Notes for Chapter 9

Welcome to the `tidyverse`

`tidyverse` is a bundle of many inter-related packages. They share the same philosophy, as well as a common data structure. Learning one of the packages in the `tidyverse` should make it simpler to learn the next one, as they all work well with each other. We are going to explore the manipulation of data sets and their visualization.

To load the **core** packages of `tidyverse` simply load the library:

```
library(tidyverse)
```

For a list of available packages

```
tidyverse_packages()
```

You can update all the packages by calling

```
tidyverse_update()
```

Loading data

There are a number of specialized functions to read character-delimited and fixed-width tables:

```
ants <- read_csv("../data/Mersch2013/behavior.csv")
```

```
## Parsed with column specification:
## cols(
##   colony = col_integer(),
##   tag_id = col_integer(),
##   body_size = col_double(),
##   age = col_integer(),
##   foraging_events = col_integer(),
##   group_period1 = col_character(),
##   group_period2 = col_character(),
##   group_period3 = col_character(),
##   group_period4 = col_character(),
##   visits_brood = col_double(),
##   visits_entrance = col_double(),
##   visits_rubbishpile = col_double()
## )
```

use `read_csv2()` for `;`-delimited, `read_tsv()` for `Tab`-delimited, `read_delim()` for general character-delimited (set `delim =` to set the delimiter), and `read_fwf()` for fixed-width tables.

All these functions: i) try to guess the type of content of each column, and report their choice for you to check; ii) do not convert strings into damn factors; iii) are much faster than the corresponding base-R functions; iv) show you a progress bar for large data sets.

Data is returned as a `tibble`, a modern version of `data.frame`. Several functions can be used to take a look at the data:

```
ants # shows the default visualization of tibbles
head(ants) # first few rows
head(ants, 3)
tail(ants, 3) # last three rows
```

```r
glimpse(ants) # structure of the data
View(ants) # open a spreadheet visualization
```

Subsetting the data

We can subset the rows (observations) or columns (fields). To select rows matching a given condition, use `filter`:

```r
filter(ants, colony == 4)
# multiple filters
filter(ants, colony == 4, age > 30)
filter(ants, colony == 4, age > 100, group_period1 == "F")
# using logical operators
filter(ants, colony == 4, age < 30 | age > 100)
# using %in%
filter(ants, colony == 4, group_period1 %in% c("F", "Q"))
```

Several functions return a subset of the data:

```r
# 1% of the data at random
sample_frac(ants, 0.01)
# 5 random rows
sample_n(ants, 5)
# rows 5 to 10
slice(ants, 5:10)
# top 5 rows once ordered by body_size
top_n(ants, 5, body_size)
# bottom 5 rows once ordered by body_size
top_n(ants, -5, body_size)
```

To choose particular columns, use `select`:

```r
select(ants, colony, age)
# choose all columns whose name contains "group"
select(ants, colony, contains("group"))
# starting with "visit"
select(ants, colony, starts_with("visit")) # also ends_with
# select columns with the same base name followed by a number
select(ants, num_range("group_period", 1:2))
# select columns using regular expressions
select(ants, matches("*.\\d"))
# use - to exclude columns
select(ants, -colony, -age)
```

To take only rows with distinct values, use `distinct`:

```r
distinct(select(ants, colony))
distinct(select(ants, colony, group_period1))
```

Pipes

Writing nested commands detracts from readability, and makes it difficult to change and extend the analysis (e.g., matching parentheses). `tidyverse` is based on **pipes**, like the ones we've seen for UNIX.

```
ants %>% select(colony) %>% distinct()
```

```
## # A tibble: 6 × 1
##   colony
##    <int>
## 1      4
## 2     18
## 3     21
## 4     29
## 5     58
## 6     78
```

to type `%>%` press `Ctrl+Shift+M`.

