

A BRIEF INTRODUCTION TO R

Gavin L. Simpson
CSEE 2015 · May 20th 2015

0

```
Loading required package: methods
Loading required package: grid
```

1

WHY R?

Why use a complicated, command-line driven stats package like R?

- It's free!
- Widely used by statisticians for new statistical methods
- If something doesn't work the way you like, you can change it
- R is a programming language — you can write your own functions
- R scripts, *Sweave*, & *knitr* for reproducible research
- Works everywhere (Windows, OS X, Linux, ...)
- R was designed for expressive data analysis

2

R BASICS

R BASICS

Start **RStudio** — take a look around

R will be running in either your home directory or where it was installed

Set the working directory to folder containing your analysis

4

GETTING HELP

R comes with a lot of documentation

To get help on functions or concepts within R, use the `"?"` operator

For help on the `getwd()` function use: `?getwd`

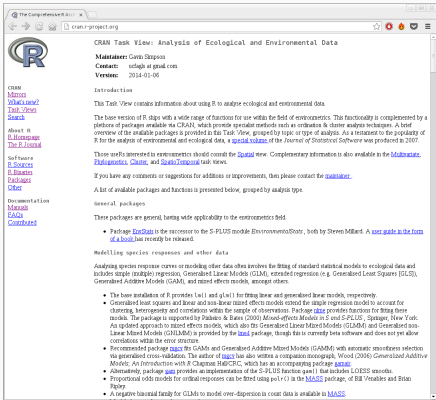
Function `help.search("foo")` will search through all packages installed for help pages with `foo` in them

R-Help mailing list: <http://www.r-project.org/mail.html>

5

GETTING HELP: TASK VIEWS

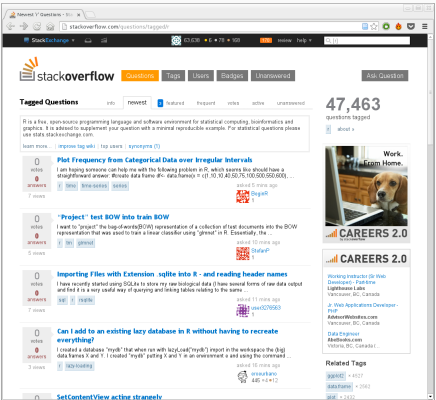
cran.r-project.org/web/views/



6

GETTING HELP: STACK OVERFLOW

stackoverflow.com/questions/tagged/r



7

WORKING WITH R

Type commands at the prompt > — R evaluates them when you hit **Return**

If a line is *not* syntactically complete, the prompt changes to +

Create an object by assigning something to it

```
radius <- 5  
pi * radius^2
```

```
[1] 78.53982
```

If we don't assign, R prints a *representation* of the object

8

PACKAGES

R comes with a basic set of functionality plus some **reccomended** packages

Additional functionality added via **packages** from **CRAN**, **github**, **Bioconductor**, **drat** repos

```
install.packages(c("gapminder", "ggplot2"))  
library("gapminder")  
library("ggplot2")
```

9

READING DATA INTO R

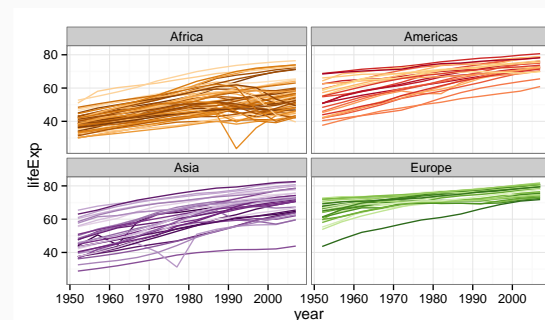
```
gap <- system.file("gapminder.tsv", package = "gapminder")  
gapminder <- read.delim(gap)  
head(gapminder)
```

	country	continent	year	lifeExp	pop	gdpPercap
1	Afghanistan	Asia	1952	28.801	8425333	779.4453
2	Afghanistan	Asia	1957	30.332	9240934	820.8530
3	Afghanistan	Asia	1962	31.997	10267083	853.1007
4	Afghanistan	Asia	1967	34.020	11537966	836.1971
5	Afghanistan	Asia	1972	36.088	13079460	739.9811
6	Afghanistan	Asia	1977	38.438	14880372	786.1134

