly recommend that you save the output as a new data frame.

```r
# Using the anovaScreens Function ----
## Section 2
plane2_Screens <- anovaScreens(
  dataFrame = plane2,                                                       ①
  response = "distance",                                                    ②
  factor = "design"
)

## Section 3
plane3_Screens <- anovaScreens(
  dataFrame = plane3,
  response = "distance",
  factor = "design"
)
```

① The `dataFrame` argument needs to be the object name of the data frame exactly as listed in your current R session/environment.

② The values for the `response` and `factor` arguments need to be strings (i.e., enclosed in quotation marks) that match the column names in the data frame.

Table 1: ANOVA Screens for Section 2's Paper Airplane Study

distance	design	Screen1.Action	Screen2.Factor	Screen3.Residuals
33.0	Basic	99.239	-34.489	-31.750
30.0	Basic	99.239	-34.489	-34.750
100.0	Basic	99.239	-34.489	35.250
27.5	Basic	99.239	-34.489	-37.250
86.0	Basic	99.239	-34.489	21.250
112.0	Basic	99.239	-34.489	47.250
105.0	Buzz	99.239	-7.739	13.500
137.0	Buzz	99.239	-7.739	45.500
147.5	Buzz	99.239	-7.739	56.000
81.5	Buzz	99.239	-7.739	-10.000
84.0	Buzz	99.239	-7.739	-7.500
-6.0	Buzz	99.239	-7.739	-97.500
44.5	Royal Wing	99.239	-66.239	11.500
32.5	Royal Wing	99.239	-66.239	-0.500
24.5	Royal Wing	99.239	-66.239	-8.500
19.5	Royal Wing	99.239	-66.239	-13.500
44.0	Royal Wing	99.239	-66.239	11.000
193.5	Origami Plane	99.239	97.428	-3.167
243.0	Origami Plane	99.239	97.428	46.333
162.0	Origami Plane	99.239	97.428	-34.667
180.0	Origami Plane	99.239	97.428	-16.667
197.5	Origami Plane	99.239	97.428	0.833
204.0	Origami Plane	99.239	97.428	7.333

In the above code, I saved the output of the `anovaScreens` function calls into `plane2_Screens` and `plane3_Screens` by using the assignment operator, `<-`. If you do the same, then the output will not get printed to either your console or in your document. Table 1 shows Section 2's results.

```r
# Display the ANOVA Screens for Section 2's Data ----
kable(
  x = plane2_Screens,
  digits = 3,
  align = "cccc",
  booktab = TRUE,
  table.attr = 'data-quarto-disable-processing="true"' # (1)
) %>%
  kableExtra::kable_classic(
    latex_options = c("HOLD_position"),
    full_width = FALSE
  )
```

(1) Omit this line if you are using R Markdown; only necessary if you are using Quarto.

Table 2: ANOVA Screens for Section 3's Paper Airplane Study

distance	design	Screen1.Action	Screen2.Factor	Screen3.Residuals
43.375	Basic	160.573	-102.358	-14.840
75.375	Basic	160.573	-102.358	17.160
56.250	Basic	160.573	-102.358	-1.965
5.880	Basic	160.573	-102.358	-52.335
66.500	Basic	160.573	-102.358	8.285
75.625	Basic	160.573	-102.358	17.410
84.500	Basic	160.573	-102.358	26.285
222.175	Dart	160.573	88.237	-26.636
239.375	Dart	160.573	88.237	-9.436
199.750	Dart	160.573	88.237	-49.061
295.000	Dart	160.573	88.237	46.189
221.000	Dart	160.573	88.237	-27.811
247.375	Dart	160.573	88.237	-1.436
317.000	Dart	160.573	88.237	68.189
203.000	Bullet	160.573	109.837	-67.411
352.000	Bullet	160.573	109.837	81.589
88.875	Bullet	160.573	109.837	-181.536
288.000	Bullet	160.573	109.837	17.589
341.000	Bullet	160.573	109.837	70.589
327.500	Bullet	160.573	109.837	57.089
292.500	Bullet	160.573	109.837	22.089
69.500	Ring Wing	160.573	-95.716	4.643
73.000	Ring Wing	160.573	-95.716	8.143
93.000	Ring Wing	160.573	-95.716	28.143
50.000	Ring Wing	160.573	-95.716	-14.857
31.500	Ring Wing	160.573	-95.716	-33.357
97.250	Ring Wing	160.573	-95.716	32.393
39.750	Ring Wing	160.573	-95.716	-25.107

