C2M2-autograded

February 4, 2022

1 C2M2: Autograded Assignment

1.0.1 Outline:

Here are the objectives of this assignment:

1.

Here are some general tips:

1. Read the questions carefully to understand what is being asked.
2. When you feel that your work is completed, feel free to hit the `Validate` button to see your results on the *visible* unit tests. If you have questions about unit testing, please refer to the "Module 0: Introduction" notebook provided as an optional resource for this course. In this assignment, there are hidden unit tests that check your code. You will not recieve any feedback for failed hidden unit tests until the assignment is submitted. **Do not misinterpret the feedback from visible unit tests as all possible tests for a given question–write your code carefully!**
3. Before submitting, we recommend restarting the kernel and running all the cells in order that they appear to make sure that there are no additional bugs in your code.

[21]:
```r
# Load required libraries
library(testthat)
library(tidyverse)
library(RCurl)
```

2 Problem 1: Post-Hoc Tests

Recently, your local highschool switched their student lunches from circular pizzas to square pizzas. Suprisingly the school reported a change in the overall testing of students in the following weeks. The school decided to test this theory, and has recorded test results following lunches with four different shaped pizzas. It is up to you to determine if the shapes of pizza does in fact improve student's abilities to take tests, and if so, which shaped pizza results in the best test scores.

The school has tested four different shapes, coded as the following: * a: Circular * b: Square * c: Triangular * d: Cylindrical

```
[22]: # Load the data
      df.pizza = read.csv("pizza.csv")
      head(df.pizza)
```

A data.frame: 6 × 3

	X <int>	shape <fct>	score <dbl>
1	1	a	78.44676
2	2	a	88.75349
3	3	a	80.19209
4	4	a	84.04420
5	5	a	78.50873
6	6	a	82.34018

2.0.1 1. (a) Intuition and ANOVA

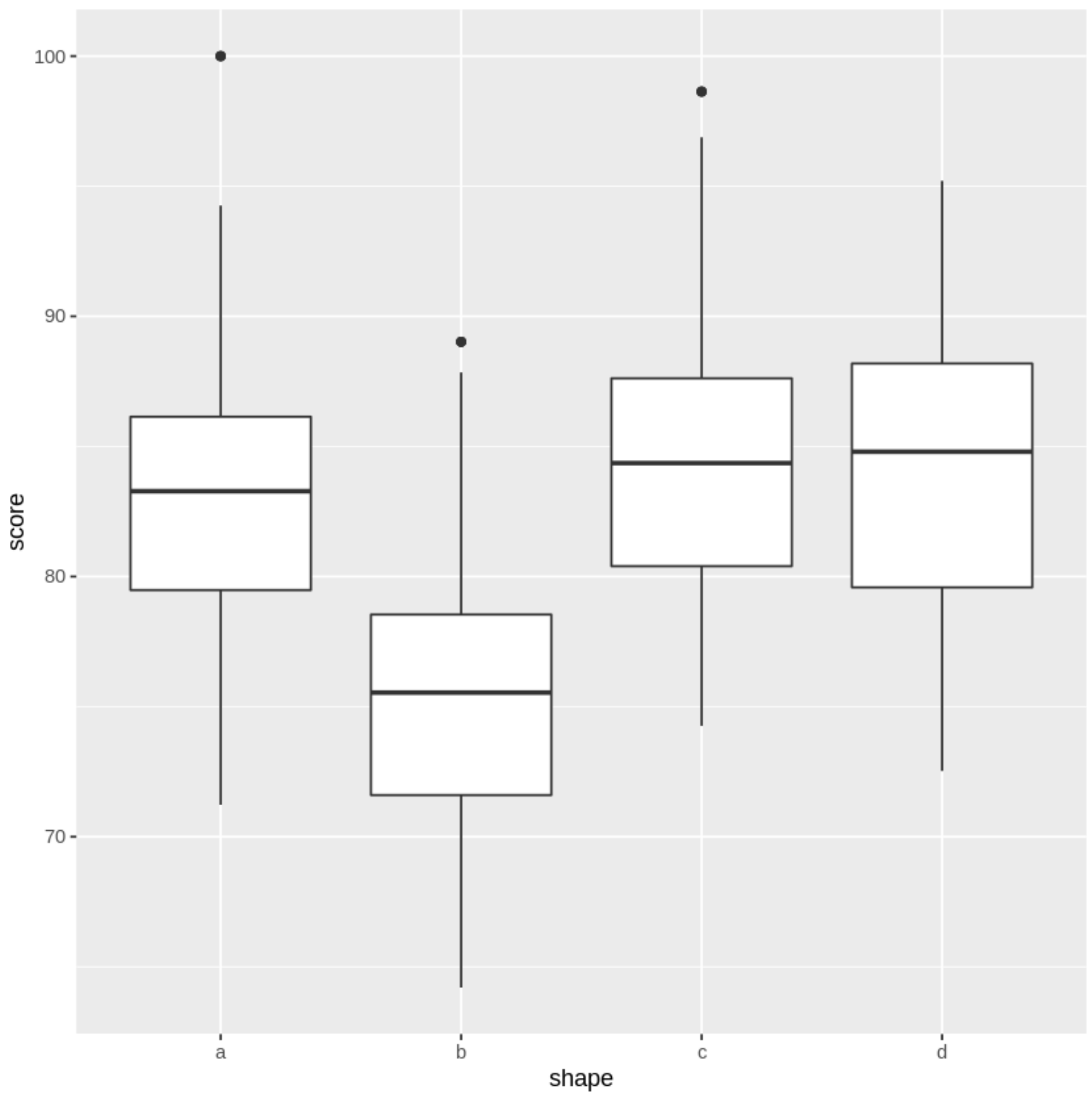
Instead of jumping into direct comparisons, we should check all the pizza shapes resulted in the same scores. Or, in other words, if at least one shape resulted in different test scores than the others.