10

THIS IS WHERE WE ARE HEADING

```
ggplot(subset(gapminder, continent != "Oceania"),  
  aes(x = year, y = lifeExp, group = country, color = country)) +  
  geom_line(show_guide = FALSE) + facet_wrap(~ continent) +  
  scale_color_manual(values = country_colors) +  
  theme_bw()
```



11

R OBJECTS

What kind of object is `gapminder`?

```
str(gapminder)
```

```
'data.frame':  1704 obs. of  6 variables:
 $ country  : Factor w/ 142 levels "Afghanistan",...: 1 1 1 1 1 1 1 1 1 1 ...
 $ continent: Factor w/  5 levels "Africa","Americas",...: 3 3 3 3 3 3 3 3 3 3 ...
 $ year     : int  1952 1957 1962 1967 1972 1977 1982 1987 1992 1997 ...
 $ lifeExp  : num  28.8 30.3 32 34 36.1 ...
 $ pop      : num  8425333 9240934 10267083 11537966 13079460 ...
 $ gdpPercap: num  779 821 853 836 740 ...
```

```
class(gapminder)
```

```
[1] "data.frame"
```

12

DATA FRAMES

A **data frame** is R's version of an Excel spreadsheet

Columns are variables

Rows are observations

Different **types** of data in columns

Each column (**component**) is of the same length

Is a special case of a list

13

SUBSETTING

Access the columns of a data frame using `[`, `[[` or `$`

`$` is simple:

Access just a single variable

```
head(gapminder$country)
```

```
[1] Afghanistan Afghanistan Afghanistan Afghanistan Afghanistan Afghanistan
142 Levels: Afghanistan Albania Algeria Angola Argentina ... Zimbabwe
```

Uses the name of required variable

Partial matching

```
head(gapminder$cou)
```

```
[1] Afghanistan Afghanistan Afghanistan Afghanistan Afghanistan Afghanistan
142 Levels: Afghanistan Albania Algeria Angola Argentina ... Zimbabwe
```

14

SUBSETTING

`[[` is a little more flexible:

Access just a single variable

Use the name of required variable

```
head(gapminder[["continent"]])
```

```
[1] Asia Asia Asia Asia Asia Asia
Levels: Africa Americas Asia Europe Oceania
```

Or select the *n*th component

```
head(gapminder[[2]])
```

```
[1] Asia Asia Asia Asia Asia Asia
Levels: Africa Americas Asia Europe Oceania
```

Partial matching optional — `gapminder[["cont", exact = FALSE]]`

15

SUBSETTING

[is a even more flexible:

Access one or more variables

Use the name(s) of required variable

```
head(gapminder[c("country", "continent")])
```

	country	continent
1	Afghanistan	Asia
2	Afghanistan	Asia
3	Afghanistan	Asia
4	Afghanistan	Asia
5	Afghanistan	Asia
6	Afghanistan	Asia

16

SUBSETTING

Or select the *nth* component(s)

```
head(gapminder[1:2])
```

	country	continent
1	Afghanistan	Asia
2	Afghanistan	Asia
3	Afghanistan	Asia
4	Afghanistan	Asia
5	Afghanistan	Asia
6	Afghanistan	Asia

17

SUBSETTING

Or we can index by rows and columns: [rows, cols, other_args]

```
head(gapminder[1:4, c(1,3)])
```

	country	year
1	Afghanistan	1952
2	Afghanistan	1957
3	Afghanistan	1962
4	Afghanistan	1967

18

SUBSETTING

Leaving the row or column identifier *blank* means “give me all of the rows (columns)”

```
head(gapminder[1:4, ]) # all columns, rows 1--4
```

	country	continent	year	lifeExp	pop	gdpPercap
1	Afghanistan	Asia	1952	28.801	8425333	779.4453
2	Afghanistan	Asia	1957	30.332	9240934	820.8530
3	Afghanistan	Asia	1962	31.997	10267083	853.1007
4	Afghanistan	Asia	1967	34.020	11537966	836.1971

```
head(gapminder[, 1:3], 3) # all rows, columns 1--3
```

	country	continent	year
1	Afghanistan	Asia	1952
2	Afghanistan	Asia	1957
3	Afghanistan	Asia	1962

