Chapter 4: Basic Analyses

Tyson S. Barrett

Summer 2017

Utah State University

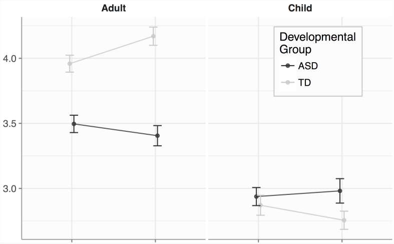
Introduction

Basic Analyses: The analyses taught in the first stats course

These include:

1. T-tests
2. ANOVA
3. Linear Regression

These allow us to assess relationships like that in the figure.



Maybe surprising:\ These all are doing essentially the same thing!

First, **T-TESTS!**

T-tests

Three Types

1. Simple
2. Independent Samples
3. Paired Samples

Three Types

Each will be demonstrated using:

```
df <- data.frame("A"=sample(c(0,1), 100, replace = TRUE),
                 "B"=rnorm(100),
                 "C"=rnorm(100))
df
```

```
    A            B            C
1   1 -1.634569035  1.136084564
2   0  0.920975586 -0.351869884
3   1 -0.968021229 -0.339548892
4   1  1.303420399 -0.644911064
5   0  0.439410726 -0.648788673
6   0 -1.117808884  0.324842056
7   1  0.721734088 -0.323065810
8   1  1.718606636 -0.820410249
9   0 -0.371234569 -0.856676250
```

Simple

Comparing a mean of a variable with μ.

```r
t.test(df$B, mu = 0)
```

```
    One Sample t-test

data:  df$B
t = 0.62805, df = 99, p-value = 0.5314
alternative hypothesis: true mean is not equal to 0
95 percent confidence interval:
 -0.1255241  0.2417868
sample estimates:
 mean of x
0.05813135
```

Independent Samples

Comparing the means of two groups (`dfA` is the grouping variable).

```
t.test(df$B ~ df$A)
```

```
	Welch Two Sample t-test

data:  df$B by df$A
t = 0.1167, df = 90.352, p-value = 0.9074
alternative hypothesis: true difference in means is not equal to
95 percent confidence interval:
 -0.3515987  0.3954865
sample estimates:
mean in group 0 mean in group 1
     0.07063939      0.04869546
```

Paired Samples

Comparing repeated measures (e.g., Pretest vs. Posttest).

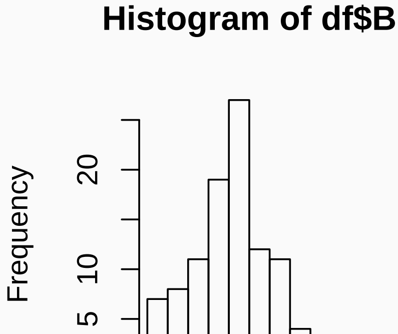
```
t.test(df$B, df$C, paired = TRUE)
```

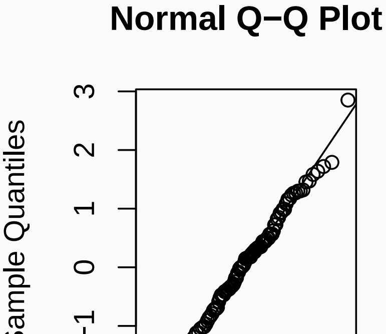
```

	Paired t-test

data:  df$B and df$C
t = 0.15378, df = 99, p-value = 0.8781
alternative hypothesis: true difference in means is not equal to
95 percent confidence interval:
 -0.2393093  0.2795205
sample estimates:
mean of the differences
             0.02010561
```

Testing Assumptions of T-Tests

T-tests require that the data be normally distributed with approximately the same variance.

```
## Normality
par(mfrow = c(1,2))
hist(df$B)
qqnorm(df$B)
abline(a=0, b=1)
```





ANOVA

Analysis of Variance

The Analysis of Variance (ANOVA) is highly related to t-tests but can handle 2+ groups.

1. Provides the same p-value as t-tests
2. $t^2 = F$

For example:

```
fit_ano = aov(df$B ~ df$A)
summary(fit_ano)
```

```
            Df Sum Sq Mean Sq F value Pr(>F)
df$A         1   0.01  0.0118   0.014  0.907
Residuals   98  84.80  0.8653
```

```
t.test(df$B ~ df$A)$p.value
```

```
[1] 0.9073553
```

Analysis of Variance

```
fit_ano = aov(df$B ~ df$A)
summary(fit_ano)
t.test(df$B ~ df$A)$p.value
```

Notice in the code:

- We assigned the `aov()` the name `fit_ano` (which we could have called anything)
- We used the `summary()` function to see the F and p values.
- We pulled the p-value right out of the `t.test()` function.