Ordering the data

You can sort according to column(s) content with `arrange`:

```
ants %>% select(colony) %>% distinct() %>% arrange(colony)
```

```
## # A tibble: 6 × 1
##   colony
##    <int>
## 1      4
## 2     18
## 3     21
## 4     29
## 5     58
## 6     78
```

```
ants %>% select(colony) %>% distinct() %>% arrange(desc(colony))
```

```
## # A tibble: 6 × 1
##   colony
##    <int>
## 1     78
## 2     58
## 3     29
## 4     21
## 5     18
## 6      4
```

Modifying columns

To rename a column, use `rename`

```
ants %>% select(colony, age, body_size) %>% rename(bsize = body_size)
```

```
## # A tibble: 805 × 3
##    colony   age   bsize
##     <int> <int>   <dbl>
## 1       4   330 153.496
## 2       4   330 165.421
## 3       4   246 161.450
## 4       4    57 170.848
```

```
## 5       4   246 151.400
## 6       4    85 143.757
## 7       4   330 140.801
## 8       4   330 150.546
## 9       4    40 173.046
## 10      4    40 153.264
## # ... with 795 more rows
```

To add a new column, whose content is a function of the other columns, use `mutate`

```
ants %>% select(colony, age) %>% mutate(age_yrs = age / 365.25)
```

```
## # A tibble: 805 × 3
##    colony   age   age_yrs
##     <int> <int>     <dbl>
## 1       4   330 0.9034908
## 2       4   330 0.9034908
## 3       4   246 0.6735113
## 4       4    57 0.1560575
## 5       4   246 0.6735113
## 6       4    85 0.2327173
## 7       4   330 0.9034908
## 8       4   330 0.9034908
## 9       4    40 0.1095140
## 10      4    40 0.1095140
## # ... with 795 more rows
```

To drop the columns used for the calculation, use `transmute`

```
ants %>% select(colony, age) %>% transmute(age_yrs = age / 365.25)
```

```
## # A tibble: 805 × 1
##      age_yrs
##        <dbl>
## 1  0.9034908
## 2  0.9034908
## 3  0.6735113
## 4  0.1560575
## 5  0.6735113
## 6  0.2327173
## 7  0.9034908
## 8  0.9034908
## 9  0.1095140
## 10 0.1095140
## # ... with 795 more rows
```

Counting and computing statistics

To count the number of rows, use `tally`

```
ants %>% tally()
```

```
## # A tibble: 1 × 1
##       n
##   <int>
## 1   805
```

To calculate simple statistical summaries, use `summarise`

```
ants %>% summarise(avg_bs = mean(body_size),
                   sd_bs = sd(body_size),
                   cor_age_bs = cor(age, body_size))
```

```
## # A tibble: 1 × 3
##     avg_bs    sd_bs cor_age_bs
##      <dbl>    <dbl>      <dbl>
## 1 163.8883 27.37042 0.05062622
```

Grouping

The real strength of this approach is that you can group the rows according to some criterion, and perform `mutate`, `tally` or `summarise` for each group separately:

```
ants %>% group_by(colony, group_period1) %>% tally()
```

```
## Source: local data frame [24 x 3]
## Groups: colony [?]
##
##    colony group_period1     n
##     <int>         <chr> <int>
## 1       4             C    31
## 2       4             F    53
## 3       4             N    25
## 4       4             Q     1
## 5      18             C    35
## 6      18             F    22
## 7      18             N    70
## 8      18             Q     1
## 9      21             C    43
## 10     21             F    59
## # ... with 14 more rows
```

```
ants %>% group_by(age) %>%
  summarise(avg_bs = mean(body_size))
```

```
## # A tibble: 48 × 2
##      age   avg_bs
##    <int>    <dbl>
## 1      0 144.0770
## 2      5 161.7916
## 3     14 156.0908
## 4     19 142.8045
## 5     40 158.9864
## 6     50 158.1900
## 7     57 159.1266
## 8     64 155.0855
## 9     71 155.7866
## 10    78 162.4277
## # ... with 38 more rows
```

```
ants %>% group_by(colony, group_period1) %>%
  summarise(number = n(), avg_bs = mean(body_size)) %>%
  arrange(colony, avg_bs)
```

```
## Source: local data frame [24 x 4]
## Groups: colony [6]
##
##    colony group_period1 number   avg_bs
##     <int>         <chr>  <int>    <dbl>
## 1       4             N     25 156.8239
## 2       4             C     31 159.3499
## 3       4             F     53 162.6750
## 4       4             Q      1 286.8950
## 5      18             F     22 161.8184
## 6      18             N     70 166.7965
## 7      18             C     35 176.4051
## 8      18             Q      1 308.0060
## 9      21             F     59 150.5762
## 10     21             C     43 163.3813
## # ... with 14 more rows
```

```
ants %>% group_by(group_period1) %>% summarise(max_bs = max(body_size))
```

```
## # A tibble: 4 × 2
##   group_period1  max_bs
##           <chr>   <dbl>
## 1             C 255.633
## 2             F 250.154
## 3             N 262.762
## 4             Q 308.006
```

