An Introduction to R[1]

Mark Andrews
Psychology Department, Nottingham Trent University

@xmjandrews
mark.andrews@ntu.ac.uk
https://github.com/mark-andrews/intro-to-R-sept-2019

[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.

What is R and why should you care

- R is a program for doing statistics and data analysis.
- R's advantages or selling points relative to other programs (e.g, SPSS, SAS, Stata, Minitab, Python, Matlab, Maple, Mathematica, Tableau, Excel, SQL, and many others) come down to three inter-related factors:
  - It is immensely powerful.
  - It is open-source.
  - It is very and increasingly widely used.

R: A power tool for data analysis

The range and depth of statistical analyses and general data analyses that can be accomplished with R is immense.

- Built into R are virtually the entire repertoire of widely known and used statistical methods.
- Also built in to R is an extensive graphics library.
- R has a vast set of add-on or contributed packages. There are presently 14871 additional contributed packages.
- R is a programming language that is specialized to efficiently manipulate and perform calculations on data.
- The R programming language itself can be extended by interfacing with other programming languages like C, C++, Fortran, Python, and high performance computing or big data tools like Hadoop, Spark, SQL.

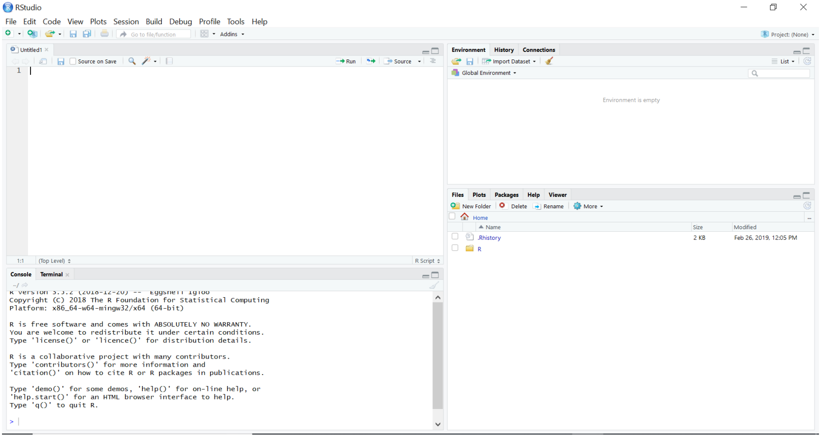
R: Open source software

- R is free and open source software, distributed according to the GNU public license.
- Likewise, virtually all of contributed R packages are likewise free and open source.
- In practical terms, this means that is freely available for everyone to use, now and forever, on more or less any device they choose.
- Open source software always has the potential to *go viral* and develop a large self-sustaining community of user/developers. This has arguably happened with R.

R: Popularity and widespread use

- When it comes to the computational implementation of modern statistical methods, R is the de facto standard. For example, the Journal of Statistical Software is overwhelmingly dominated by programs written in R.
- R is also currently very highly ranked according to many rankings of widely used programming languages of any kind. It ranked in the top 10 or top 20 most widely used programming languages.
- R is ranked as one of the top five most popular data science programs in jobs for data scientists, and in multiple surveys of data scientists, it is often ranked as the first or second mostly widely used data science tool.

A guided tour of RStudio



Introducing R commands

- A useful way to think about R, and not an inaccurate one either, is that it is simply a calculator.

```
> 2 + 2 # addition
#> [1] 4
> 3 - 5 # subtraction
#> [1] -2
> 3 * 2 # multiplication
#> [1] 6
> 4 / 3 # division
#> [1] 1.333333
> (2 + 2) ^ (3 / 3.5) # exponents and brackets
#> [1] 3.281341
```

Variables and assignment

- If we type the following at the command prompt and then press Enter, the result is displayed but not stored.

```
> (12/3.5)^2 + (1/2.5)^3 + (1 + 2 + 3)^0.33
#> [1] 13.6254
```

- We can, however, assign the value of the above calculation to a variable named `x`.

```
> x <- (12/3.5)^2 + (1/2.5)^3 + (1 + 2 + 3)^0.33
```

- Now, we can use `x` as is it were a number.

```
> x
#> [1] 13.6254
> x ^ 2
#> [1] 185.6516
> x * 3.6
#> [1] 49.05145
```

Assignment rules

- In general, the assignment rule is

```
name <- expression
```

The `expression` is any R code that returns some value.

- The `name` must consist of letters, numbers, dots, and underscores.

```
x123   # acceptable
.x
x_y_z
xXx_123
```

- It must begin with a letter or a dot that is not followed by a number.

```
_x   # not acceptable
.2x
x-y-z
```

- The recommendation is to use names that are meaningful, relatively short, without dots (using _ instead for punctuation), and primarily consisting of lowercase characters.

