# SOC-5811 Week 3: Introduction to R and ggplot2

Nick Graetz

University of Minnesota, Department of Sociology

9/15/2025

# IN THIS LECTURE

1. What is R?
2. RStudio interface
3. Packages
4. R as calculator
5. Anatomy of a function
6. Help files
7. R scripts

# WHAT IS R?

- R is a language for statistical computing and graphics
- Originally developed in 1992 by Robert Gentleman and Ross Ihaka based on the programming language S
- The core of the R language is maintained by the R Core Team
- A (very) large number of packages which add additional functionality are maintained by other contributors

# Why use R?

- R can do many useful things
  - Flexible data management
  - Powerful statistical capabilities, particularly for modeling
  - Extensive graphics capabilities

# Why use R?

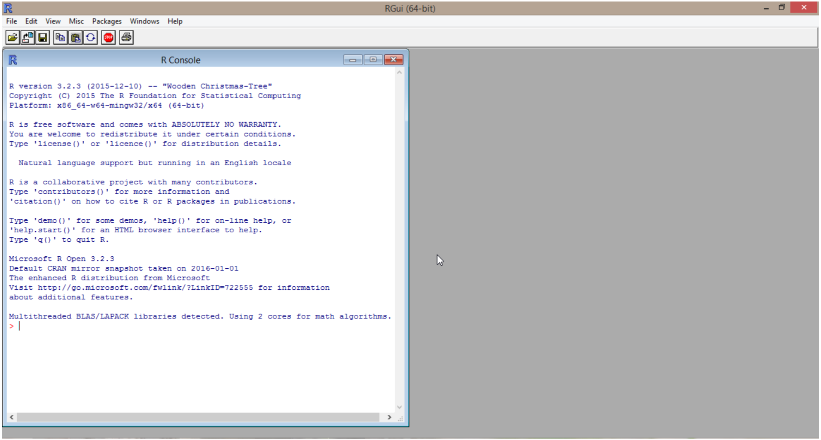
- R can do many useful things
  - Flexible data management
  - Powerful statistical capabilities, particularly for modeling
  - Extensive graphics capabilities

- R is free software
  - You don't have to pay for it (and you can share it with anyone)
  - You can use and modify it as you see fit

# Why use R?

- R can do many useful things
  - Flexible data management
  - Powerful statistical capabilities, particularly for modeling
  - Extensive graphics capabilities
- R is free software
  - You don't have to pay for it (and you can share it with anyone)
  - You can use and modify it as you see fit
- R has a large (and enthusiastic) user base
  - This makes finding help relatively straightforward
  - New methods are often implemented in R very quickly

# R (GUI) INTERFACE



```
R version 3.2.3 (2015-12-10) -- "Wooden Christmas-Tree"
Copyright (C) 2015 The R Foundation for Statistical Computing
Platform: x86_64-w64-mingw32/x64 (64-bit)

R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

  Natural language support but running in an English locale

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

Microsoft R Open 3.2.3
Default CRAN mirror snapshot taken on 2016-01-01
The enhanced R distribution from Microsoft
Visit http://go.microsoft.com/fwlink/?LinkID=722555 for information
about additional features.

Multithreaded BLAS/LAPACK libraries detected. Using 2 cores for math algorithms.
> |
```

# What is RStudio?

- "Integrated development environment" (IDE)

# What is RStudio?

- “Integrated development environment” (IDE)
- Convenient interface for R which incorporates a number of useful features for developing code
  - syntax highlighting
  - code completion
  - code navigation
  - debugging tools
  - etc.

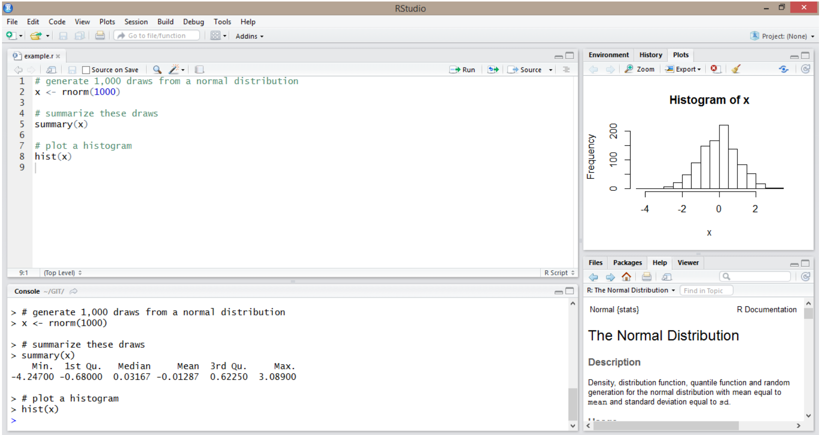
# What is RStudio?

- "Integrated development environment" (IDE)

- Convenient interface for R which incorporates a number of useful features for developing code
  - syntax highlighting
  - code completion
  - code navigation
  - debugging tools
  - etc.

- Also provides integration with other useful tools
  - Shiny (for developing web apps)
  - R Markdown (for authoring documents and slides)
  - Git/Subversion (for version control)

# RSTUDIO INTERFACE



```
# generate 1,000 draws from a normal distribution
x <- rnorm(1000)

# summarize these draws
summary(x)

# plot a histogram
hist(x)
```

```
> # generate 1,000 draws from a normal distribution
> x <- rnorm(1000)

> # summarize these draws
> summary(x)
    Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
-4.24700 -0.68000  0.03167 -0.01287  0.62250  3.08900

> # plot a histogram
> hist(x)
>
```

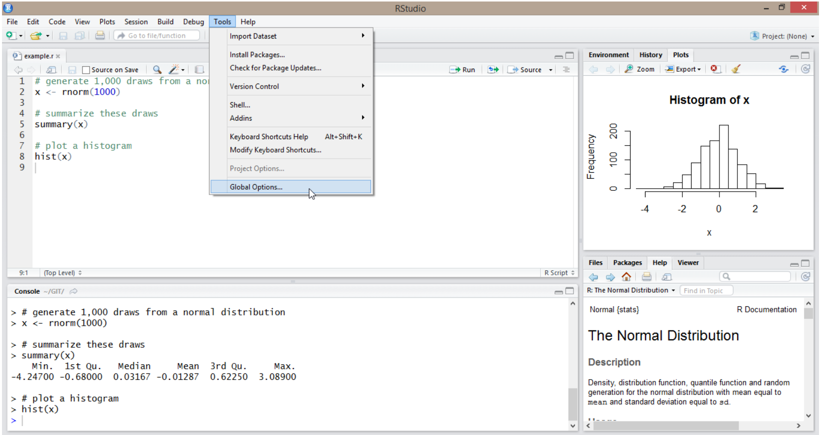
Normal {stats} R Documentation

## The Normal Distribution

### Description

Density, distribution function, quantile function and random generation for the normal distribution with mean equal to mean and standard deviation equal to sd.