INTERMEZZO

- calculate average body size per colony
- queens can live up to 26 years: how old were the queens in this experiment?
- how old is the oldest forager? (use `group_period1 == "F"`)
- find the oldest individual per role (`F`, `C`, `N`, `Q`) — there's an error in the data! One ant lived for 107 years...

Plotting

Idea behind `ggplot2`: describe graphs like well-formed sentences (a grammar of graphics).

- Data to plot (the subject) `ggplot(data = ants)`
- Aesthetic mapping: what is what? `ggplot(data = ants) + aes(x = age, y = body_size)`
- Geometry `ggplot(data = ants) + aes(x = age, y = body_size) + geom_point()`

```
ggplot(data = ants %>% filter(age < 2000)) +
  aes(x = age, y = body_size) + geom_point()
# add a smoother
ggplot(data = ants %>% filter(age < 2000)) +
  aes(x = age, y = body_size) + geom_point() +
  geom_smooth()
# exclude queens
ggplot(data = ants %>%
         filter(age < 2000, group_period1 != "Q")) +
  aes(x = age, y = body_size) + geom_point() +
  geom_smooth()
```

```r
# by role
ggplot(data = ants %>%
         filter(age < 2000, group_period1 != "Q")) +
  aes(x = age, y = body_size, colour = group_period1) +
  geom_point() + geom_smooth()
```

One variable

```r
pl <- ggplot(data = ants %>% filter(age < 2000, group_period1 != "Q"))
pl + geom_histogram(aes(body_size), bins = 40) # histogram
pl + geom_density(aes(body_size)) # density plot
pl + geom_bar(aes(group_period1)) # barplot (discrete)
```

One variable, discrete x axis

```r
pl + geom_boxplot(aes(x = group_period1, y = age)) # boxplot
pl + geom_violin(aes(x = group_period1, y = age)) # violin plot
```

Two continuous variables

```r
pl <- ggplot(data = ants %>% filter(group_period1 == "F"),
       aes(x = body_size, y = foraging_events))
pl + geom_point()
pl + geom_point() + geom_smooth()
pl + geom_point() + geom_smooth(method = "lm") # linear model
pl + geom_point() + geom_smooth(se = FALSE) # no error ribbon
pl + geom_point() + geom_line()
```

Scales

```r
pl <- ggplot(data = ants %>% filter(age < 2000),
             aes(x = age, y = body_size,
                 colour = group_period1)) +
  geom_point()
pl + scale_colour_brewer(palette = "Set1")
pl + scale_colour_manual(values = c("yellow",
                                    "red",
                                    "green",
                                    "blue"))
pl + scale_x_log10("age in days") +
  scale_y_sqrt("body size (in pixels, 0.06 mm)")
pl <- ggplot(data = ants %>%
               filter(age < 2000),
             aes(x = age, y = body_size,
                 colour = log(age))) +
  geom_point() +
  scale_x_log10()
```

```
pl + scale_color_gradient2(low = "black",
                           mid = "green", high = "red")
pl + aes(shape = group_period1,
         size = body_size)
```

Facets

```
pl <- ggplot(data = ants,
             aes(x = body_size, colour = group_period1, fill = group_period1)) +
  geom_histogram(alpha = 0.1, position = position_dodge())
pl + facet_grid(~colony)
pl + facet_grid(colony~.)
pl + facet_grid(colony~group_period1)
```

Themes

```
pl <- ggplot(data = ants %>% filter(age < 2000),
             aes(x = age, y = body_size, colour = group_period1)) +
  geom_point()
pl + theme_bw()
pl + theme_dark()
pl + theme_minimal()
pl + theme_bw() + theme(legend.position = "bottom")
pl + theme_bw() + theme(legend.position = "none") +
  xlab("description x axis") + ggtitle("my main title")
```