Vectors

- Vectors are one dimensional sequences of values.
- For example, if we want to create a vector of the first 6 primes numbers, we could do the following.

```
> primes <- c(2, 3, 5, 7, 11, 13)
```

- We can now perform operations (arithmetic, logical, etc) on the `primes` vector.

```
> primes + 1
#> [1]  3  4  6  8 12 14
> primes / 2
#> [1] 1.0 1.5 2.5 3.5 5.5 6.5
> primes == 3
#> [1] FALSE  TRUE FALSE FALSE FALSE FALSE
> primes >= 7
#> [1] FALSE FALSE FALSE  TRUE  TRUE  TRUE
```

Functions

- In functions, we put data in, calculations or done to or using this data, and new data, perhaps just a single value, is then returned.
- There are probably hundreds of thousands of functions in R.
- For example,

```
> length(primes)
#> [1] 6
> sum(primes)
#> [1] 41
> mean(primes)
#> [1] 6.833333
> median(primes)
#> [1] 6
> sd(primes)
#> [1] 4.400758
> var(primes)
#> [1] 19.36667
```

Writing R scripts

- Scripts are files where we write R commands, which can be then saved for later use.
- You can bring up RStudio's script editor with Ctrl+Shift+N, or go to the File/ New File/ R script, or click on the New icon on the left of the taskbar below the menu and choose R script.
- In a script, you can have as many lines of code as you wish, and there can be as many blank lines as you wish.

```
1 composites <- c(4, 6, 8, 9, 10, 12)
2
3 composites_plus_one <- composites + 1
4
5 composites_minus_one <- composites - 1
```

- If you place the cursor on line 1, you can then click the Run icon, or press the Ctrl+Enter keys.

Reading in data

- R allows you to import data from a very large variety of data file types, including from other statistics programs like SPSS, Stata, SAS, Minitab, and so on, and common file formats like `.xlsx` and `.csv`.
- When learning R initially, the easiest way to import data is using the Import Dataset button in the Environment window.
- If we use the *From Text (readr)...* option, it runs the `read_csv` R command, which we can run ourselves on the command line, or write in a script.

```
> library(tidyverse)
> weight_df <- read_csv("data/weight.csv")
```

Viewing data

- The easiest way to view a data frames is to type its name.

```
> weight_df
#> # A tibble: 6,068 x 7
#>    subjectid gender height weight handedness   age race
#>        <dbl> <chr>   <dbl>  <dbl> <chr>      <dbl> <chr>
#>  1     10027 male     178.   81.5 right         41 white
#>  2     10032 male     170.   72.6 left          35 white
#>  3     10033 male     174.   92.9 left          42 black
#>  4     10092 male     166.   79.4 right         31 white
#>  5     10093 male     191.   94.6 right         21 black
#>  6     10115 male     172    80.2 right         39 white
#>  7     10117 male     181   116.  right         32 black
#>  8     10237 male     185    95.4 right         23 white
#>  9     10242 male     178.   99.5 right         36 white
#> 10     10244 male     181.   70.2 left          23 white
#> # ... with 6,058 more rows
```

Viewing data (continued)

- Another option to view a data frame is to `glimpse` it.

```
> glimpse(weight_df)
#> Observations: 6,068
#> Variables: 7
#> $ subjectid  <dbl> 10027, 10032, 10033, 10092, 10093, 10115,
#> $ gender     <chr> "male", "male", "male", "male", "male", "m
#> $ height     <dbl> 177.6, 170.2, 173.5, 165.5, 191.4, 172.0,
#> $ weight     <dbl> 81.5, 72.6, 92.9, 79.4, 94.6, 80.2, 116.2,
#> $ handedness <chr> "right", "left", "left", "right", "right",
#> $ age        <dbl> 41, 35, 42, 31, 21, 39, 32, 23, 36, 23, 32
#> $ race       <chr> "white", "white", "black", "white", "black
```

Summarizing data with summary

- An easy way to summarize a data frame is with `summary`.

```
> summary(weight_df)
#>    subjectid         gender              height          weig
#>  Min.   : 10027   Length:6068        Min.   :140.9   Min.   :
#>  1st Qu.: 14842   Class :character   1st Qu.:165.2   1st Qu.:
#>  Median : 20064   Mode  :character   Median :171.9   Median :
#>  Mean   : 20757                      Mean   :171.4   Mean   :
#>  3rd Qu.: 27234                      3rd Qu.:177.9   3rd Qu.:
#>  Max.   :920103                      Max.   :199.3   Max.   :
#>   handedness             age             race
#>  Length:6068        Min.   :17.00   Length:6068
#>  Class :character   1st Qu.:23.00   Class :character
#>  Mode  :character   Median :28.00   Mode  :character
#>                     Mean   :29.76
#>                     3rd Qu.:36.00
#>                     Max.   :58.00
```