# RStudio interface



```
# generate 1,000 draws from a no
x <- rnorm(1000)

# summarize these draws
summary(x)

# plot a histogram
hist(x)
```

```
> # generate 1,000 draws from a normal distribution
> x <- rnorm(1000)

> # summarize these draws
> summary(x)
    Min.   1st Qu.    Median      Mean   3rd Qu.      Max. 
-4.24700 -0.68000  0.03167 -0.01287  0.62250  3.08900 

> # plot a histogram
> hist(x)
> 
```

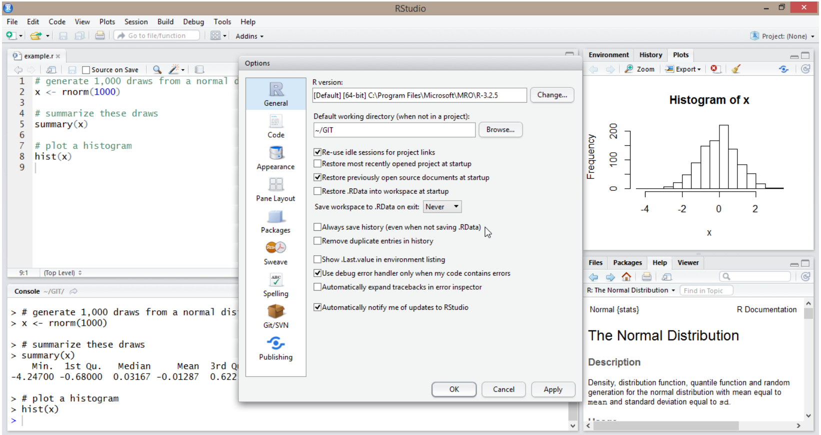
Normal {stats} R Documentation

The Normal Distribution

Description

Density, distribution function, quantile function and random generation for the normal distribution with mean equal to mean and standard deviation equal to sd.

# RStudio interface



```
# generate 1,000 draws from a normal d
x <- rnorm(1000)

# summarize these draws
summary(x)

# plot a histogram
hist(x)
```

```
> # generate 1,000 draws from a normal dis
> x <- rnorm(1000)

> # summarize these draws
> summary(x)
    Min.  1st Qu.   Median     Mean  3rd Q
-4.24700 -0.68000  0.03167 -0.01287  0.622

> # plot a histogram
> hist(x)
>
```

R version:
[Default] [64-bit] C:\Program Files\Microsoft\MRO\R-3.2.5

Default working directory (when not in a project):
~/GIT

☑ Re-use idle sessions for project links
☐ Restore most recently opened project at startup
☑ Restore previously open source documents at startup
☐ Restore .RData into workspace at startup

Save workspace to .RData on exit: Never

☐ Always save history (even when not saving .RData)
☐ Remove duplicate entries in history

☐ Show .Last.value in environment listing
☑ Use debug error handler only when my code contains errors
☐ Automatically expand tracebacks in error inspector

☑ Automatically notify me of updates to RStudio

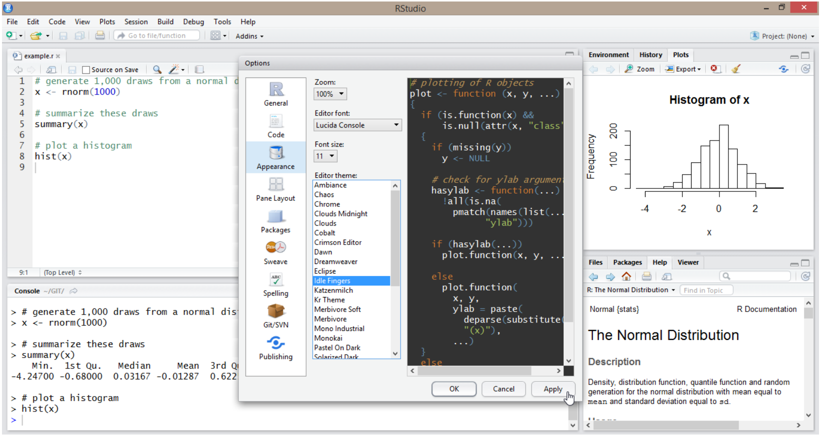
Normal {stats} R Documentation

The Normal Distribution

Description

Density, distribution function, quantile function and random generation for the normal distribution with mean equal to mean and standard deviation equal to sd.

# RStudio interface



```
# generate 1,000 draws from a normal d
x <- rnorm(1000)

# summarize these draws
summary(x)

# plot a histogram
hist(x)
```

9:1 (Top Level)

Zoom: 100%

Editor font: Lucida Console

Font size: 11

Editor theme:

- Ambiance
- Chaos
- Chrome
- Clouds Midnight
- Clouds
- Cobalt
- Crimson Editor
- Dawn
- Dreamweaver
- Eclipse
- Idle Fingers
- Katzenmilch
- Kr Theme
- Merbivore Soft
- Merbivore
- Mono Industrial
- Monokai
- Pastel On Dark
- Solarized Dark

```
# plotting of R objects
plot <- function (x, y, ...)
{
  if (is.function(x) &&
      is.null(attr(x, "class"
  {
    if (missing(y))
      y <- NULL

    # check for ylab argumen
    hasylab <- function(...)
      !all(is.na(
        pmatch(names(list(...
               "ylab")))

    if (hasylab(...))
      plot.function(x, y, ...

    else
      plot.function(
        x, y,
        ylab = paste(
          deparse(substitute(
          "(x)"),
        ...)
  }
  else
```

Console ~/GIT/

```
> # generate 1,000 draws from a normal dis
> x <- rnorm(1000)

> # summarize these draws
> summary(x)
    Min.  1st Qu.   Median      Mean  3rd Qu
-4.24700 -0.68000  0.03167 -0.01287  0.622
                                  
> # plot a histogram
> hist(x)
> 
```

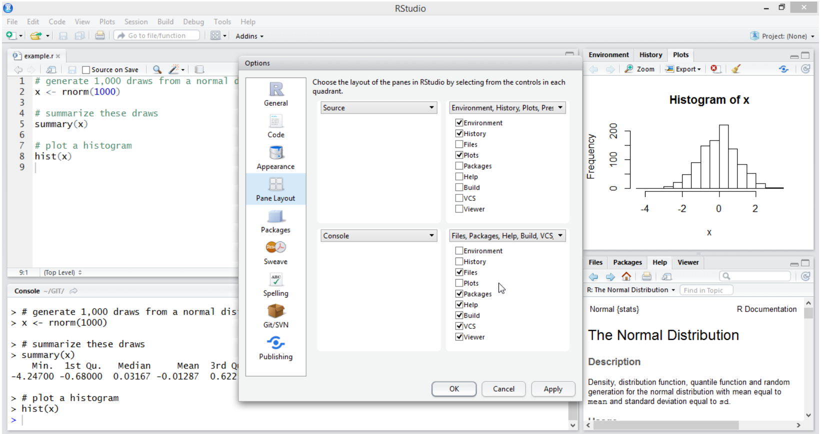
Normal {stats}    R Documentation

The Normal Distribution

Description

Density, distribution function, quantile function and random generation for the normal distribution with mean equal to mean and standard deviation equal to sd.

