An Introduction to R: Part II[1]

Mark Andrews
Psychology Department, Nottingham Trent University

@xmjandrews
mark.andrews@ntu.ac.uk
https://github.com/mark-andrews/u-herts-r-workshop

[1] These slides are not intended to be self-contained and comprehensive, but just aim to provide some of the workshop's content. Elaborations and explanations will be provided in the workshop itself.

Setup

```
> load('data/workshop_data.Rda')
```

Independent samples t-test

- We can use `t.test` for t-tests.

```
> M <- t.test(height ~ gender, data = weight_df)
> M
#>
#>  Welch Two Sample t-test
#>
#> data:  height by gender
#> t = -71.115, df = 4173.4, p-value < 2.2e-16
#> alternative hypothesis: true difference in means is not equal
#> 95 percent confidence interval:
#>  -13.12629 -12.42197
#> sample estimates:
#> mean in group female   mean in group male
#>             162.8473             175.6215
```

- By default, we get the *Welch Two Sample t-test*. Use `var.equal=T` to obtain the independent samples t-test.

```
> M <- t.test(height ~ gender, var.equal = T, data = weight_df)
```

Independent samples t-test

- We can access the attributes of the t-test with e.g.

```
> M$statistic
#>         t
#> -69.5244
> M$parameter
#>   df
#> 6066
> M$p.value
#> [1] 0
> M$conf.in
#> [1] -13.13432 -12.41394
#> attr(,"conf.level")
#> [1] 0.95
```

One sample t-test

```
> t.test(weight_df$weight, mu = 80)
#>
#>  One Sample t-test
#>
#> data:  weight_df$weight
#> t = -1.4462, df = 6067, p-value = 0.1482
#> alternative hypothesis: true mean is not equal to 80
#> 95 percent confidence interval:
#>  79.31548 80.10331
#> sample estimates:
#> mean of x
#>  79.70939
```

Paired samples t-test

```
> t.test(anorexia_df_cbt$Prewt,
+        anorexia_df_cbt$Postwt,
+        paired = T)
#>
#>  Paired t-test
#>
#> data:  anorexia_df_cbt$Prewt and anorexia_df_cbt$Postwt
#> t = -2.2156, df = 28, p-value = 0.03502
#> alternative hypothesis: true difference in means is not equal
#> 95 percent confidence interval:
#>  -5.7869029 -0.2268902
#> sample estimates:
#> mean of the differences
#>               -3.006897
```

Mann Whitney test

```
> wilcox.test(weight~gender, data=weight_df)
#>
#>  Wilcoxon rank sum test with continuity correction
#>
#> data:  weight by gender
#> W = 1267698, p-value < 2.2e-16
#> alternative hypothesis: true location shift is not equal to 0
```

Wilcoxon signed rank test

```
> wilcox.test(anorexia_df_cbt$Prewt,
+             anorexia_df_cbt$Postwt,
+             paired = T)
#>
#>  Wilcoxon signed rank test with continuity correction
#>
#> data:  anorexia_df_cbt$Prewt and anorexia_df_cbt$Postwt
#> V = 131.5, p-value = 0.06447
#> alternative hypothesis: true location shift is not equal to 0
```

Correlation

```
> cor.test(~ weight + height, data = weight_df)
#>
#>  Pearson's product-moment correlation
#>
#> data:  weight and height
#> t = 68.472, df = 6066, p-value < 2.2e-16
#> alternative hypothesis: true correlation is not equal to 0
#> 95 percent confidence interval:
#>  0.6458323 0.6742251
#> sample estimates:
#>       cor
#> 0.6602645
```

Spearman's ρ

```
> cor.test(~ weight + height,
+          method = 'spearman',
+          data = weight_df)
#>
#>  Spearman's rank correlation rho
#>
#> data:  weight and height
#> S = 1.2536e+10, p-value < 2.2e-16
#> alternative hypothesis: true rho is not equal to 0
#> sample estimates:
#>       rho
#> 0.6633652
```