19

SUBSETTING

Empty dimensions get **dropped** if you select a single column

```
head(gapminder[, 2], 3)           # just column 2
```

```
[1] Asia Asia Asia  
Levels: Africa Americas Asia Europe Oceania
```

Preserve dimensions using `drop = FALSE`

```
head(gapminder[, 2, drop = FALSE], 3) # all rows, columns 1--3
```

```
  continent  
1      Asia  
2      Asia  
3      Asia
```

20

SUBSETTING

Can use a range of **index** types

- Numeric values select the nth elements
- Negative numeric values select all but those elements
- Character values select elements by name (possibly with partial matching)
- Logical values select (TRUE) & deselect (FALSE) elements
- Logical indices are **recycled** to the correct length

```
(1:10)[c(FALSE, TRUE)]
```

```
[1] 2 4 6 8 10
```

```
(1:9)[c(FALSE, TRUE)]           # no warning
```

```
[1] 2 4 6 8
```

21

VECTORS

What are the columns of `gapminder`?

```
str(gapminder)
```

```
'data.frame': 1704 obs. of 6 variables:  
 $ country : Factor w/ 142 levels "Afghanistan",...: 1 1 1 1 1 1 1 1 1 1 ...  
 $ continent: Factor w/ 5 levels "Africa","Americas",...: 3 3 3 3 3 3 3 3 3 3 ...  
 $ year : int 1952 1957 1962 1967 1972 1977 1982 1987 1992 1997 ...  
 $ lifeExp : num 28.8 30.3 32 34 36.1 ...  
 $ pop : num 8425333 9240934 10267083 11537966 13079460 ...  
 $ gdpPercap: num 779 821 853 836 740 ...
```

Each component is a vector, of which there are several types: **numeric**, **character**, **logical**, **factor**, **integer**

22

VECTORS — NUMERIC & INTEGER

R normally stores numeric data as *doubles* (decimal values)

There is an *integer* type too

```
class(gapminder$lifeExp)
```

```
[1] "numeric"
```

```
class(gapminder$year)
```

```
[1] "integer"
```

23

VECTORS — NUMERIC & INTEGER

Create numeric vectors using `c()` or `:`

```
c(1,3,5,7,9)
```

```
[1] 1 3 5 7 9
```

```
1:10
```

```
[1] 1 2 3 4 5 6 7 8 9 10
```

`x:y` is shorthand for `seq(from = x, to = y, by = 1)`

24

VECTORS — CHARACTER

Character vectors contain text (strings)

```
c("foo", "bar")
```

```
[1] "foo" "bar"
```

Quote each string using single or double quotes

25

VECTORS — LOGICAL

Logical vectors are vectors of `TRUE` or `FALSE` values

```
c(TRUE, TRUE, FALSE)
```

```
[1] TRUE TRUE FALSE
```

- `FALSE` is 0
- `TRUE` is anything else, but is coerced to 1

```
as.numeric(c(TRUE, TRUE, FALSE))
```

```
[1] 1 1 0
```

26

VECTORS — FACTORS

Factors are a special kind of vector

- stored internally as a vector of *codes*
- the codes index a set of *levels* or categories, which can be numeric or character

```
f <- factor(c("Male", "Female", "Male"))  
levels(f)
```

```
[1] "Female" "Male"
```

```
f <- factor(c(1,2,5,5,2,1))  
as.numeric(f) # WRONG! Gets internal codes
```

```
[1] 1 2 3 3 2 1
```

```
as.numeric(as.character(f)) # RIGHT! correct coercion
```

```
[1] 1 2 5 5 2 1
```

27

SEQUENCES & PATTERNED VECTORS

Sequences and patterned vectors are very useful in some circumstances

```
seq(from = 1, to = 10, by = 2)
```

```
[1] 1 3 5 7 9
```

```
1:5
```

```
[1] 1 2 3 4 5
```

```
rep(1:3, each = 2)
```

```
[1] 1 1 2 2 3 3
```

```
rep(1:3, times = 3:1)
```

```
[1] 1 1 1 2 2 3
```