# RStudio interface



```r
# generate 1,000 draws from a normal d
x <- rnorm(1000)

# summarize these draws
summary(x)

# plot a histogram
hist(x)
```

```
> # generate 1,000 draws from a normal dis
> x <- rnorm(1000)

> # summarize these draws
> summary(x)
    Min.  1st Qu.   Median     Mean  3rd Q
-4.24700 -0.68000  0.03167 -0.01287  0.622

> # plot a histogram
> hist(x)
>
```

Choose the layout of the panes in RStudio by selecting from the controls in each quadrant.

Source

Environment, History, Plots, Pres

- ☑ Environment
- ☑ History
- □ Files
- ☑ Plots
- □ Packages
- □ Help
- □ Build
- □ VCS
- □ Viewer

Console

Files, Packages, Help, Build, VCS,

- □ Environment
- □ History
- ☑ Files
- □ Plots
- ☑ Packages
- ☑ Help
- ☑ Build
- ☑ VCS
- ☑ Viewer

Normal {stats} R Documentation

The Normal Distribution

Description

Density, distribution function, quantile function and random generation for the normal distribution with mean equal to `mean` and standard deviation equal to `sd`.

# Packages

Most basic R functionality is part of `base` and is loaded automatically when you start R. Additional functionality can be added through packages.

# Packages

Most basic R functionality is part of `base` and is loaded automatically when you start R. Additional functionality can be added through packages.

The first time you use a package, it needs to be installed:

```
> install.packages("ggplot2")
```

After that, you just need to load the package using the `library()` command whenever you start a new instance of R:

```
> library(ggplot2)
```

# R AS CALCULATOR

R can be used as a calculator by just typing in the console.

All of the basic arithmetic operators (+, -, *, /, ^) do what you would expect them to do, following normal order of operations conventions:

```
> 230 + 97
[1] 327
> 500/20
[1] 25
```

# R AS CALCULATOR

Parentheses can be used to alter the order of operations:

```
> 300/20^1/2
[1] 7.5
> (300/20)^(1/2)
[1] 3.872983
```

# R as calculator: Quick exercise

1. How many seconds are in September?
2. What is 80 degrees Fahrenheit in degrees Celsius?
3. How much longer is 1 mile than 1600 meters (in feet)?

# R as calculator: Quick exercise

1. How many seconds are there in September?

```
> 30 * 24 * 60 * 60
[1] 2592000
```

2. What is 80 degrees Fahrenheit in degrees Celsius?

```
> (80 - 32) * (5/9)
[1] 26.66667
```

3. How much longer is 1 mile than 1600 meters (in feet)?

```
> 5280 - 1600 * 3.28084
[1] 30.656
```

# Functions

R functions are used to transform input into output in some way.

For example…

```
> log(10)
[1] 2.302585
```

```
> exp(3)
[1] 20.08554
```

```
> sqrt(80)
[1] 8.944272
```

# Functions: anatomy

```
> log(x = 300, base = 10)
[1] 2.477121
```

1. Function name: **`log()`**
2. Argument name(s): **`x, base`**
3. Argument value(s): **`300, 10`**
4. Output: **`2.4771213`**

# Functions: argument order

Arguments can be specified in any order *if they are named*:

```
> log(x = 300, base = 10)
[1] 2.477121
```

```
> log(base = 10, x = 300)
[1] 2.477121
```

# Functions: argument names

Arguments don’t need to be named, but then *there is only one correct order*:

```
> log(x = 300, base = 10)
[1] 2.477121
> log(base = 10, x = 300)
[1] 2.477121
```

```
> log(300, 10)
[1] 2.477121
> log(10, 300)
[1] 0.4036944
```

# Functions: defaults

Some (but not all) arguments have defaults and don't need to be specified, assuming you are happy with the default:

```
> log(x = 300)
[1] 5.703782
```

```
> log(base = 10)
Error: argument "x" is missing, with no default
```

# FUNCTIONS: COMBINING

Functions can be combined or nested with other functions and operators:

```
> exp(log(10) + log(10))
[1] 100
```

```
> log(x = (4 * 10)/7, base = 10)
[1] 0.756962
```

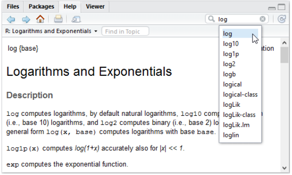
# Help files

Every function has a help file.

You can access a help file from the console:

```
> help(log)
```

or from the help tab in RStudio:



log {base}

ation

## Logarithms and Exponentials

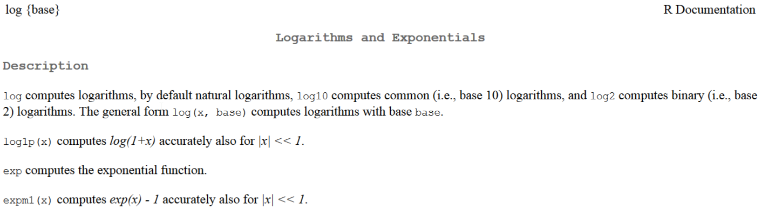
### Description

`log` computes logarithms, by default natural logarithms, `log10` comp
(i.e., base 10) logarithms, and `log2` computes binary (i.e., base 2) lo
general form `log(x, base)` computes logarithms with base `base`.

`log1p(x)` computes *log(1+x)* accurately also for |*x*| << *1*.

`exp` computes the exponential function.

# HELP FILES



log {base} R Documentation

### Logarithms and Exponentials

#### Description

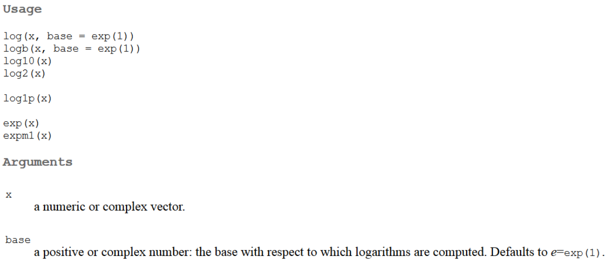
`log` computes logarithms, by default natural logarithms, `log10` computes common (i.e., base 10) logarithms, and `log2` computes binary (i.e., base 2) logarithms. The general form `log(x, base)` computes logarithms with base `base`.

`log1p(x)` computes $log(1+x)$ accurately also for $|x| << 1$.

`exp` computes the exponential function.

`expm1(x)` computes $exp(x) - 1$ accurately also for $|x| << 1$.

# Help files



### Usage

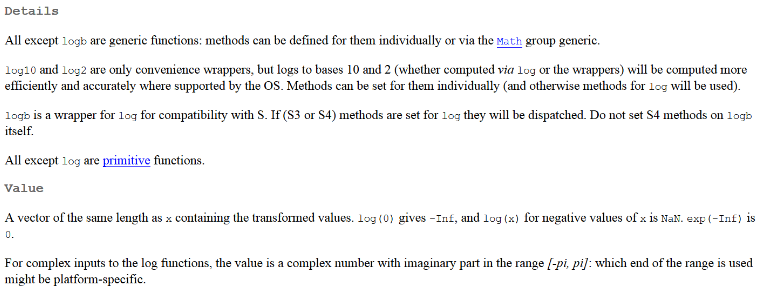
```
log(x, base = exp(1))
logb(x, base = exp(1))
log10(x)
log2(x)

log1p(x)

exp(x)
expm1(x)
```

### Arguments

`x`
: a numeric or complex vector.