Pearson's χ^2

```
> (M <- chisq.test(titanic_survival))
#>
#>  Pearson's Chi-squared test with Yates' continuity correction
#>
#> data:  titanic_survival
#> X-squared = 363.62, df = 1, p-value < 2.2e-16
```

As before, we can access properties of the test, e.g.

```
> M$expected
#>          sex
#> survived   female     male
#>      no  288.0015 520.9985
#>      yes 177.9985 322.0015
```

Linear regression

```
> M <- lm(weight ~ height, data= weight_df)
> summary(M)
#>
#> Call:
#> lm(formula = weight ~ height, data = weight_df)
#>
#> Residuals:
#>     Min      1Q  Median      3Q     Max
#> -35.563  -8.163  -0.594   7.239  46.482
#>
#> Coefficients:
#>              Estimate Std. Error t value Pr(>|t|)
#> (Intercept) -117.12802    2.87869  -40.69   <2e-16 ***
#> height         1.14814    0.01677   68.47   <2e-16 ***
#> ---
#> Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' '
#>
#> Residual standard error: 11.76 on 6066 degrees of freedom
#> Multiple R-squared:  0.4359, Adjusted R-squared:  0.4359
#> F-statistic:  4688 on 1 and 6066 DF,  p-value: < 2.2e-16
```

Linear regression

- We can get standardized residuals with `reghelper::beta`.

```
> library(reghelper)
> beta(M)
#>
#> Call:
#> lm(formula = "weight.z ~ height.z", data = data)
#>
#> Residuals:
#>      Min       1Q   Median       3Q      Max
#> -2.27196 -0.52149 -0.03797  0.46248  2.96955
#>
#> Coefficients:
#>              Estimate Std. Error t value Pr(>|t|)
#> (Intercept) 8.037e-16  9.642e-03    0.00        1
#> height.z    6.603e-01  9.643e-03   68.47   <2e-16 ***
#> ---
#> Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' '
#>
#> Residual standard error: 0.7511 on 6066 degrees of freedom
```

Prediction in linear regression

```
> new_df <- data.frame(height = c(140, 150, 160))
> predict(M, newdata = new_df)
#>        1        2        3
#> 43.61125 55.09263 66.57400
```

Multiple linear regression

```
> M <- lm(price ~ area + bedrooms, data = houseprice_df)
> summary(M)
#>
#> Call:
#> lm(formula = price ~ area + bedrooms, data = houseprice_df)
#>
#> Residuals:
#>     Min      1Q  Median      3Q     Max
#> -80.897  -4.247   1.539  13.249  42.027
#>
#> Coefficients:
#>              Estimate Std. Error t value Pr(>|t|)
#> (Intercept) -141.76132   67.87204  -2.089  0.05872 .
#> area           0.14255    0.04697   3.035  0.01038 *
#> bedrooms      58.32375   14.75962   3.952  0.00192 **
#> ---
#> Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' '
#>
#> Residual standard error: 33.06 on 12 degrees of freedom
#> Multiple R-squared:  0.731,  Adjusted R-squared:  0.6861
```

Multiple linear regression

- Multicolinearity

```
> library(car)
> car::vif(M)
#>     area bedrooms
#> 1.063122 1.063122
```

- Prediction

```
> new_df <- data.frame(area = median(houseprice_df$area),
+                      bedrooms = c(1, 2, 3, 4, 5)
+ )
> predict(M, newdata = new_df)
#>         1         2         3         4         5
#>  33.59346  91.91721 150.24096 208.56471 266.88846
```

