Stat405

Working directories, shortcuts & iteration

Hadley Wickham

Roadmap

- Lectures 1-3: basic graphics
- Lectures 4-6: basic data handling
- Lectures 7-9: basic functions

- The absolutely most essential tools. Rest of course is building your vocab, and learning how to use them all together.

1. Working directory
2. Shortcuts
3. Iteration

Working directory

Why?

All paths in R are relative to the working directory. Life is much easier when you have it correctly set.

Usually want one project per directory. (See also Rstudio's project support)

Makes code easy to move between computers. **Never** use `setwd()` in a script.

How?

Rstudio: Tools | Set working directory | Choose directory ... (^⇧K)

Windows: File | Change dir. For frequent use, make shortcut in that folder.

Mac: ⌘D

Terminal: start R from the desired directory

```
# Find out what directory you’re in
getwd()

# List files in that directory
dir()
```

Your turn

If you haven't already, create a directory for homework 2.

Download the dataset into that directory.

Switch working directories and load the dataset.

```r
# Uses size on screen:
ggsave("my-plot.pdf")
ggsave("my-plot.png")

# Specify size
ggsave("my-plot.pdf", width = 6, height = 6)

# Plots are saved in the working directory
getwd()
```

PDF	PNG
Vector based (can zoom in infinitely)	Raster based (made up of pixels)
Good for most plots	Good for plots with thousands of points

Your turn

Draw a plot of carat vs. price. Save as pdf and png. What are the differences? Where did the files save?

Short cuts

Short cuts

You've been typing `diamonds` many many times. These following shortcuts save typing, but may be a little harder to understand, and will not work in some situations. (Don't forget the basics!)

Four specific to data frames, one more generic.

Function	Package
`subset`	`base`
`summarise`	`plyr`
`mutate`	`plyr`
`arrange`	`plyr`

load plyr with `library(plyr)`.

`base` always automatically loaded

```
# They all have similar syntax.  The first argument
# is a data frame, and all other arguments are
# interpreted in the context of that data frame
# (so you don't need to use data$ all the time)

library(plyr)
subset(df, subset)
mutate(df, var1 = expr1, ...)
summarise(df, var1 = expr1, ...)
arrange(df, var1, ...)

# They all return a modified data frame. You still
# have to save that to a variable if you want to
# keep it
```

color	value
blue	1
black	2
blue	3
blue	4
black	5

color	value
blue	1
blue	3
blue	4

```
subset(df, color == "blue")
```

```
# subset: short cut for subsetting
zero_dim <- diamonds$x == 0 | diamonds$y == 0 |
  diamonds$z == 0
diamonds[zero_dim, ]

subset(diamonds, x == 0 | y == 0 | z == 0)
```

color	value
blue	1
black	2
blue	3
blue	4
black	5

double
2
4
6
8
10

```
summarise(df, double = 2 * value)
```

color	value
blue	1
black	2
blue	3
blue	4
black	5

total
15

```
summarise(df, total = sum(value))
```

```
# summarise/summarize: short cut for creating
# a summary

biggest <- data.frame(
  price.max = max(diamonds$price),
  carat.max = max(diamonds$carat))

biggest <- summarise(diamonds,
  price.max = max(price),
  carat.max = max(carat))
```

color	value
blue	1
black	2
blue	3
blue	4
black	5

color	value	double
blue	1	2
black	2	4
blue	3	6
blue	4	8
black	5	10

```
mutate(df, double = 2 * value)
```

color	value
blue	1
black	2
blue	3
blue	4
black	5

color	value	double	quad
blue	1	2	4
black	2	4	8
blue	3	6	12
blue	4	8	16
black	5	10	20

```
mutate(df, double = 2 * value,
    quad = 2 * double)
```

```r
# mutate: short cut for adding new variables
diamonds$volume <- diamonds$x * diamonds$y * diamonds$z
diamonds$density <- diamonds$volume / diamonds$carat

diamonds <- mutate(diamonds,
  volume = x * y * z,
  density = volume / carat)
```

color	value
4	1
1	2
5	3
3	4
2	5

color	value
1	2
2	5
3	4
4	1
5	3

```
arrange(df, color)
```

color	value
4	1
1	2
5	3
3	4
2	5

color	value
5	3
4	1
3	4
2	5
1	2

```
arrange(df, desc(color))
```

```
# arrange: short cut for reordering
diamonds <- diamonds[order(diamonds$price,
  desc(diamonds$carat)), ]

diamonds <- arrange(diamonds, price, desc(carat))
```

Your turn

Use `summarise`, `mutate`, `subset` and `arrange` to:

Find all diamonds bigger than 3 carats and order from most expensive to cheapest.