Using ggplot, create a boxplot of the different shapes. Save your boxplot as `pizza.boxplot`.

Then determine if at least one shape resulted in different test scores than the others. In `pizza.diff`, answer `TRUE` if there is a difference and `FALSE` if there is not a difference.

```
[23]: pizza.boxplot = NA
      pizza.diff = NA

      # your code here
      pizza.boxplot = ggplot(df.pizza, aes(x=shape, y=score)) + geom_boxplot()
      pizza.diff = TRUE
      pizza.boxplot
```



[24]:
```
# Test Cell
# This cell has hidden test cases that will run after submission.
```

2.0.2 1. (b) Type I Error Rate

The problem with pairwise tests is that of compounding error. As each test has a probability of getting an incorrect answer, then the chance of at least one test being incorrect increases as you increase the number of tests. For the following, use a significance of $\alpha = 0.05$.

For our data, calculate the probability that at least one test has type 1 error if we conduct pair-wise comparisons for all combinations of labels? Store your answer in `pizza.error`.

Then determine the probability that at least one test has type 1 error for all possible pairwise tests, using the Bonferroni correction. Store your answer as `bonferroni.error`.

[25]:
```
#got 6 points for this
pizza.error = NA
bonferroni.error = NA
alpha = 0.05

# your code here
pizza.error = 1 - ((1-alpha)^6)
bonferroni.error = 1 -((1 - alpha/6)^6)
pizza.error
bonferroni.error
```

0.264908109375

0.0489698353102359

[26]:
```
# Test Cell
# This cell has hidden test cases that will run after submission.
```

2.0.3 1. (c) Tukey's Test

Now let's do our post-hoc tests. Using Tukey's Test, determine which shapes result in the same test scores. Store the pairs of shapes that are the same in a dataframe named `pizza.post.hoc` with the first column named `shape.1` and second column named `shape.2`.