Varying intercepts regression

```
> M_vi <- lm(weight ~ height + gender, data = weight_df)
> summary(M_vi)
#>
#> Call:
#> lm(formula = weight ~ height + gender, data = weight_df)
#>
#> Residuals:
#>     Min      1Q  Median      3Q     Max
#> -34.246  -7.811  -0.693   7.165  47.203
#>
#> Coefficients:
#>             Estimate Std. Error t value Pr(>|t|)
#> (Intercept) -87.72937    3.61930  -24.24   <2e-16 ***
#> height        0.95481    0.02217   43.07   <2e-16 ***
#> gendermale    5.56894    0.42523   13.10   <2e-16 ***
#> ---
#> Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' '
#>
#> Residual standard error: 11.59 on 6065 degrees of freedom
#> Multiple R-squared:  0.4515, Adjusted R-squared:  0.4513
```

Varying slopes regression

```
> M_vs <- lm(weight ~ height * gender, data=weight_df)
> summary(M_vs)
#>
#> Call:
#> lm(formula = weight ~ height * gender, data = weight_df)
#>
#> Residuals:
#>     Min      1Q  Median      3Q     Max
#> -34.388  -7.811  -0.664   7.130  47.042
#>
#> Coefficients:
#>                    Estimate Std. Error t value Pr(>|t|)
#> (Intercept)       -80.88583    6.60624 -12.244   <2e-16 ***
#> height              0.91278    0.04054  22.518   <2e-16 ***
#> gendermale         -4.42314    8.08057  -0.547    0.584
#> height:gendermale   0.05995    0.04842   1.238    0.216
#> ---
#> Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' '
#>
#> Residual standard error: 11.59 on 6064 degrees of freedom
```

Model comparison

- Model comparison of the varying intercepts and varying slopes models.

```
> anova(M_vi, M_vs)
#> Analysis of Variance Table
#>
#> Model 1: weight ~ height + gender
#> Model 2: weight ~ height * gender
#>   Res.Df    RSS Df Sum of Sq      F Pr(>F)
#> 1   6065 815391
#> 2   6064 815185  1    206.12 1.5333 0.2157
```

One-way Anova

```
> M <- aov(weight ~ group, data=plantgrowth_df)
> M
#> Call:
#>    aov(formula = weight ~ group, data = plantgrowth_df)
#>
#> Terms:
#>                    group Residuals
#> Sum of Squares   3.76634  10.49209
#> Deg. of Freedom        2        27
#>
#> Residual standard error: 0.6233746
#> Estimated effects may be unbalanced
```

One-way Anova

- We can do Tukey's range test to perform multiple comparisons:

```
> TukeyHSD(M)
#>   Tukey multiple comparisons of means
#>     95% family-wise confidence level
#>
#> Fit: aov(formula = weight ~ group, data = plantgrowth_df)
#>
#> $group
#>             diff        lwr       upr     p adj
#> trt1-ctrl -0.371 -1.0622161 0.3202161 0.3908711
#> trt2-ctrl  0.494 -0.1972161 1.1852161 0.1979960
#> trt2-trt1  0.865  0.1737839 1.5562161 0.0120064
```

One way Anova

- Note that we can also we can do Anova using `lm()`:

```
> M <- lm(weight ~ group, data=PlantGrowth)
> anova(M)
#> Analysis of Variance Table
#>
#> Response: weight
#>           Df  Sum Sq Mean Sq F value  Pr(>F)
#> group      2  3.7663  1.8832  4.8461 0.01591 *
#> Residuals 27 10.4921  0.3886
#> ---
#> Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' '
```

Two-way anova

```
> M <- aov(len ~ supp*dose, data=toothgrowth_df)
```

and we can also do

```
> TukeyHSD(M)
```

Repeated measures Anova

```
> M <- aov(score ~ condition + Error(Subject/condition), data=re
```

Multiple comparisons, with Bonferroni correction

```
> with(recall_long_df,
+      pairwise.t.test(x=score, g=condition),
+      p.adjust.methods='bonferroni',
+      paired=T)
#>
#>  Pairwise comparisons using t tests with pooled SD
#>
#> data:  score and condition
#>
#>     Neg     Neu
#> Neu 1.9e-05 -
#> Pos 0.00014 7.1e-08
#>
#> P value adjustment method: holm
```