Summarizing with `summarize`

- The `summarize` (or `summarise`) function allows us to calculate summary statistics.

```
> summarize(weight_df,
+           mean_weight = mean(weight),
+           median_weight = median(weight),
+           sd_weight = sd(weight)
+ )
#> # A tibble: 1 x 3
#>   mean_weight median_weight sd_weight
#>         <dbl>         <dbl>     <dbl>
#> 1        79.7          78.5      15.7
```

Summarizing with `summarize` *and* `group_by`

- Combined with `group_by`, `summarize` allows us to calculate summary statistics by group

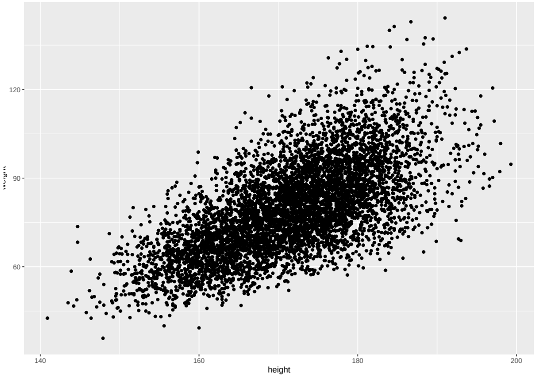
```
> summarize(group_by(weight_df, gender),
+           mean_weight = mean(weight),
+           median_weight = median(weight),
+           sd_weight = sd(weight)
+ )
#> # A tibble: 2 x 4
#>   gender mean_weight median_weight sd_weight
#>   <chr>        <dbl>         <dbl>     <dbl>
#> 1 female        67.8          66.8      11.0
#> 2 male          85.5          84.6      14.2
```

Plots and data visualiztion

- The best way to data visualization in R is with `ggplot2`, which is part of the `tidyverse`.
- `ggplot2` is a package whose main function is `ggplot`.
- `ggplot` is a *layered* plotting system where we map variables to aesthetic properties of a graphic and then add layers.

Scatterplot

```
> ggplot(weight_df,
+        aes(x = height, y = weight)
+ ) + geom_point()
```

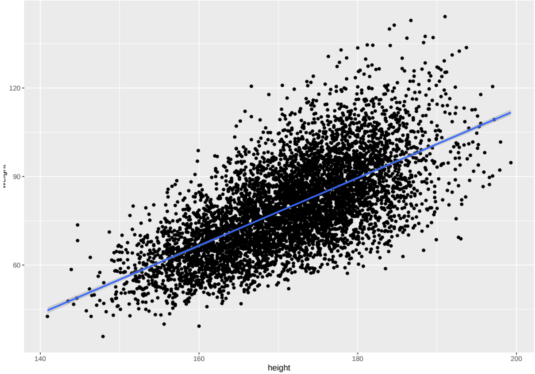


Scatterplot with `gender` indicated by colour

```
> ggplot(weight_df,
+        aes(x = height, y = weight, col = gender)
+ ) + geom_point()
```

Scatterplot with line of best fit

```
> ggplot(weight_df,
+        aes(x = height, y = weight)
+ ) + geom_point() +
+   stat_smooth(method = 'lm')
```

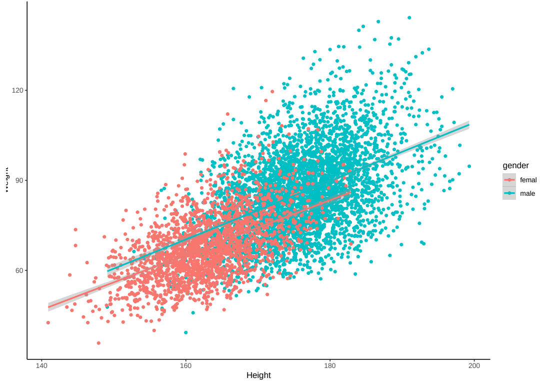


Scatterplot with line of best fit, for each value of `gender`

```
> ggplot(weight_df,
+        aes(x = height, y = weight, col = gender)
+ ) + geom_point() + stat_smooth(method = 'lm')
```

Changing style of a plot

```
> ggplot(weight_df,
+         aes(x = height, y = weight, col = gender)
+ ) + geom_point() + stat_smooth(method = 'lm') +
+   xlab('Height') +
+   ylab('Weight') +
+   theme_classic()
```



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
```

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 = 12535580542, p-value < 2.2e-16
#> alternative hypothesis: true rho is not equal to 0
#> sample estimates:
#>       rho
#> 0.6633652
```

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
```

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
```

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)
> M
#> Call:
#>    aov(formula = weight ~ group, data = PlantGrowth)
#>
#> 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)
#>
#> $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 ' '
```