`base`
: a positive or complex number: the base with respect to which logarithms are computed. Defaults to *e*=`exp(1)`.

# HELP FILES



**Details**

All except `logb` are generic functions: methods can be defined for them individually or via the `Math` group generic.

`log10` and `log2` are only convenience wrappers, but logs to bases 10 and 2 (whether computed *via* `log` or the wrappers) will be computed more efficiently and accurately where supported by the OS. Methods can be set for them individually (and otherwise methods for `log` will be used).

`logb` is a wrapper for `log` for compatibility with S. If (S3 or S4) methods are set for `log` they will be dispatched. Do not set S4 methods on `logb` itself.

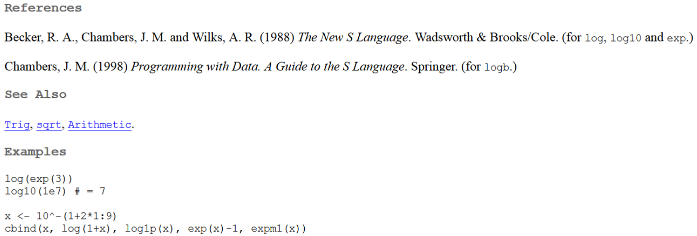
All except `log` are primitive functions.

**Value**

A vector of the same length as `x` containing the transformed values. `log(0)` gives `-Inf`, and `log(x)` for negative values of `x` is `NaN`. `exp(-Inf)` is `0`.

For complex inputs to the log functions, the value is a complex number with imaginary part in the range *[-pi, pi]*: which end of the range is used might be platform-specific.

# HELP FILES



### References

Becker, R. A., Chambers, J. M. and Wilks, A. R. (1988) *The New S Language*. Wadsworth & Brooks/Cole. (for `log`, `log10` and `exp`.)

Chambers, J. M. (1998) *Programming with Data. A Guide to the S Language*. Springer. (for `logb`.)

### See Also

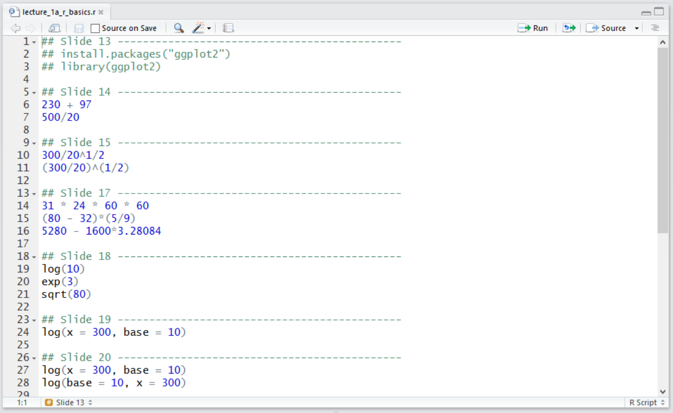
Trig, sqrt, Arithmetic.

### Examples

```
log(exp(3))
log10(1e7) # = 7

x <- 10^-(1+2*1:9)
cbind(x, log(1+x), log1p(x), exp(x)-1, expm1(x))
```

# R SCRIPTS

An R script is a text file (`.r` extension) with a series of R commands and (ideally) some useful commentary.



```r
## Slide 13 -------------------------------------------
## install.packages("ggplot2")
## library(ggplot2)

## Slide 14 -------------------------------------------
230 + 97
500/20

## Slide 15 -------------------------------------------
300/20^1/2
(300/20)^(1/2)

## Slide 17 -------------------------------------------
31 * 24 * 60 * 60
(80 - 32)*(5/9)
5280 - 1600*3.28084

## Slide 18 -------------------------------------------
log(10)
exp(3)
sqrt(80)

## Slide 19 -------------------------------------------
log(x = 300, base = 10)

## Slide 20 -------------------------------------------
log(x = 300, base = 10)
log(base = 10, x = 300)
```

# Why use a script?

Typing in the console is fine for quick calculations or experimentation with a command, but a script provides…

- a full record of all commands required to carry out an analysis
- a convenient mechanism for repeating an analysis without needing to retype everything (no need to reinvent the wheel)
- a starting point for writing new code
- a vehicle for providing context and commentary for your code

# Why use a script?

**Any analysis you do should be saved as a script!**

Without a script…

- you will forget what you’ve done
- you will forget why you did it
- no one else will ever know what you did or why you did it
- you will have do things over again for no reason

# Running a script

If your script is open in RStudio, you can run the whole thing using `ctrl + shift + enter` or just a single line (or highlighted block) using `ctrl + enter`.

Or you can run a script from the command line using the `source()` function:

```
> source("C:/Users/ngraetz/Dropbox/my_script.R")
```

# Commenting a script

R will ignore any line in a script that starts with #, so you can use this to add comments to your code:

```
> # add 1-5
> 1 + 2 + 3 + 4 + 5
[1] 15
>
> # find the natural log of 10
> log(10)
[1] 2.302585
```

# Commenting a script

Use comments to:

- Label blocks of code. This will help you navigate your code later
- Explain why you're doing something (if it's not self-evident)
- Write yourself (and other users) notes about particularly tricky lines of code

# Commenting a script

Use comments to:

- Label blocks of code. This will help you navigate your code later
- Explain why you're doing something (if it's not self-evident)
- Write yourself (and other users) notes about particularly tricky lines of code

You want to provide enough information so that your future self, or someone else, can quickly understand the structure and purpose of your code at a later date.

**However,** it is possible to provide too much information, making your code more cumbersome (e.g., writing out what each line of code does).

# Headers

It's also good practice to use '#' to provide some sort of header at the top of your code:

```
###################################################
## Author:      John Doe
##
## Description: A short description of what this co
##              and any important context for why.
##
## Output:      A list of files that are output by
##              code.
##
## Notes:       Anything someone should know when r
##              this code.
###################################################
```

# IN THIS LECTURE

1. Understanding the `ggplot` approach
2. Aesthetics
3. Geoms
4. Facets
5. Options and customization
6. Reshaping
7. Saving plots
8. Additional packages

# What is `ggplot2`?

`ggplot2` is an R package for making sophisticated and great-looking graphs

It's based on the book "Grammar of Graphics", which defined a fundamental theory of data visualization

`ggplot2` contains functions that allow you to build complex graphics using a relatively small set of building blocks

NOTE: the online documentation for `ggplot2` is fantastic, and lays all the functions out in terms of these building blocks:

http://ggplot2.tidyverse.org/reference/

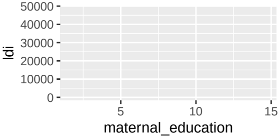
https://www.rstudio.com/wp-content/uploads/2015/03/ggplot2-cheatsheet.pdf

# Load libraries & data

# How does ggplot2 work?

First, you set up the graph:

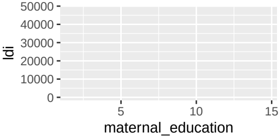
```
> ggplot(data = mmr_data, aes(x = maternal_education, y
```



# HOW DOES GGPLOT2 WORK?

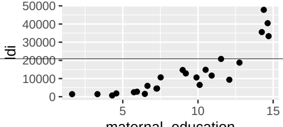
First, you set up the graph:

```
> ggplot(data = mmr_data, aes(x = maternal_education, y
```