Twoway repeated measures Anova

```
> M <- aov(Recall ~ Valence*Task + Error(Subject/(Task*Valence))
+          data=recall_long_df_2)
> M
#>
#> Call:
#> aov(formula = Recall ~ Valence * Task + Error(Subject/(Task *
#>     Valence)), data = recall_long_df_2)
#>
#> Grand Mean: 11.8
#>
#> Stratum 1: Subject
#>
#> Terms:
#>                 Residuals
#> Sum of Squares   349.1333
#> Deg. of Freedom         4
#>
#> Residual standard error: 9.342555
#>
#> Stratum 2: Subject:Task
```

Multilevel models

The repeated measures anova above can be done, and I think *should* be done, using multilevel models too.

```
> library(lme4)
> M <- lmer(Recall ~ Valence*Task + (1|Subject),
+           data=recall_long_df_2)
> summary(M)
#> Linear mixed model fit by REML ['lmerMod']
#> Formula: Recall ~ Valence * Task + (1 | Subject)
#>    Data: recall_long_df_2
#>
#> REML criterion at convergence: 118.4
#>
#> Scaled residuals:
#>      Min       1Q   Median       3Q      Max
#> -2.06749 -0.61797  0.00007  0.63519  1.35808
#>
#> Random effects:
#>  Groups   Name        Variance Std.Dev.
#>  Subject  (Intercept) 14.027   3.745
```

Random intercepts model

- If we label our the reaction time, subject, and day on observation i by y_i, $s_i \in \{1, 2 \ldots J\}$, and x_i, respectively, a random intercepts model of this data would be

$$y_i \sim N(a_{s_i} + bx_i, \sigma^2), \quad \text{for all } i \in 1, 2 \ldots n$$
$$a_j \sim N(\alpha, \tau^2), \quad \text{for all } j \in 1, 2 \ldots J$$

```
> M_0 <- lmer(Reaction ~ Days + (1|Subject),
+             data = sleepstudy)
```

Random slopes regression

- A random slopes model of this data would be

$$y_i \sim N(a + b_{s_i} x_i, \sigma^2), \quad \text{for all } i \in 1, 2 \ldots n$$
$$b_j \sim N(\beta, \tau_\beta^2), \quad \text{for all } j \in 1, 2 \ldots J$$

```
> M_1 <- lmer(Reaction ~ Days + (0 + Days|Subject),
+             data = sleepstudy)
```

Random slopes and random intercepts

$$
\begin{aligned}
y_i \sim N(a_{s_i} + b_{s_i} x_i, \sigma^2), &\quad \text{for all } i \in 1, 2 \ldots n \\
a_j \sim N(\alpha, \tau_\alpha^2), &\quad \text{for all } j \in 1, 2 \ldots J \\
b_j \sim N(\beta, \tau_\beta^2), &\quad \text{for all } j \in 1, 2 \ldots J
\end{aligned}
$$

```
> M_1 <- lmer(Reaction ~ Days + (1 + Days|Subject),
+             data = sleepstudy)
```

Nested multilevel models

- Sometimes we have groups nested in other groups.
- In `science_df`, we have `class`, with values $\{1, 2, 3, 4\}$, nested in `school`, with values $\{1, 2 \ldots 41\}$.
- To model this nesting, we'd do the following:

```
> M_1 <- lmer(like ~ sex + PrivPub + (1|school/class),
+             data = science_df)
```

which is identical to

```
> M_2 <- lmer(like ~ sex + PrivPub + (1|school) + (1|school:clas
+             data = science_df)
```

Nested models

- If we use unique identifiers for `class`, i.e. `Class`, which takes values `1.1`, `1.2`, etc., then we can simply do

```
> M_3 <- lmer(like ~ sex + PrivPub + (1|school) + (1|Class),
+             data = science_df)
```