Add a new variable that estimates the diameter of the diamond (average of x and y).

Compute depth (z / diameter * 100) yourself. How does it compare to the depth in the data?

```
arrange(subset(diamonds, carat > 3), desc(price))
subset(arrange(diamonds, desc(price)), carat > 3)
biggest <- subsets(diamonds, carat > 3)
arrange(biggest, desc(price)

diamonds <- mutate(diamonds,
  diameter = (x + y) / 2,
  depth2 = z / diameter * 100)

qplot(depth, depth2, data = diamonds)
qplot(depth - depth2, data = diamonds)
```

Aside:
never use attach!

Non-local effects; not symmetric; implicit, not explicit.

Makes it very easy to make mistakes.

Use one of the shortcuts we just discussed or `with()`:

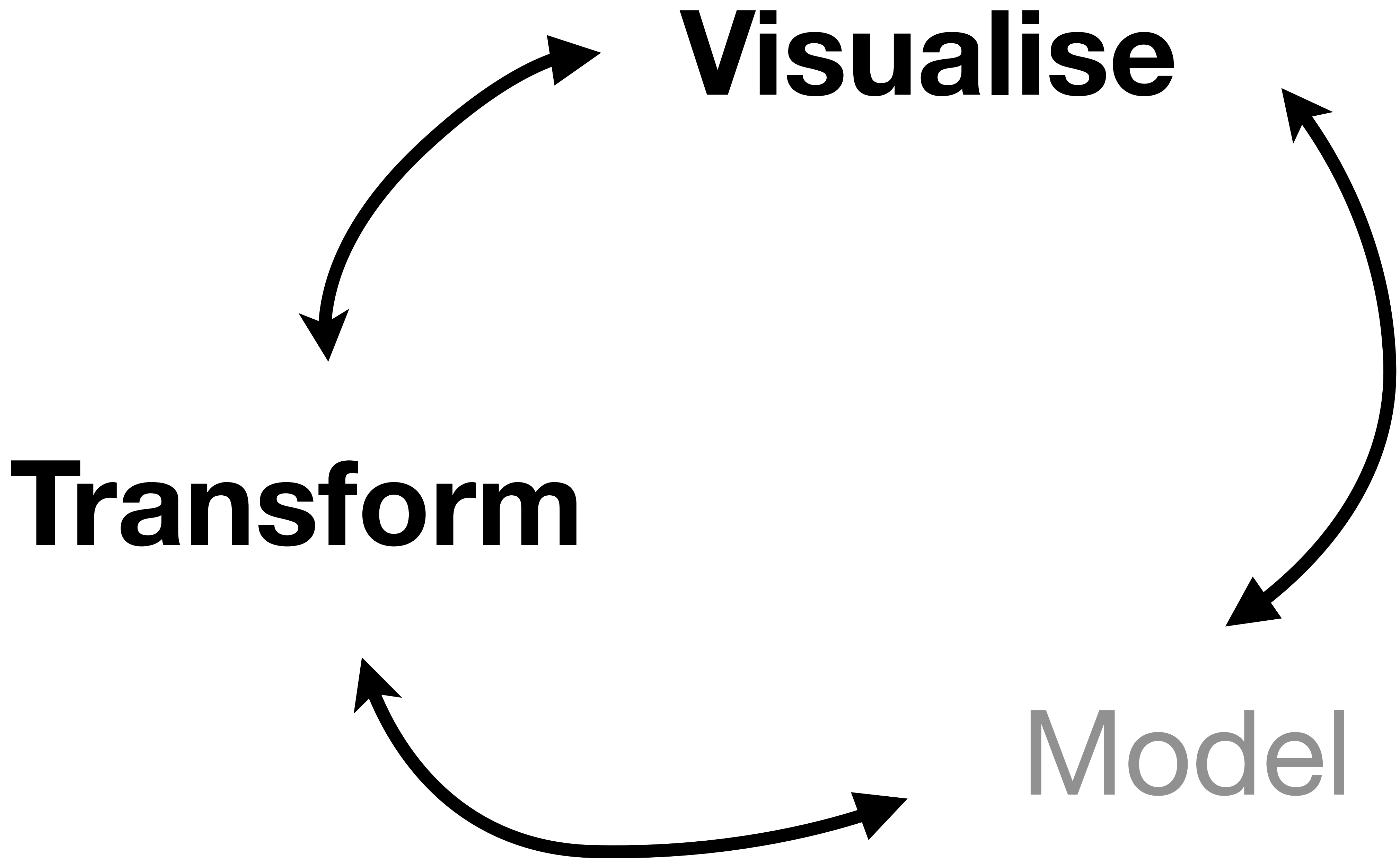
```
with(diamonds, table(color, clarity))
```

```
# with is more general. Use in concert with other
# functions, particularly those that don't have a data
# argument

diamonds$volume <- with(diamonds, x * y * z)

# This won't work:
with(diamonds, volume <- x * y * z)
# with only changes lookup, not assignment
```