Then, you add to it. Basically telling `ggplot` what type of graph to make:

```
> ggplot(data = mmr_data, aes(x = maternal_education, y
+    geom_point()
```



# What are the building blocks of a ggplot?

- **Aesthetics**
- **Geoms**
- **Facets**
- Positions
- Scales
- Labels
- Themes

# AESTHETICS

The aes in the initial ggplot() call

```
> ggplot(data = mmr_data, aes(x = maternal_educatio
+   geom_point()
```

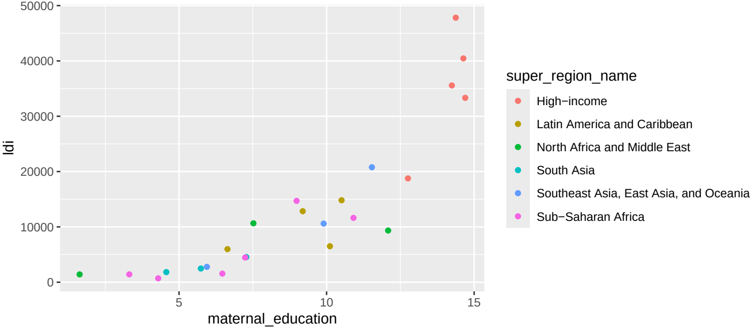
"Aesthetic mapping" is how you tell ggplot which variable is x, which is y

**But**, you can use them for more than just the axes:

- color (border color)
- fill (fill color)
- shape
- linetype (solid, dashed, dotted etc.)
- size
- alpha (transparency)
- labels

# EXAMPLE OF AESTHETIC MAPPING

```
> ggplot(data = mmr_data, aes(x = maternal_education, y
+     color = super_region_name)) +
+   geom_point()
```

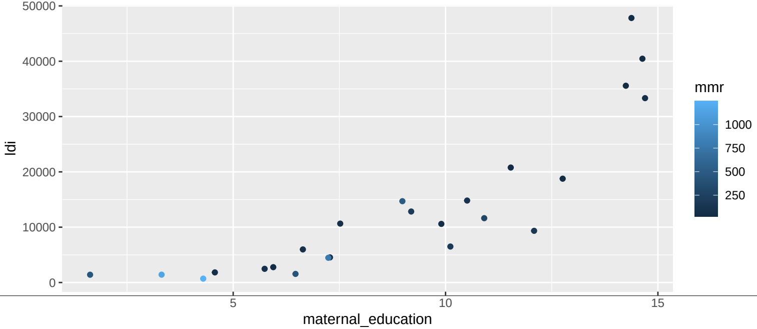


Note that `ggplot` conveniently makes a legend for you! In ggplot lingo, legends are called “scales”

# Example of aesthetic mapping

In many cases, aesthetic mapping works for both continuous and categorical data

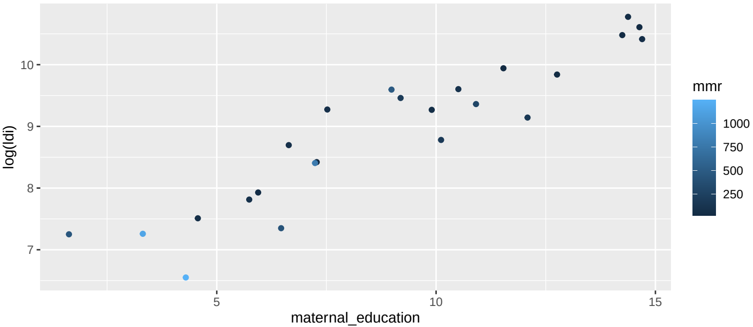
```
> ggplot(data = mmr_data, aes(x = maternal_educatio
+      color = mmr)) +
+    geom_point()
```



# Example of aesthetic mapping

ggplot allows you to manipulate variables “on the fly”:

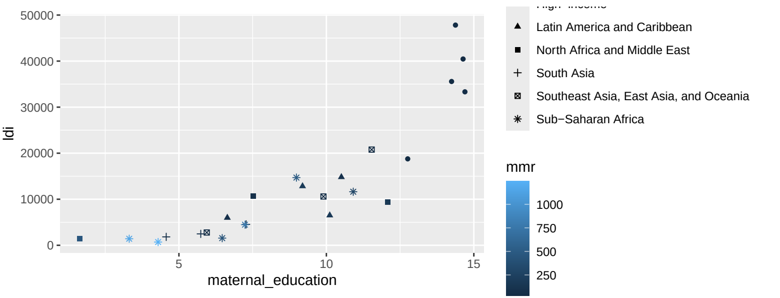
```
> ggplot(data = mmr_data, aes(x = maternal_educatio
+     y = log(ldi), color = mmr)) +
+   geom_point()
```



# Example of aesthetic mapping

You can keep adding more aesthetics to add more information to your graph:

```
> ggplot(data = mmr_data, aes(x = maternal_education, y
+       color = mmr, shape = super_region_name)) +
+    geom_point()
```



Note that not all aesthetics are meaningful for all geoms (e.g., linetype doesn't make sense if there are no lines in your graph)

# What are the building blocks of a ggplot?

- **Aesthetics**
- **Geoms**
- **Facets**
- Positions
- Scales
- Labels
- Themes

# Geoms

```
> ggplot(data = mmr_data, aes(x = maternal_educatio
+    geom_point()
```

ggplot "geoms" (geometries) are the different types of graphs you can make:

- geom_point() for scatter plots
- geom_line() for line graphs
- geom_bar() for bar graphs
- And more: geom_histogram(), geom_violin(), geomboxplot(), geom_errorbar(), geom_ribbon(), geom_segment(), geom_path(), geom_tile(), geom_polygon(), etc.

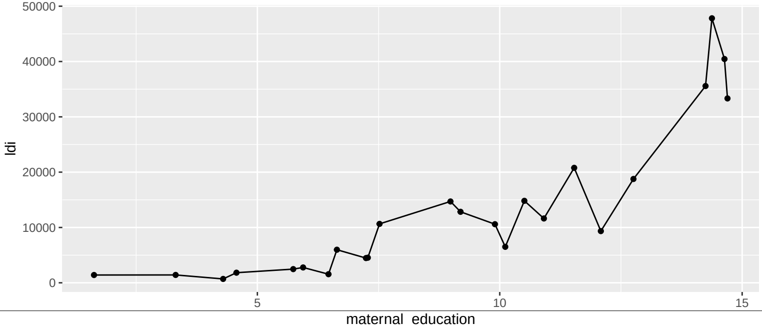
There are dozens of different geometries you can use for ggplot.