Types

1. One-Way
2. Two-Way (Factorial)
3. Repeated Measures
4. A combination of Factorial and Repeated Measures

Types

We will use the following data set for the examples:

```
library(tidyverse)
df <- data.frame("A"=sample(c(0,1), 100, replace = TRUE) %>% fac
                 "B"=rnorm(100),
                 "C"=rnorm(100),
                 "D"=sample(c(1:4), 100, replace = TRUE) %>% fac
df
```

```
    A            B            C D
1   1 -0.765813349 -1.227246676 2
2   1 -1.470818479 -0.953798870 3
3   0  0.318140483  0.676365198 1
4   0  0.478931301 -0.690003721 4
5   1  0.797005962  0.471830539 4
6   0 -1.905408725 -0.241857264 1
7   0  0.369894344 -0.078830706 4
8   0 -0.134437900  0.427207160 4
```

One-Way

A One-Way ANOVA can be run using aov().

```
fit1 = aov(B ~ D, data = df)
summary(fit1)
```

```
            Df Sum Sq Mean Sq F value Pr(>F)
D            3   7.68  2.5588   3.145 0.0287 *
Residuals   96  78.11  0.8137
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Two-Way

A Two-Way ANOVA uses essentially the exact same code with a minor change—including the other variable in an interaction.

```
fit2 = aov(B ~ D * A, data = df)
summary(fit2)
```

```
            Df Sum Sq Mean Sq F value Pr(>F)
D            3   7.68  2.5588   3.114  0.030 *
A            1   0.04  0.0406   0.049  0.825
D:A          3   2.48  0.8257   1.005  0.394
Residuals   92  75.60  0.8217
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

The `D:A` line highlights the interaction term whereas the others show the main effects.

Repeated Measures

To show this, we will add a fake `ID` variable to our already fake data set `df`.

```
df$ID = 1:100
```

And change our data to long (Can you remember how to do it?)

```
library(tidyverse)
df_long = gather(df, "var", "value", 2:3)
df_long
```

```
    A D  ID var        value
1   1 2   1   B -0.765813349
2   1 3   2   B -1.470818479
3   0 1   3   B  0.318140483
4   0 4   4   B  0.478931301
5   1 4   5   B  0.797005962
6   0 1   6   B -1.905408725
7   0 4   7   B  0.369894344
```

The repeated measures, besides using a long-form of the data, is very similar in code. In addition to our usual formula (e.g., `something ~ other + stuff`), we have the `Error()` function. This function tells `R` how the repeated measures are clustered. In general, you'll provide the subject ID.

The next slide highlights this.

Repeated Measures

```
fit3 = aov(value ~ var + Error(ID), data = df_long)
summary(fit3)
```

```
Error: ID
          Df Sum Sq Mean Sq F value Pr(>F)
Residuals  1  1.499   1.499

Error: Within
           Df Sum Sq Mean Sq F value Pr(>F)
var         1   4.24   4.236   5.043 0.0258 *
Residuals 197 165.48   0.840
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Here, `value` was the value of the repeated measures where `var` is the time. That means our oucome is testing if there were any differences from pre-test to post-test across all the groups.

Combination

To take the repeated measures a step further, we can do a Three-Way Repeated Measures ANOVA.

```
fit4 = aov(value ~ var * D * A + Error(ID), data = df_long)
summary(fit4)
```

The output is on the next slide...

Combination

```
Error: ID
  Df Sum Sq Mean Sq
D  1  1.499   1.499

Error: Within
           Df Sum Sq Mean Sq F value Pr(>F)
var         1   4.24   4.236   5.319 0.0222 *
D           3   1.63   0.544   0.683 0.5633
A           1   0.07   0.072   0.091 0.7636
var:D       3   8.57   2.858   3.588 0.0148 *
var:A       1   0.00   0.004   0.005 0.9461
D:A         3   8.30   2.765   3.472 0.0173 *
var:D:A     3   1.15   0.385   0.483 0.6942
Residuals 183 145.75   0.796
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Checking Assumptions

Of course, as with any statistical analysis, there are assumptions.