Section 3's results appear in Table 2.

```r
# Display the ANOVA Screens for Section 3's Data ----
kable(
  x = plane3_Screens,
  digits = 3,
  align = "cccc",
  booktab = TRUE,
  table.attr = 'data-quarto-disable-processing="true"'  # (1)
) %>%
  kableExtra::kable_classic(
    latex_options = c("HOLD_position"),
    full_width = FALSE
  )
```

(1) Omit this line if you are using R Markdown; only necessary if you are using Quarto.

In both Table 1 and Table 2, we have separate columns for each of our screens. Looking through these columns, we can see the values that each screen captured. We can work with these data frames such as verifying the constraint on factor effects and that our residuals add to zero.

```
# Check factor effects add to zero ----
sum(plane2_Screens$Screen2.Factor)
```

```
[1] 1.136868e-13
```

```
sum(plane3_Screens$Screen2.Factor)
```

```
[1] -1.136868e-13
```

```
# Check Residuals add to zero ----
sum(plane2_Screens$Screen3.Residuals)
```

```
[1] 5.684342e-14
```

```
sum(plane3_Screens$Screen3.Residuals)
```

```
[1] 2.842171e-14
```

You'll notice that the sums aren't technically zero. The values reported are incredibly close to zero and a result of how computers do arithmetic. We will call them zero.

Generally speaking, these tables should ***not*** appear in the body of a report. Hasse diagrams and algebraic formulas are the go-to visual representations. If you feel that you absolutely must include such a table in a report, then these should appear only as an appendix. Tables of data don't have a worthwhile communicative purpose in reports–even if you only include a small subset of the data frame. Use the space for more meaningful visualizations.

ANOVA Sreens with a Block

My `anovaScreens` function may be used in ANOVA situations involving a single factor and a single block. A full discussion of what these models entail will be saved for later in the course and another guide.

For the Barley Study, our Hasse diagram appears in Figure 2.