28

FUNCTIONS

FUNCTIONS

Pretty much everything in R is either a **function** or the result of a call to one

Called with following format: `fun_name(arg1 = value1, arg2 = value2)`

```
rnorm(10)
```

```
[1] 0.26225849 -0.44882572 0.01023055 -0.52419573 -0.93363066  
[6] -0.63689938 -0.87347879 -0.63486581 -0.79731712 -1.68350867
```

```
args(rnorm)
```

```
function (n, mean = 0, sd = 1)  
NULL
```

```
rnorm(10, mean = 2, sd = 4)
```

```
[1] -2.038808 4.445248 4.609071 6.462982 10.390252 -2.320591 4.360110  
[8] 5.049838 1.887431 -3.856392
```

30

FUNCTIONS

Can use **positional** matching for argument names, but don't except for the first

```
rnorm(10, 2, 4) # What the heck does this do?
```

```
[1] 7.9504510 2.5050757 1.3142966 -2.6856997 -2.0826316 -0.8252653  
[7] 10.5205840 7.8822262 4.4565896 3.4448368
```

If you name arguments can be in any order (can be partial names)

```
rnorm(sd = 4, mean = 2, n = 10)
```

```
[1] -0.1235672 -0.2620203 2.5597837 3.9062834 -3.5814213 -0.1317407  
[7] 1.4856301 7.5251081 0.7260785 3.2503350
```

31

FUNCTIONS

You can write your own functions using the `function()` function

```
foo <- function(x) {           # foo() squares it's input
  x * x                       # last statment determines return value
}
class(foo)
```

```
[1] "function"
```

```
foo(10)
```

```
[1] 100
```

32

SPLIT-APPLY-COMBINE

SPLIT-APPLY-COMBINE

The `split-apply-combine` model is a common type of data analysis

- `split` data into chunks based on one or more factors
- `apply` a function to each chunk
- `combine` the outputs of applying the function to each chunk

Several R packages provide consistent and efficient implementations of the split-apply-combine model

- `plyr`, `dply`, `data.table`

But base R has useful functions too

- `aggregate()`, `split()` + `apply()`-family + `c|rbind()`

34

SPLIT-APPLY-COMBINE

`aggregate` applies `FUN` to a vector, split up by one or more factors:

```
aggregate(pop ~ continent, data = gapminder, FUN = median)
```

	continent	pop
1	Africa	4579311
2	Americas	6227510
3	Asia	14530830
4	Europe	8551125
5	Oceania	6403492

35

SPLIT-APPLY-COMBINE

Can do this by hand too

```
with(gapminder, sapply(split(pop, f = continent), FUN = median))
```

Africa	Americas	Asia	Europe	Oceania
4579311	6227510	14530830	8551125	6403492

36

SPLIT-APPLY-COMBINE — APPLY FAMILY

The **apply** family provides very general approaches to applying function to aspects of data

- **apply** applies a function to the **MARGINS** of a matrix, array, or data frame
- **sapply** applies the function to components of a list or data frame & **simplifies** if possible
- **lapply** applies the function to components of a list or data frame & returns a list
- **tapply** applies the function to chunks of data created by splitting on a factor
- **mapply**, **vapply()**, **rapply()** are specialist alternatives

37

SPLIT-APPLY-COMBINE — APPLY FAMILY

```
apply(gapminder[, 4:5], 2, FUN = median)
```

lifeExp	pop
60.7125	7023595.5000

```
tapply(gapminder$pop, gapminder$continent, FUN = median)
```

Africa	Americas	Asia	Europe	Oceania
4579311	6227510	14530830	8551125	6403492

38

SPLIT-APPLY-COMBINE — APPLY FAMILY

```
with(gapminder, lapply(split(pop, f = continent), FUN = median))
```

```
$Africa  
[1] 4579311
```

```
$Americas  
[1] 6227510
```

```
$Asia  
[1] 14530830
```

```
$Europe  
[1] 8551125
```

```
$Oceania  
[1] 6403492
```