Legends

```
pl <- ggplot(data = ants %>% filter(age < 2000),
             aes(x = age, y = body_size,
                 colour = group_period1,
                 shape = group_period1,
                 size = body_size)) +
  geom_point() +
  facet_wrap(~group_period1, scales = "free")
show(pl)
pl + guides(colour = "none")
pl + scale_size_continuous("my legend",
                           breaks = c(100, 200, 300))
pl + scale_shape_discrete("my shapes", solid = FALSE)
```

Setting a value

```
ggplot(data = ants, aes(x = factor(colony))) +
  geom_bar(colour = "red", fill = "darkblue")
ggplot(data = ants, aes(x = group_period1,
                        y = body_size,
```

```
                        colour = group_period1)) +
  geom_jitter(alpha = 0.5)
```

Stats

```
ggplot(data = ants %>% filter(age < 2000),
       aes(x = age, y = body_size)) + stat_bin2d()
ggplot(data = ants %>% filter(age < 2000),
       aes(x = age, y = body_size)) + stat_density2d()
ggplot(data = ants %>% filter(age < 2000),
       aes(x = age, y = body_size)) +
  geom_point() + stat_ellipse(level = 0.75, colour = "red")
ggplot(data = ants %>% filter(age < 2000),
       aes(x = age)) + stat_ecdf()
pl <- ggplot(data = ants %>% filter(age < 2000),
             aes(x = age, y = body_size)) +
  geom_point()
pl + stat_summary(fun.y = "median",
                  geom = "point",
                  colour = "red", shape = 3)
ggplot(data = ants,
       aes(x = group_period1,
           y = body_size,
           colour = group_period1)) +
  geom_jitter(alpha = 0.5) +
  stat_summary(fun.data = "mean_cl_normal", colour = "black")
```

Adding multiple geometries

```
pl <- ggplot(data = ants %>% filter(age < 2000),
             aes(x = age, y = body_size)) + geom_point()
pl + geom_abline(intercept = 0,
                 slope = 2, linetype = 2,
                 colour = "red") +
  geom_vline(xintercept = 500, linetype = 3)
```

Saving

```
ggsave(pl, file = "test.png", width = 5, height = 7)
```

INTERMEZZO

- produce a boxplot showing the distribution of foraging events by `group_period1`. Set the scale for the `y` axis in square root.
- plot an histogram of foraging events by colony and `group_period1`.

Tidy data

The concept of tidy data is simple: each variable has its own column; each observation has its own row. As the name implies `tidyverse` is meant to manipulate tidy data. There are many functions to help you wrestle with your data to make it into tidy form.

Gathering

```
visits <- ants %>% group_by(group_period1, colony) %>%
  summarise(rubbish = mean(visits_rubbishpile),
            brood = mean(visits_brood),
            entrance = mean(visits_entrance))
visits <- visits %>% gather(location, average_visits, 3:5)
ggplot(data = visits, aes(x = group_period1,
                          y = location,
                          fill = group_period1,
                          alpha = sqrt(average_visits))) +
  geom_tile() + facet_wrap(~colony)
```

Spreading

```
foraging_by_colony <- ants %>%
  group_by(colony, group_period1) %>%
  summarise(avg_foraging = mean(foraging_events))
foraging_by_colony <- foraging_by_colony %>%
  spread(group_period1, avg_foraging)
```

Example: moving between groups

Some ants change of status between `group_period1` and `group_period2`. We are going to investigate how frequent this is.

```
roles <- ants %>% select(group_period1, group_period2) %>%
  rename(p1 = group_period1, p2 = group_period2)
# add a counter
roles <- roles %>%
  mutate(count = 1) %>%
  group_by(p1, p2) %>%
  summarise(transitions = sum(count))
# remove nas
roles <- roles %>% filter(!is.na(p2))
# we want to have probabilities: sum by p1
roles <- roles %>% group_by(p1) %>%
  mutate(tottrans = sum(transitions)) %>%
  mutate(prob = transitions / tottrans)
ggplot(data = roles, aes(x = p1, y = p2, fill = prob)) +
  geom_tile()
# table for paper
roles %>% select(p1, p2, prob) %>% spread(p2, prob, fill = 0)
```

Joining data