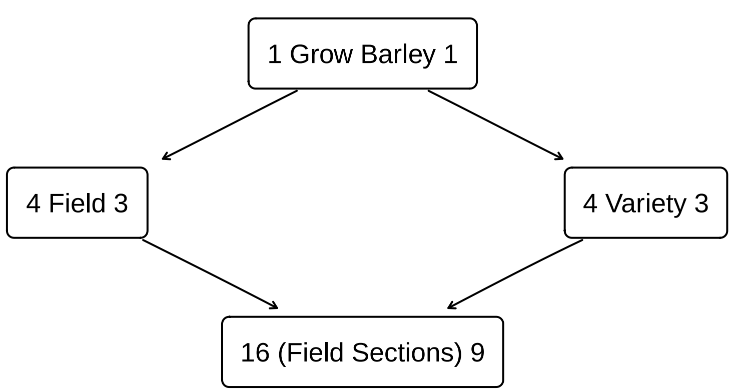

Figure 2: Hasse Diagrams for the Barley Study

Table 3: ANOVA Screens for Barley Study

Yield	Field	Varietal	Screen1.Action	Screen2.Block	Screen3.Factor	Screen4.Residuals
93.5	Field 1	Variety 1	59.8	3.450	24.175	6.075
66.6	Field 1	Variety 2	59.8	3.450	-4.300	7.650
50.5	Field 1	Variety 3	59.8	3.450	-13.600	0.850
42.4	Field 1	Variety 4	59.8	3.450	-20.075	-0.775
102.9	Field 2	Variety 1	59.8	2.025	25.600	15.475
53.2	Field 2	Variety 2	59.8	2.025	-2.875	-5.750
47.4	Field 2	Variety 3	59.8	2.025	-12.175	-2.250
43.8	Field 2	Variety 4	59.8	2.025	-18.650	0.625
67.0	Field 3	Variety 1	59.8	-6.850	34.475	-20.425
54.7	Field 3	Variety 2	59.8	-6.850	6.000	-4.250
50.0	Field 3	Variety 3	59.8	-6.850	-3.300	0.350
40.1	Field 3	Variety 4	59.8	-6.850	-9.775	-3.075
86.3	Field 4	Variety 1	59.8	1.375	26.250	-1.125
61.3	Field 4	Variety 2	59.8	1.375	-2.225	2.350
50.7	Field 4	Variety 3	59.8	1.375	-11.525	1.050
46.4	Field 4	Variety 4	59.8	1.375	-18.000	3.225

The `anovaScreens` function has a `block` argument that allows us to pass the name of the block along to the process. Table 3 shows the result of doing this in the Barley study. Generally, we will pass data through a block screen before a factor screen.

```
# Apply ANOVA Screens to Barley Study ----
barley_Screens <- anovaScreens(
  dataFrame = barleyData,
  response = "Yield",
  factor = "Varietal",
  block = "Field"
)

# Display the ANOVA Screens for the Barley Study ----
kable(
  x = barley_Screens,
  digits = 3,
  align = "cccc",
  booktab = TRUE,
  table.attr = 'data-quarto-disable-processing="true"'
) %>%
  kableExtra::kable_classic(
    latex_options = c("HOLD_position"),
    full_width = FALSE
  )
```

Just as with One-way ANOVA, we can work with the output of the `anovaScreens` call in this context. Similarly, we don't want to typically include tables such as Table 3 in a report.