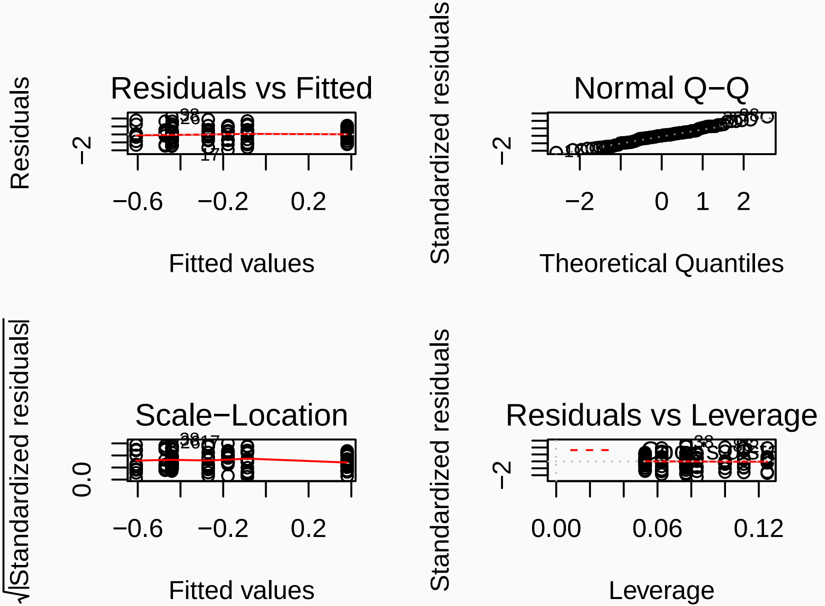
Many of these we can test.

Using our `fitX` objects from our ANOVAs above, we can look at our assumptions:

```
par(mfrow = c(2,2))  ## puts the four plots on a 2 x 2 grid
plot(fit2)
```

Again, the output is on the next slide...

Checking Assumptions



Checking Assumptions

They don't fit great on the slides but trust me that normality looks good. The assumption of homogeneity of variance looks good as well.

But, if you wanted to test it, you could.

```
library(car)
leveneTest(fit2)
```

```
Levene's Test for Homogeneity of Variance (center = median)
      Df F value Pr(>F)
group  7   0.327 0.9399
      92
```

Large p-value here is a good thing: `emo::ji("smile")` [1]

[1] This shows a smiley in 'R', just not on these slides—from the 'emo' package on GitHub.

Linear Regression

Linear Regression

Once again, linear regression is essentially the more flexible twin of ANOVA and t-tests.[2]

It can:

1. Handle continuous and categorical predictors (i.e., independent variables)
2. Less stringent assumption of equality of variances
3. Is what many other methods are built on (Chapter 5 and 6 will talk about some of these)

[2] It mainly only differs from ANOVA in the way it takes a dummy code rather than an effect code of the categorical variables.

Linear Regression

We will use `lm()` (Linear Model) to fit these models.

```r
fit5 = lm(B ~ A, data = df)
summary(fit5)
```

```

Call:
lm(formula = B ~ A, data = df)

Residuals:
    Min      1Q  Median      3Q     Max 
-1.9094 -0.6652  0.0356  0.6692  1.9487 

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept) -0.22050    0.13361  -1.650    0.102
A1           0.05337    0.18709   0.285    0.776

Residual standard error: 0.9352 on 98 degrees of freedom
```

Linear Regression

We can add an interaction with the *.

```
fit6 = lm(B ~ A*D, data = df)
summary(fit6)
```

```
Call:
lm(formula = B ~ A * D, data = df)

Residuals:
     Min       1Q   Median       3Q      Max 
-1.95215 -0.63769  0.00982  0.45819  2.22228 

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept) -0.27022    0.25141  -1.075    0.285
A1          -0.17046    0.35555  -0.479    0.633
D2           0.09247    0.36288   0.255    0.799
D3          -0.20133    0.40733  -0.494    0.622
```

Other Specifications

We can also make adjustments to the variables within the model.

First, we can transform the variables (e.g., log transformation).

```
fit7 = lm(log(B) ~ A*D, data = df)
summary(fit7)
```

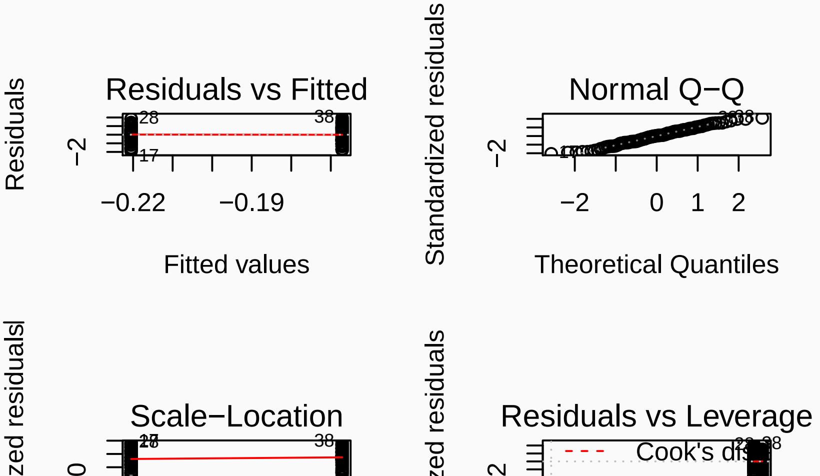
We can change the reference level of a variable, too.

```
fit8 = lm(B ~ relevel(D,ref = "4"), data = df)
summary(fit8)
```