39

MODELLING

PLOTTING

PLOTTING

Your R installation comes with two main plotting toolboxes

- base graphics
- grid graphics

Grid graphics is extremely flexible but that comes at a cost of complexity

Two high-level interfaces to grid provides extensive plotting capabilities

- **lattice**, which comes with R
- **ggplot2**, which needs to be installed from CRAN

BASE GRAPHICS

These are the standard types of plots available in & produced by R & add-on packages

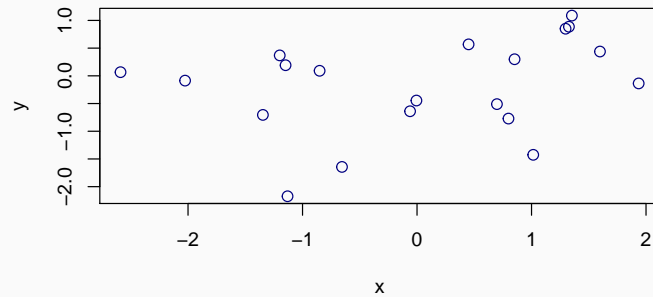
The main function is **plot()**, with **points()**, **lines()**, **text()**, **segments()**, **polygons()**, etc acting as lower-level elements

The look and feel of the plots is essentially controlled via **graphical parameters** — **?par**

Other high-level functions provide to access to the main plot types — **boxplot()**, **hist()**, **stripchart()**, **barchart()**

BASE GRAPHICS

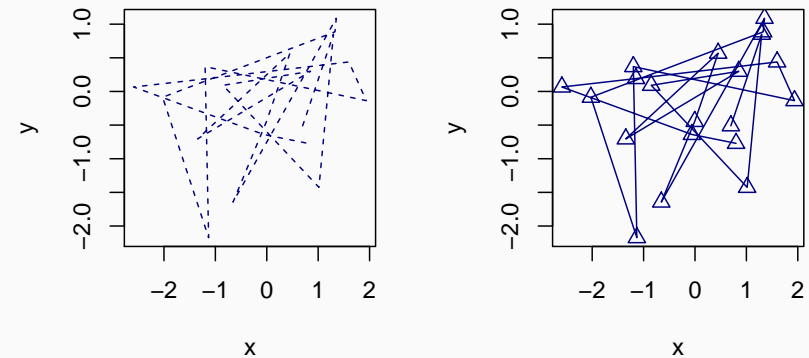
```
x <- rnorm(20)
y <- rnorm(20)
plot(x, y, pch = 1, col = "navyblue", cex = 1.2, type = "p")
```



44

BASE GRAPHICS

```
plot(x, y, lty = "dashed", col = "navyblue", cex = 1.2, type = "l")
plot(x, y, pch = 2, col = "navyblue", cex = 1.2, type = "o")
```



45

BASE GRAPHICS

```
op <- par(mar = c(5,4,5,4) + 0.1) # alter margins
plot(x, y, pch = 1, col = "navyblue", cex = 1.2, type = "p", ann = FALSE, axes = FALSE)
axis(side = 1); axis(side = 2) # add axis, can be customised
axis(side = 3); axis(side = 4)
box() # draw the box round the plot
title(main = "My plot", xlab = "x-axis", ylab = "my y-axis label")
par(op) # reset plotting parameters
```



46

GGPLOT

Base graphics are serviceable but require a lot of craft code to go beyond basic plots — encoding size, colour, etc using data

This is where **lattice** and **ggplot2** graphics come in

These are high-level plotting toolboxes that provide interfaces espousing Trellis Graphics and The Grammar of Graphics ideas, both built on top of **grid**

These are *not* general purpose graphics toolkits — need to follow the ideas & theory behind the respective paradigm

If you want general-purpose, you need to use base graphics or grid

Can't (easily) mix base and grid graphics

47

GGPLOT

`ggplot2` is an implementation of Leland Wilkinson's Grammar of Graphics (hence `gg` in the name)

Three key components of a `ggplot2` plot

- **data** — the data must be in the form of a data frame
- **aesthetics** — how should data be represented on the plot
 - essentially **mappings** from variables to coordinates, size, colour, shape, transparency
- **geometries** — how to physically draw the data & mappings