For example, if `a-b` and `a-c` are the only two shapes that result in the same scores, your final dataframe would be created by:

`data.frame(shape.1=c("a", "a"), shape.2=c("b", "c"))`

[27]:
```
pizza.post.hoc = NA

# your code here
lmod = lm(score ~ shape, data = df.pizza)
av_tuk = aov(lmod)
TukeyHSD(av_tuk)
```

```
  Tukey multiple comparisons of means
    95% family-wise confidence level

Fit: aov(formula = lmod)

$shape
          diff        lwr        upr     p adj
b-a -7.1816919 -8.4009572 -5.9624266 0.0000000
c-a  1.4769816  0.2577163  2.6962469 0.0101259
d-a  1.1158565 -0.1034088  2.3351218 0.0866321
c-b  8.6586735  7.4394082  9.8779388 0.0000000
```

```
d-b  8.2975484  7.0782831  9.5168137 0.0000000
d-c -0.3611251 -1.5803904  0.8581402 0.8714151
```

[28]:
```
# data frame of shaped with no difference
pizza.post.hoc = data.frame(shape.1=c("d", "d"), shape.2=c("a", "c"))
pizza.post.hoc
```

A data.frame: 2 × 2

shape.1 <fct>	shape.2 <fct>
d	a
d	c

[29]:
```
# Test Cell

test_that("Check that answer is a dataframe", expect_is(pizza.post.hoc, "data.
↪frame"))
# This cell has hidden test cases that will run after submission.
```

2.0.4 1. (d) Bonferroni's Correction

Repeat the calculations from **1.c**, but include the Bonferroni Correction in your calculations. Report the pairs of shapes in a dataframe named `bonferroni.post.hoc`, of the same specifications as in **1.c**.

[30]:
```
library(multcomp)
```

[31]:
```
bonferroni.post.hoc = NA

# your code here
pairwise.t.test(df.pizza$score, df.pizza$shape, p.adjust.method="bonferroni",␣
↪conf.level= 0.95)
bonferroni.post.hoc = data.frame(shape.1=c("d", "d"), shape.2=c("a", "c"))
```

```
Pairwise comparisons using t tests with pooled SD

data:  df.pizza$score and df.pizza$shape

  a      b      c
b <2e-16 -      -
c 0.011  <2e-16 -
d 0.112  <2e-16 1.000

P value adjustment method: bonferroni
```

[32]:
```
# Test Cell
# This cell has hidden test cases that will run after submission.
```

3 Problem 2: The Great Gum Debacle (16 Points)

Consider the following experiment: You record data on how long different brands of gum hold their flavor. The brands under consideration are Scepter, Frost, Dubba Bubba, and 8-3 Gum. For each brand, you test 5 pieces and get the following average number of minutes that they maintained their flavor, respectively: $33, 24, 12, 15$. All of the gums had a variance of 49 minutes. Somehow.

3.0.1 2. (a) Power Overwhelming

Determine the power of this experiment at the 0.05 significance level. Store you answer as `power.gum`.

[33]:
```
groupmeans = c(33, 24, 12, 15)
power.gum = NA

# your code here
power.anova.test(groups = length(groupmeans), between.var= var(groupmeans),
                             within.var=49, power=NULL, sig.level=0.05, n =5)
power.gum = 0.9822989
```

```
     Balanced one-way analysis of variance power calculation

         groups = 4
              n = 5
    between.var = 90
     within.var = 49
      sig.level = 0.05
          power = 0.9822989

NOTE: n is number in each group
```

[34]:
```
# Test Cell
# This cell has hidden test cases that will run after submission.
```

3.0.2 2. (b) How much gum does it take?

Suppose we haven't performed this experiment yet, and are using theoretical gum statistics to get our values. For the same experiment, if we want a final power of 0.85, how many piece of each gum would we need to chew? Save your answer as `n.gum`.

[35]:
```
n.gum = NA

# your code here
power.anova.test(groups = length(groupmeans), between.var= var(groupmeans),
                             within.var=49, power=0.85, sig.level=0.05, n =NULL)
n.gum = 3.393957
```

```
     Balanced one-way analysis of variance power calculation

         groups = 4
              n = 3.393957
    between.var = 90
     within.var = 49
      sig.level = 0.05
          power = 0.85

NOTE: n is number in each group
```

[36]:
```
# Test Cell
# This cell has hidden test cases that will run after submission.
```

[ ]:
```

```