Code Appendix

```
source("https://raw.github.com/neilhatfield/STAT461/master/rScripts/checkSetup.R")
checkSetup()
# Load Packages ----
packages <- c("tidyverse", "hasseDiagram", "openxlsx", "knitr", "kableExtra")   ①
lapply(                                                                            ②
  X = packages,
  FUN = library,
  character.only = TRUE,
  quietly = TRUE
)

# Load extra tools ----
source("https://raw.github.com/neilhatfield/STAT461/main/rScripts/ANOVATools.R")

# Load Paper Airplane Data ----
centralPath <- "https://raw.github.com/neilhatfield/STAT461/main/dataFiles/"
## Section 2 ----
plane2 <- readWorkbook(
  xlsxFile = paste0(centralPath, "PaperAirplanes_Sp24_Sec2.xlsx")
)
## Section 3 ----
plane3 <- readWorkbook(
  xlsxFile = paste0(centralPath, "PaperAirplanes_Sp24_Sec3.xlsx")
)

# Load Barley Data ----
## Example for the One-way ANOVA + Block Section
barleyData <- read.table(
  file = "https://raw.github.com/neilhatfield/STAT461/master/dataFiles/barley.dat",
  header = TRUE,
  sep = ","
)

# Example Cleaning ----
## Section 2 Paper Airplane ----
### Simplify column names (i.e., variables)
names(plane2) <- c("design", "folder", "thrower", "distance")
### Set factor data type to columns involving character data
plane2 <- plane2 %>%
  mutate(
    across(where(is.character), factor)
  )

## Section 3 Paper Airplane ----
### Simplify column names
names(plane3) <- c("design", "distance", "notes")
### Clean the designs to remove the asterisks
plane3$design <- gsub(pattern = "\\*", replacement = "", x = plane3$design)
### Set design to factor data type
#### NOTE: this has to be done AFTER the removal of asterisks
```

```r
plane3$design <- as.factor(plane3$design)

## Barley Data ----
### Use a contextual name for the factor and set data type
names(barleyData)[which(names(barleyData) == "Treatment")] <- "Varietal"
barleyData$Varietal <- as.factor(barleyData$Varietal)
### Set factor data type for field
barleyData$Field <- as.factor(barleyData$Field)
### Use a simpler name for order
names(barleyData)[which(names(barleyData) == "Planting.Harvesting.Order")] <- "Order"

# Section 2 Hasse Diagram ----
modelLabels <- c("1 Fly 1", "4 Design 3", "23 (Planes) 19")
modelMatrix <- matrix(
  data = c(FALSE, FALSE, FALSE, TRUE, FALSE, FALSE, TRUE, TRUE, FALSE),
  nrow = 3,
  ncol = 3,
  byrow = FALSE
)
hasseDiagram::hasse(
 data = modelMatrix,
 labels = modelLabels
)

# Section 3 Hasse Diagram ----
modelLabels <- c("1 Fly 1", "4 Design 3", "28 (Planes) 24")
modelMatrix <- matrix(
  data = c(FALSE, FALSE, FALSE, TRUE, FALSE, FALSE, TRUE, TRUE, FALSE),
  nrow = 3,
  ncol = 3,
  byrow = FALSE
)
hasseDiagram::hasse(
 data = modelMatrix,
 labels = modelLabels
)

# Using the anovaScreens Function ----
## Section 2
plane2_Screens <- anovaScreens(
  dataFrame = plane2,                                                          ①
  response = "distance",                                                       ②
  factor = "design"
)

## Section 3
plane3_Screens <- anovaScreens(
  dataFrame = plane3,
  response = "distance",
  factor = "design"
)
```

```r
# Display the ANOVA Screens for Section 2's Data ----
kable(
  x = plane2_Screens,
  digits = 3,
  align = "cccc",
  booktab = TRUE,
  table.attr = 'data-quarto-disable-processing="true"'          ①
) %>%
  kableExtra::kable_classic(
    latex_options = c("HOLD_position"),
    full_width = FALSE
  )

# Display the ANOVA Screens for Section 3's Data ----
kable(
  x = plane3_Screens,
  digits = 3,
  align = "cccc",
  booktab = TRUE,
  table.attr = 'data-quarto-disable-processing="true"'
) %>%
  kableExtra::kable_classic(
    latex_options = c("HOLD_position"),
    full_width = FALSE
  )

# Check factor effects add to zero ----
sum(plane2_Screens$Screen2.Factor)
sum(plane3_Screens$Screen2.Factor)

# Check Residuals add to zero ----
sum(plane2_Screens$Screen3.Residuals)
sum(plane3_Screens$Screen3.Residuals)
# Barley Hasse Diagram
modelLabels <- c("1 Grow Barley 1", "4 Field 3", "4 Variety 3", "16 (Field Sections) 9")
modelMatrix <- matrix(
  data = c(FALSE, FALSE, FALSE, FALSE, TRUE, FALSE, FALSE, FALSE, TRUE, FALSE, FALSE,
           FALSE, TRUE, TRUE, TRUE, FALSE),
  nrow = 4,
  ncol = 4,
  byrow = FALSE
)
hasseDiagram::hasse(
 data = modelMatrix,
 labels = modelLabels
)

# Apply ANOVA Screens to Barley Study ----
barley_Screens <- anovaScreens(
  dataFrame = barleyData,
  response = "Yield",
  factor = "Varietal",
```

```r
  block = "Field"
)

# Display the ANOVA Screens for the Barley Study ----
kable(
  x = barley_Screens,
  digits = 3,
  align = "cccc",
  booktab = TRUE,
  table.attr = 'data-quarto-disable-processing="true"'
) %>%
  kableExtra::kable_classic(
    latex_options = c("HOLD_position"),
    full_width = FALSE
  )
```