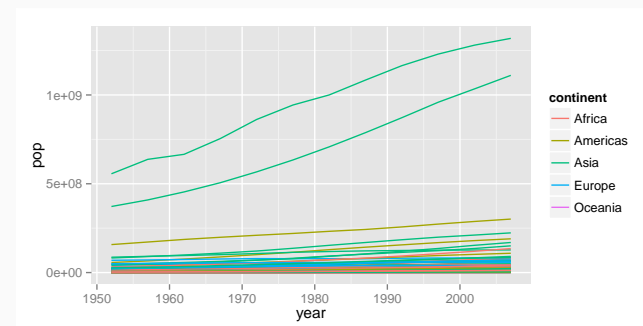
`ggplot` graphics consist of zero or more layers

Additionally, **stats** transform variables, **scales** control axis scaling & legends, **themes** control overall look & feel, **facets** split data into panels

48

GGPLOT — A BASIC PLOT

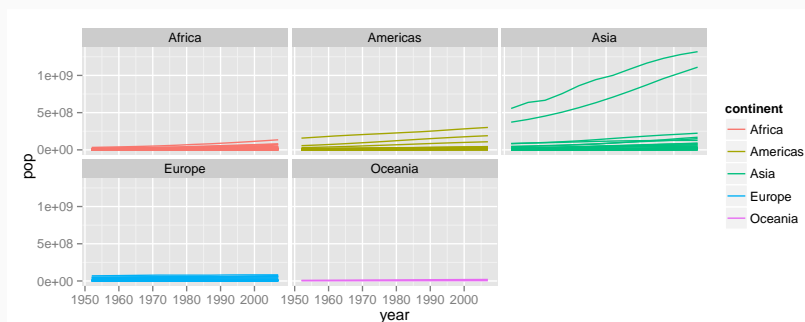
```
library("ggplot2")           # load the package
plt <- ggplot(gapminder, mapping = aes(x = year, y = pop, colour = continent, group = country)) +
  geom_line()                 # add a layer with a line geometry
plt                           # Have to print the object to draw plot
```



49

GGPLOT — FACETING

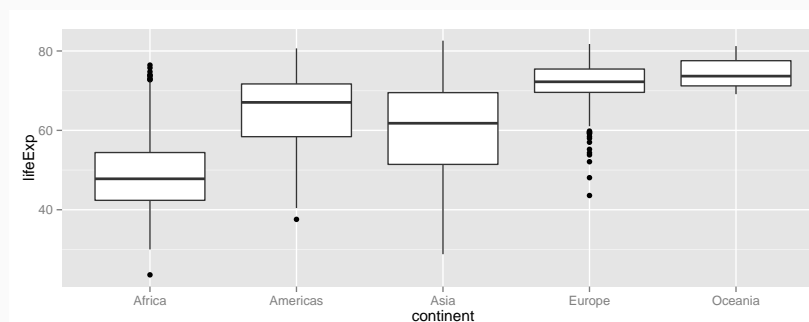
```
plt + facet_wrap(~ continent) # facet by continent
```



50

GGPLOT — STATS

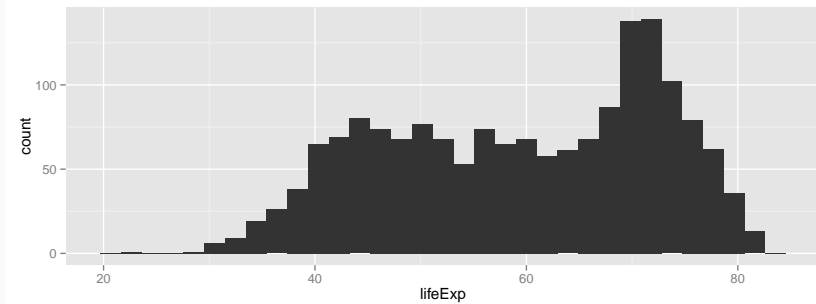
```
ggplot(gapminder, mapping = aes(x = continent, y = lifeExp)) +
  geom_boxplot()              # has a default stat "boxplot"
```