Crossed structures

- When grouping variables are not nested, they are *crossed*.

```
> M <- lmer(diameter ~ 1 + (1|plate) + (1|sample),
+           data=penicillin_df)
```

Model comparison

We proceed just like in the case of generalized linear models.

```
> M_null <- lmer(diameter ~ 1 + (1|sample),
+                data=penicillin_df)
>
> anova(M_null, M)
#> Data: penicillin_df
#> Models:
#> M_null: diameter ~ 1 + (1 | sample)
#> M: diameter ~ 1 + (1 | plate) + (1 | sample)
#>        Df    AIC    BIC  logLik deviance Chisq Chi Df Pr(>Chi
#> M_null  3 443.19 452.10 -218.59   437.19
#> M       4 340.19 352.07 -166.09   332.19   105      1  < 2.2e
#> ---
#> Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' '
```

Introducing Brms

- Brms is an R package doing Bayesian general and generalized linear models, and general and generalized multilevel variants.
- To understand why Brms is so valuable, we must understand the following facts:
  1. Bayes is best. No further discussion necessary.
  2. Doing Bayesian data analysis, except for when using a prohibitively small set of models, requires Markov Chain Monte Carlo (MCMC) samplers.
  3. Writing your own MCMC is either hard or very hard.
  4. Probabilistic programming languages like Stan essentially write your MCMC sampler for any model you programmatically define.
  5. Although probabilistic programming languages reduce down the time and effort to obtain your sampler by orders of magnitude, they *still* require considerable time and effort relative to writing a single R command.
- Brms allow you to write your Bayesian model (with some restrictions) using standard R regression commands. It then writes Stan code, which then writes and compiles your sampler.

Major features

- Although ultimately more flexibility will be obtained using Stan, Brms is remarkably powerful:
- It includes far more probability models for outcome variables than almost all other regression packages: gaussian, student, binomial, bernoulli, poisson, negbinomial, geometric, Gamma, skew_normal, lognormal, shifted_lognormal, exgaussian, wiener, inverse.gaussian, exponential, weibull, frechet, Beta, von_mises, asym_laplace, gen_extreme_value, categorical, cumulative, cratio, sratio, acat, hurdle_poisson, hurdle_negbinomial, hurdle_gamma, hurdle_lognormal, zero_inflated_binomial, zero_inflated_beta, zero_inflated_negbinomial, zero_inflated_poisson, and zero_one_inflated_beta.
- It also allows for censored data, missing data, measurment error, nonlinear regression, probabilistic mixture models, *distributional* models (whereby all parameters of the outcome variables have predictors), and so on.

Brms example

```
> M <- lm(price ~ area + bedrooms, data = houseprice_df)
> library(brms)
> M_bayes <- brm(price ~ area + bedrooms, data = houseprice_df)
#>
#> SAMPLING FOR MODEL 'f6c53ab6b1cd1c57e71789bc84193f5e' NOW (CH
#> Chain 1:
#> Chain 1: Gradient evaluation took 1.1e-05 seconds
#> Chain 1: 1000 transitions using 10 leapfrog steps per transit
#> Chain 1: Adjust your expectations accordingly!
#> Chain 1:
#> Chain 1:
#> Chain 1: Iteration:    1 / 2000 [  0%]  (Warmup)
#> Chain 1: Iteration:  200 / 2000 [ 10%]  (Warmup)
#> Chain 1: Iteration:  400 / 2000 [ 20%]  (Warmup)
#> Chain 1: Iteration:  600 / 2000 [ 30%]  (Warmup)
#> Chain 1: Iteration:  800 / 2000 [ 40%]  (Warmup)
#> Chain 1: Iteration: 1000 / 2000 [ 50%]  (Warmup)
#> Chain 1: Iteration: 1001 / 2000 [ 50%]  (Sampling)
#> Chain 1: Iteration: 1200 / 2000 [ 60%]  (Sampling)
#> Chain 1: Iteration: 1400 / 2000 [ 70%]  (Sampling)
```