See the ggplot cheat sheet for the whole list: https://www.rstudio.com/wp-

# Geoms

If you specify more than one geom, it “layers” them on top of each other

```
> ggplot(data = mmr_data, aes(x = maternal_education, y
+   geom_point() +
+   geom_line()
```

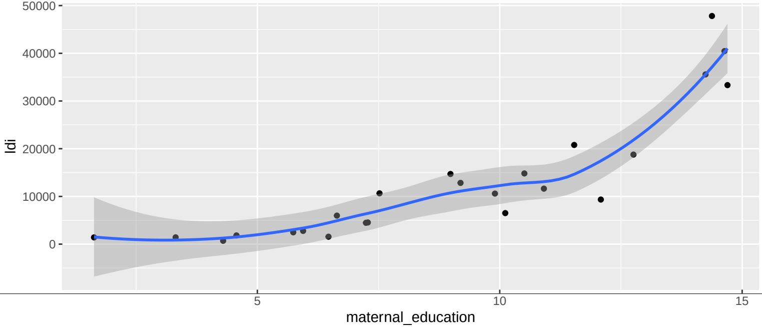


Note: the order matters, it will layer geoms in order that they

# Geoms

There are some special geoms that do computation for you on the fly, just for convenience

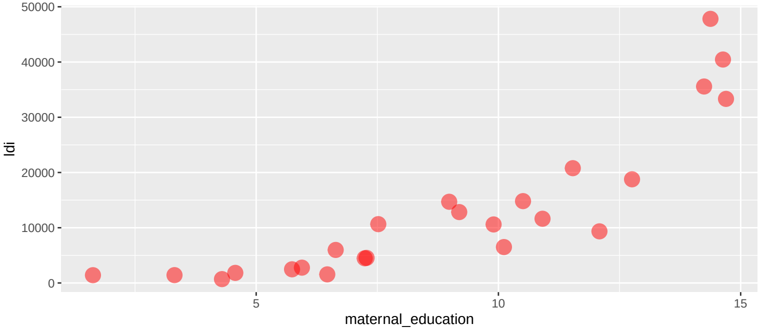
```
> ggplot(data = mmr_data, aes(x = maternal_educatio
+     geom_point() +
+     geom_smooth()
```



# Geoms

Aesthetic arguments can also be provided directly to a geom in cases where you don't want them to map to some variable

```
> ggplot(data = mmr_data, aes(x = maternal_educatio
+   geom_point(color = 'red', size = 2, alpha = .5)
```



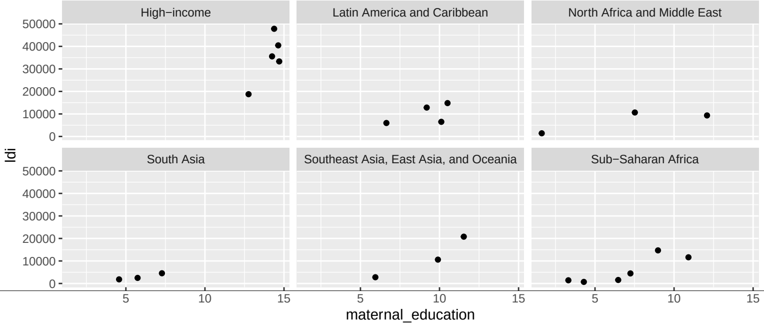
# What are the building blocks of a ggplot?

- **Aesthetics**
- **Geoms**
- **Facets**
- Positions
- Scales
- Labels
- Themes

# Facets

Facets allow you to incorporate more complexity into your graphs by adding multiple panels:

```
> ggplot(data = mmr_data, aes(x = maternal_educatio
+     geom_point() +
+     facet_wrap(~super_region_name)
```



# Facets

`facet_wrap` stratifies and wraps. You can specify `nrow` and `ncol` to modify dimensions.

`facet_grid` forms a grid of panels based on rown and column facetting variables. Example: `facet_grid(sex ~ age_group)` will create rows of panels based on sex and columns of panels based on age group.

# What are the building blocks of a ggplot?

- **Aesthetics**
- **Geoms**
- **Facets**
- Positions
- Scales
- Labels
- Themes

# Positions

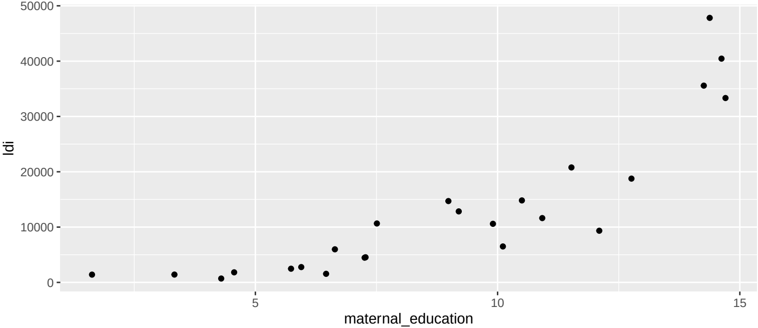
`ggplot` lets you modify where geoms appear relative to each other, using `position` functions:

- `position_jitter()` randomly displaces points (usually just for `geom_point`)
- `position_dodge()` automatically (tries to) shift to avoid overlap
- `position_stack()` stack, or add together geoms (usually just for `geom_bar`)
- `position_fill()` rescale the y-axis so the geoms sum to 100% (usually just for `geom_bar`)

# Positions

position_jitter randomly displaces points (usually just for geom_point)

```
> ggplot(data = mmr_data, aes(x = maternal_educatio
+     geom_point(position='jitter')
```

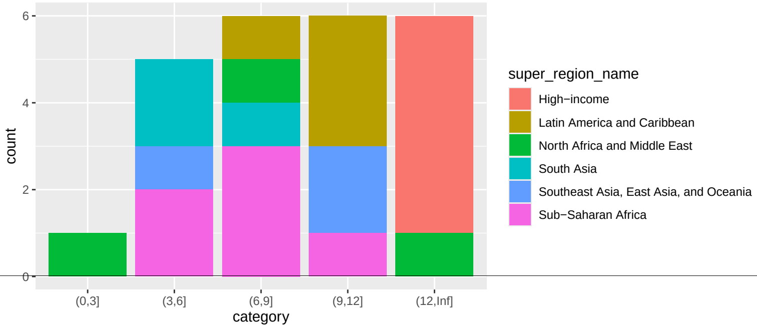


It's built right into the geom_point() function for

# Positions

`position_stack` is the default for `geom_bar` for factor variables

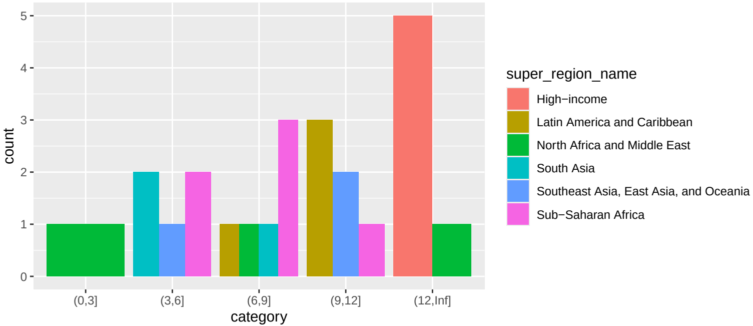
```
> mmr_data$category <- cut(mmr_data$maternal_education,
+                             breaks=c(0, 3, 6, 9, 12, Inf)
> ggplot(data = mmr_data, aes(x = category, fill = super
+   geom_bar()
```



# Positions

`position_dodge` would put the bars side-by-side

```
> ggplot(data = mmr_data, aes(x = category, fill = super
+    geom_bar(position = 'dodge')
```