Iteration


Visualise
Transform
Model

Stories

Best data analyses tell a story, with a natural flow from beginning to end.

For homeworks, try and come up with a sequence that tell a story. Biggest mistake is stopping too early.

Stories about a small sample of the data can work well.

```
qplot(x, y, data = diamonds)
qplot(x, z, data = diamonds)

# Start by fixing incorrect values
y_big <- diamonds$y > 10
z_big <- diamonds$z > 6

x_zero <- diamonds$x == 0
y_zero <- diamonds$y == 0
z_zero <- diamonds$z == 0

diamonds$x[x_zero] <- NA
diamonds$y[y_zero | y_big] <- NA
diamonds$z[z_zero | z_big] <- NA
```

```
qplot(x, y, data = diamonds)
# How can I get rid of those outliers?

qplot(x, x - y, data = diamonds)
qplot(x - y, data = diamonds)
qplot(x - y, data = diamonds, binwidth = 0.01)
last_plot() + xlim(-0.5, 0.5)
last_plot() + xlim(-0.2, 0.2)

asym <- abs(diamonds$x - diamonds$y) > 0.2
diamonds_sym <- diamonds[!asym, ]

# Did it work?
qplot(x, y, data = diamonds_sym)
qplot(x, x - y, data = diamonds_sym)
# Something interesting is going on there!
qplot(x, x - y, data = diamonds_sym,
  geom = "bin2d", binwidth = c(0.1, 0.01))
```

```
# What about x and z?
qplot(x, z, data = diamonds_sym)
qplot(x, x - z, data = diamonds_sym)

# Subtracting doesn't work - z smaller than x and y
qplot(x, x / z, data = diamonds_sym)

# But better to log transform to make symmetrical
qplot(x, log10(x / z), data = diamonds_sym)

# ...
```

```
# How does symmetry relate to price?
qplot(abs(x - y), price, data = diamonds_sym) +
  geom_smooth()
diamonds_sym <- mutate(diamonds_sym,
  sym = zapsmall(abs(x - y)))

# Are asymmetric diamonds worth more?
qplot(sym, price, data = diamonds_sym) + geom_smooth()

qplot(sym, price, data = diamonds_sym, geom = "boxplot",
  group = sym)
qplot(sym, carat, data = diamonds_sym, geom = "boxplot",
  group = sym)

qplot(carat, price, data = diamonds_sym, colour = sym)
qplot(log10(carat), log10(price), data = diamonds_sym,
  colour = sym, group = sym) + geom_smooth(method = lm, se = F)
```

```r
# Modelling

summary(lm(log10(price) ~ log10(carat) + sym,
  data = diamonds_sym))
# But statistical significance != practical
# significance

sd(diamonds_sym$sym, na.rm = T)
# [1] 0.02368828

# So 1 sd increase in sym, decreases log10(price)
# by -0.01 (= 0.23 * -0.44)
# 10 ^ -0.01 = 0.976
# So 1 sd increase in sym decreases price by ~2%
```