Checking Assumptions

Assumption checking is similar to that of the linear model.

```
par(mfrow = c(2,2))
plot(fit5)
```



Reporting Results

Making This into a Table

Often we want to present this information in a table. This can be done is several ways:

1. Pulling information out of the model objects directly
2. Using a package like `stargazer` to do that work for you
3. Manually by hand

We can certainly do number 3 but why? So we'll look at both 1 and 2.

Pull information out of the model objects

The model objects contain loads of information that we can pull out:

1. Coefficients
2. Standard Errors and P-values
3. Confidence Intervals
4. Fit Statistics
5. Predicted Values
6. and more! [3]

[3] For a low cost of $49.99! Kidding. . .

Pull information out of the model objects

To see what the model object holds:

```
names(fit5)
```

```
 [1] "coefficients"  "residuals"     "effects"       "rank"
 [5] "fitted.values" "assign"        "qr"            "df.residual"
 [9] "contrasts"     "xlevels"       "call"          "terms"
[13] "model"
```

```
names(summary(fit5))
```

```
 [1] "call"          "terms"         "residuals"     "coefficients"
 [5] "aliased"       "sigma"         "df"            "r.squared"
 [9] "adj.r.squared" "fstatistic"    "cov.unscaled"
```

Pull information out of the model objects

Using that information we can grab:

```
summary(fit5)$coefficients
```

```
               Estimate Std. Error    t value  Pr(>|t|)
(Intercept) -0.22049836  0.1336069 -1.6503519 0.1020725
A1           0.05337395  0.1870870  0.2852894 0.7760245
```

or

```
summary(fit5)$fstatistic
```

```
      value       numdf       dendf
 0.08139004  1.00000000 98.00000000
```

Pull information out of the model objects

Put it in a table:

```
rbind(data.frame(summary(fit5)$coefficients, "Type"="Simple Regr
      data.frame(summary(fit6)$coefficients, "Type"="Interaction
```

```
                 Estimate Std..Error     t.value  Pr...t..
(Intercept)  -0.22049836  0.1336069 -1.65035186 0.1020725
A1            0.05337395  0.1870870  0.28528939 0.7760245
(Intercept)1 -0.27022085  0.2514114 -1.07481545 0.2852682
A11          -0.17046286  0.3555494 -0.47943510 0.6327669
D2            0.09247451  0.3628811  0.25483418 0.7994200
D3           -0.20133155  0.4073330 -0.49426771 0.6222953
D4            0.18358504  0.3384729  0.54239206 0.5888599
A1:D2        -0.09052975  0.5349676 -0.16922472 0.8659914
A1:D3         0.03429375  0.5579407  0.06146485 0.9511223
A1:D4         0.63769917  0.4701266  1.35644146 0.1782782
                             Type
(Intercept)  Simple Regression
```

Pull information out of the model objects

On the previous slide we:

1. Created two `data.frame` with the coefficients and a variable called `"Type"`
2. Glued them together by row with `rbind()`

This is a simple way of putting a table together that you can later export.

Use a package like `stargazer` to do that work for you

A simpler but less flexible way is using a package like `stargazer`.

```r
library(stargazer)
stargazer(fit5, fit6, type = "text")
```

```
=========================================================
                              Dependent variable:
                    -------------------------------------
                                       B
                           (1)                 (2)
---------------------------------------------------------
A1                        0.053              -0.170
                         (0.187)             (0.356)

D2                                            0.092
                                             (0.363)

D3                                           -0.201
                                             (0.407)

D4                                            0.184
                                             (0.338)

A1:D2                                        -0.091
                                             (0.535)

A1:D3                                         0.034
                                             (0.558)

A1:D4                                         0.638
                                             (0.470)

Constant                 -0.220              -0.270
```

Use a package like `stargazer` to do that work for you

This particular package can take several model objects and produce a nice table. It is hard to see but it includes the number of observations, fit statistics, the coefficients, and f-statistics.

Other packages exist that do similar things (e.g., `texreg`).

```
library(texreg)
screenreg(list(fit5, fit6))
```

Conclusions

Conclusion

1. Performing linear models is straightforward in ‘R‘
2. With a few lines of code, we can fit a model and check model assumptions
3. We can easily turn our model information into an informative table