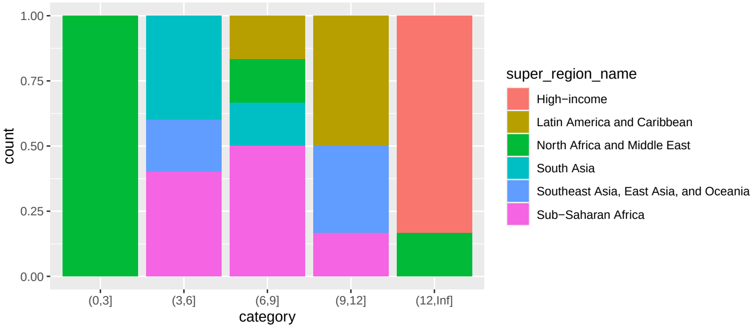


It's built right into the `geom_bar()` function for convenience

# Positions

`position_fill` makes the bars sum to 100%

```
> ggplot(data = mmr_data, aes(x = category, fill=super_r
+    geom_bar(position='fill')
```

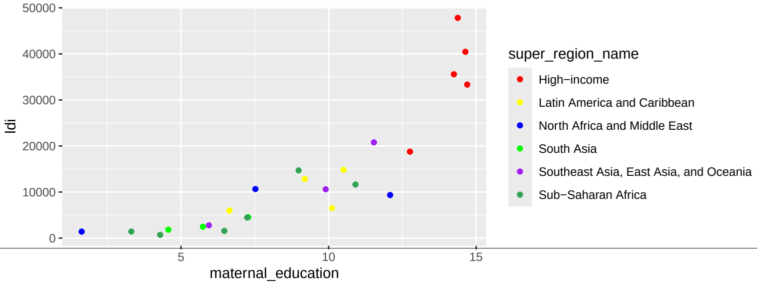


It's built right into the `geom_bar()` function for convenience

# Scales

You can also modify the “scales” (i.e., legends) to customize aesthetic mapping

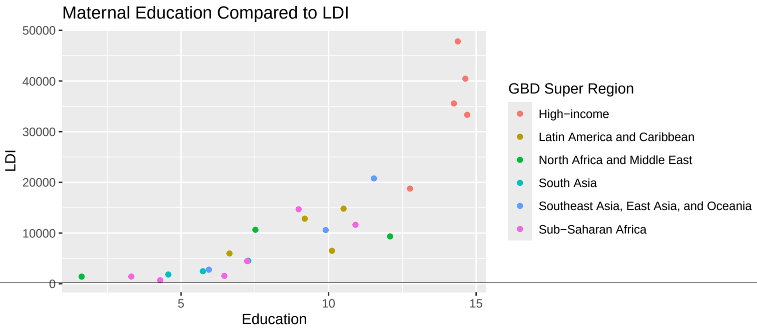
```
> ggplot(data = mmr_data, aes(x = maternal_education, y
+     color = super_region_name)) +
+   geom_point() +
+   scale_color_manual(values = c('red','yellow','blue',
+                                 'purple','#31a354'))
```



# Labels

Titles for everything can be added with the `labs()` function:

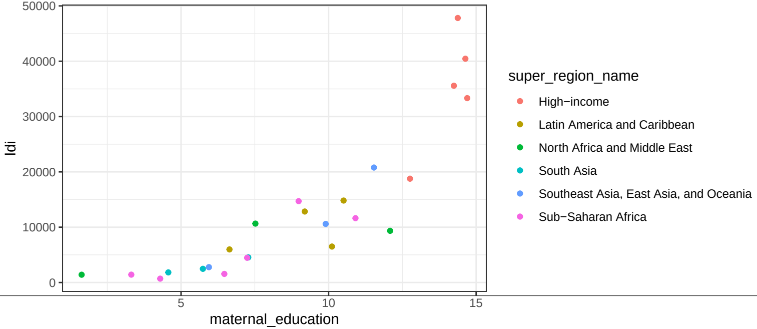
```
> ggplot(data = mmr_data, aes(x = maternal_education,  y
+      color = super_region_name)) +
+    geom_point() +
+    labs(title = 'Maternal Education Compared to LDI', y
+      x = 'Education', color = 'GBD Super Region')
```



# Themes

`ggplot` also comes with handy “themes”, or preset options:

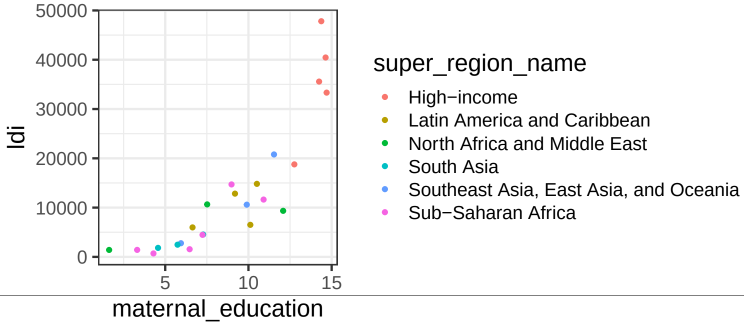
```
> ggplot(data = mmr_data, aes(x = maternal_educatio
+       color = super_region_name)) +
+     geom_point() +
+     theme_bw()
```



# THEMES

Themes also allow you to rescale all text at the same time

```
> ggplot(data = mmr_data, aes(x = maternal_educatio
+       color = super_region_name)) +
+     geom_point() +
+     theme_bw(base_size = 18)
```



# Reshaping

ggplot2 is designed to work with data shaped such that each desired aesthetic is mapped to **one** variable. If your data is not shaped this way, it's almost always easier to reshape the data than to try and make ggplot2 work with original data structure.

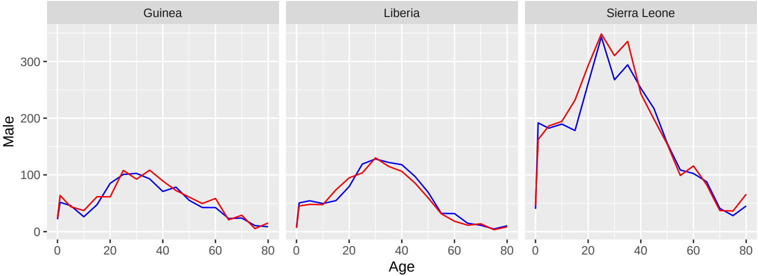
For example, if you want to plot the number of Ebola deaths by age group for both males and females, this is an inconvenient data structure since there are separate columns for deaths among males and females:

```
Key: <Country, Age>
   Country   Age Female  Male
    <char> <int>  <num> <num>
1:  Guinea     0   24.5  21.9
2:  Guinea     1   63.8  51.7
3:  Guinea     5   44.0  45.8
4:  Guinea    10   37.1  26.2
```

# Reshaping

One option is to just add different geoms for each variable:

```
> ggplot(wide_data, aes(x = Age, y = Male)) +
+   facet_wrap(~ Country) +
+   geom_line(color = 'blue') +
+   geom_line(data = wide_data, aes(y = Female))
```



# Reshaping

A better option is to reshape long before attempting to plot these data:

```
> long_data <- melt(wide_data, id.vars = c("Country", "A
+                value.name = "Deaths", variable.name = "S
>
> head(long_data, 3)
   Country   Age    Sex Deaths
    <char> <int> <fctr>  <num>
1:  Guinea     0 Female   24.5
2:  Guinea     1 Female   63.8
3:  Guinea     5 Female   44.0
>
> ggplot(long_data, aes(x = Age, y = Deaths, color = Sex
+   facet_wrap(~ Country) +
+   geom_line()
```

# SAVING PLOTS

You can save your plot directly into a pdf or image file.