51

GGPLOT — STATS

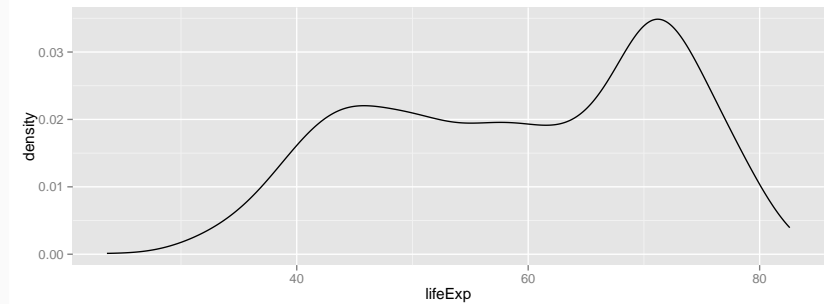
```
ggplot(gapminder, mapping = aes(x = lifeExp)) +  
  geom_histogram()           # has a default stat "bin"
```



52

GGPLOT — STATS

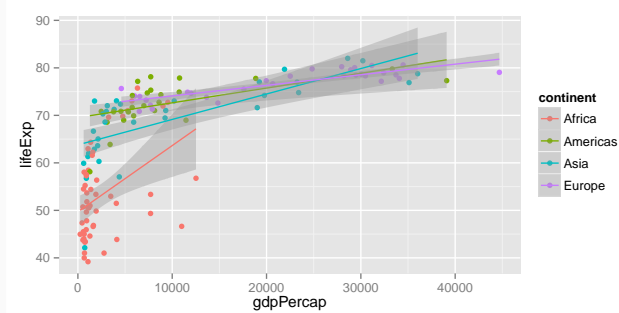
```
ggplot(gapminder, mapping = aes(x = lifeExp)) +  
  geom_line(stat = "density") # change the stat
```



53

GGPLOT — GROUPING & SMOOTHS

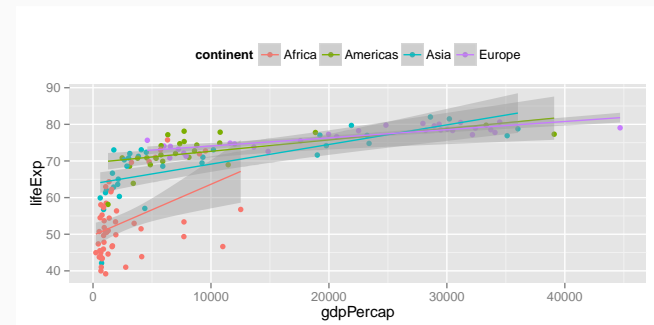
```
(p2 <- ggplot(subset(gapminder, year == 2002 & continent != "Oceania"),  
  aes(x = gdpPerCap, y = lifeExp, colour = continent, group = continent)) +  
  geom_point() + geom_smooth(method = "lm"))
```



54

GGPLOT — THEMES

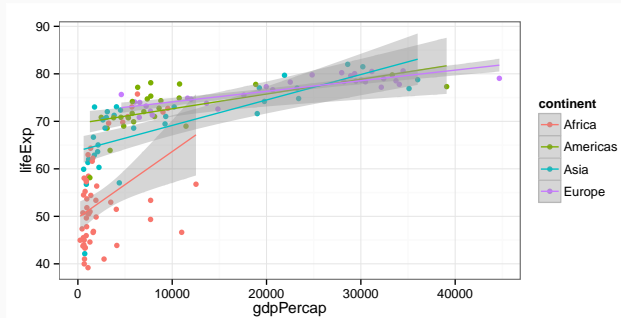
```
p2 + theme(legend.position = "top") # move the legend, see ?theme
```



55

GGPLOT — PRESENT THEMES

```
p2 + theme_bw() # a simple theme
```



56

GGPLOT — EXPORTING PLOTS

ggplot plots can be exported to disk using `ggsave()`

- If the image is on screen (last plot)

```
ggsave(file = "filename.pdf")
```

- If the plot is saved as an object

```
ggsave(p2, file = "filename.pdf")
```

- Specify the size

```
ggsave(file = "filename.pdf", width = 6, height = 4)
```

- Change the file type by modifying the extension

```
ggsave(file = "filename.png")
```

57

RE-USE

Unless indicated otherwise, this slide deck is licensed under a Creative Commons Attribution 4.0 International License.



58