First store the plot as an R object (rather than just letting it print to RStudio's viewer)

```
> p <- ggplot(data = mmr_data, aes(x = maternal_educatio
+    geom_point()
```

The use "ggsave" and specify an output file type:

```
> ggsave("images/my_plot.pdf", p)
Saving 10 x 7 in image
```

# Additional packages

`ggplot2` has become so popular that other users have started writing add-ons to it:

- **`gridExtra`** - plot tables and arrange multiple plots together
- **`ggrepel`** - label points nicely
- **`RColorBrewer`** - easy-to-use color schemes of various types (colorbrewer2.org)
- `GGally` - various extensions to `ggplot2` like a matrix of graphs
- `cowplot` - combine images with ggplots, highly-flexible multi-figure graphs
- `ggthemes` - more themes, preset colors

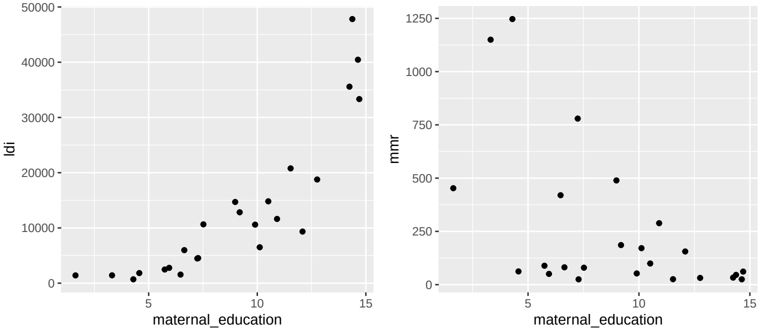
# Additional package: gridExtra

The most important thing the `gridExtra` package can do is more flexibly combine graphs

It's often a useful alternative to `facet_wrap` when you don't want to reshape your data

```
> library(gridExtra)
> p1 <- ggplot(data = mmr_data, aes(x = maternal_ed
+                                   y = ldi)) +
+   geom_point()
>
> p2 <- ggplot(data = mmr_data, aes(x = maternal_ed
+                                   y = mmr)) +
+   geom_point()
>
> grid.arrange(p1, p2, ncol=2)
```

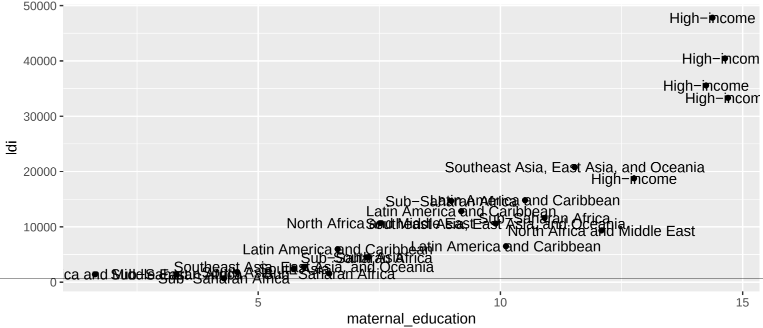
# Additional package: gridExtra



# ADDITIONAL PACKAGE: GGREPEL

ggrepel helps you label points in a cleaner way than geom_text(), by adding geom_text_repel()

```
> ggplot(data = mmr_data, aes(x = maternal_education, y
+       label = super_region_name)) +
+     geom_point() +
+     geom_text()
```

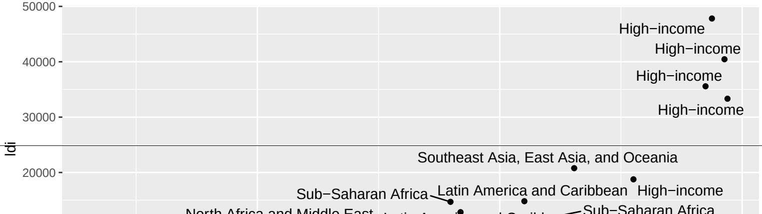


# ADDITIONAL PACKAGE: GGREPEL

ggrepel helps you label points in a cleaner way than geom_text(), by adding geom_text_repel()

```
> library(ggrepel)
> ggplot(data = mmr_data, aes(x = maternal_education, y
+       label = super_region_name)) +
+    geom_point() +
+    geom_text_repel()
```

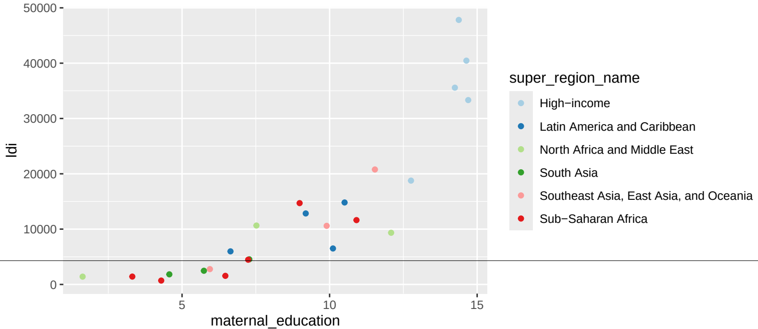
```
Warning: ggrepel: 2 unlabeled data points (too many
overlaps). Consider increasing max.overlaps
```



# ADDITIONAL PACKAGE: RCOLORBREWER

RColorBrewer helps you choose nicer-looking colors

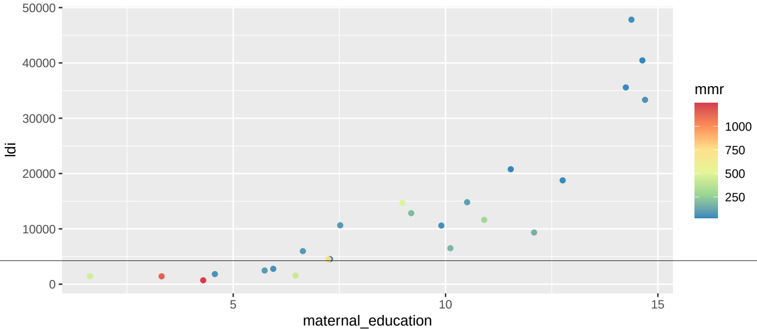
```
> library(RColorBrewer)
> ggplot(data = mmr_data, aes(x = maternal_educatio
+     color = super_region_name)) +
+   geom_point() +
+   scale_color_manual(values = brewer.pal(6, 'Pair
```



# Additional package: RColorBrewer

It comes with sequential, diverging and qualitative color palettes

```
> ggplot(data = mmr_data, aes(x = maternal_educatio
+      color = mmr)) +
+   geom_point() +
+   scale_color_gradientn(colors = rev(brewer.pal(6
```


ldi
maternal_education
mmr