Common Applications

Functions

Writing your own functions

- When to write your own function?

Writing your own functions

- when you are writing the same code over and over again
- when you are running the same code over and over again
- when you want to make your code available to others

Writing your own functions

Nice column names!

```
nice_name <- function(x){gsub(" ", "_", tolower(x))}
```

Writing your own functions

Make a nice(er) column name

```
evil_name <- "StuPid name"

nice_name(evil_name)
```

```
## [1] "stupid_name"
```

Writing your own functions

We can use the function in to in a tidy pipeline, just like any of the built-in functions

```
## # A tibble: 5 x 2
##   `A B C`  NORM
##   <chr>   <dbl>
## 1 A        5.55
## 2 B        4.72
## 3 C        6.78
## 4 D        5.19
## 5 E        6.14
```

Writing your own functions

We can use the function in to in a tidy pipeline, just like any of the built-in functions

```
df %>% rename_all(nice_name) %>% slice(1:5)
```

```
## # A tibble: 5 x 2
##   a_b_c  norm
##   <chr> <dbl>
## 1 A      5.55
## 2 B      4.72
## 3 C      6.78
## 4 D      5.19
## 5 E      6.14
```

Mutate and factors

if_else

The `if_else()` function returns one of two options based on the result of a logical test:

```
if_else(condition [test with yes/no answer],
        true [do this],
        false [do this], ...)
```

if_else

For example:

```
## # A tibble: 7 x 1
##   var
##   <chr>
## 1 teaspoon
## 2 Mr. Nick
## 3 silverspoon
## 4 just a spoon ok!
## 5 servingspoon
## 6 Mr. Spoon
## 7 tablespoon
```

if_else

Spoon or person?

```
df_type <-
df %>%
mutate(type = if_else(condition = grepl(".*spoon", var),
                      true = "spoon",
                      false = "person"))
```

combining factors

We can make factors with the `factor()` function.

```
df_factor <- df_type %>%
mutate(type_fct = factor(type))
df_factor
```

```
## # A tibble: 7 x 3
##   var              type   type_fct
##   <chr>            <chr>  <fct>
## 1 teaspoon         spoon  spoon
## 2 Mr. Nick         person person
## 3 silverspoon      spoon  spoon
## 4 just a spoon ok! spoon  spoon
## 5 servingspoon     spoon  spoon
## 6 Mr. Spoon        person person
## 7 tablespoon       spoon  spoon
```

combining factors

A factor with 6 levels

```
x <- factor(c("control", "lpg", "gasifier",
              "fan_rocket", "rocket_elbow", "three_stone"))
```

combining factors

Combine 4 of the levels

```
library(forcats)

fct_collapse(x, wood = c("gasifier", "fan_rocket",
                         "rocket_elbow", "three_stone"))
```

```
## [1] control lpg     wood    wood    wood    wood
## Levels: control wood lpg
```

Loading multiple files

Loading multiple files

Common task you might be faced with is loading multiple files in the same format

- e.g. an instrument that creates a file every day
- how might we go about this?

Loading multiple files

First we need to know the files we want to load

- The `list.files()` function can help us here:

```
list.files(
  path = ["path to directory you wish to load"],
  pattern = ["regular expression?"],
  full.names = ["full path to each file?"],
  include.dirs = ["include files in sub-folders?"]
  ...)
```

Loading multiple files

Let's list our files

```
files <- list.files(path = ".")
files[1:5]
```

```
## [1] "cb_2015_08_tract_500k.zip" "common_applications.log"
## [3] "common_applications.pdf"   "common_applications.Rmd"
## [5] "continuing_with_r.pdf"
```

Loading multiple files

Let's try again!

```
files <- list.files(path = "../data/pax_data")
files[1:5]
```

```
## [1] "PAX_04.txt" "PAX_05.txt" "PAX_06.txt" "PAX_07.txt" "PAX_
```

Loading multiple files

And again!

```
files <- list.files(path = "../data/pax_data",
           pattern = "^PAX.*.txt$",
           full.names = TRUE)
files[1:5]
```

```
## [1] "../data/pax_data/PAX_04.txt" "../data/pax_data/PAX_05.tx
## [3] "../data/pax_data/PAX_06.txt" "../data/pax_data/PAX_07.tx
## [5] "../data/pax_data/PAX_08.txt"
```

Loading multiple files

Now that we know the file names, we need a function to load the data

```
library(readr)
data_file <- read_delim(files[1],
                        delim = "\t",
                        col_type = cols())
data_file
```

```
## # A tibble: 9,411 x 3
##    current_fire_datetime current_fire_Babs_bkg_f~ current_fir
##    <chr>                                   <dbl>
##  1 2016/10/6 10:31:30                      -1.54
##  2 2016/10/6 10:31:31                       7.05
##  3 2016/10/6 10:31:32                       3.22
##  4 2016/10/6 10:31:33                      -2.04
##  5 2016/10/6 10:31:34                      10.0
##  6 2016/10/6 10:31:35                       8.22
##  7 2016/10/6 10:31:36                      12.9
```

Loading multiple files

Can up `map()` over vector of file names to read in all the data.

```
library(purrr)
library(dplyr)
data_pax <- map(files, read_delim,
                delim = "\t", col_type = cols()) %>%
bind_rows()
```

Loading multiple files

Looking at what we loaded:

```
head(data_pax, 3)
```

```
## # A tibble: 3 x 3
##   current_fire_datetime current_fire_Babs_bkg_fit current_fir
##   <chr>                                     <dbl>
## 1 2016/10/6 10:31:30                        -1.54
## 2 2016/10/6 10:31:31                         7.05
## 3 2016/10/6 10:31:32                         3.22
```

```
nrow(data_pax)
```

```
## [1] 721130
```

Loading multiple files

We don't have to use an existing function:

```
read_pax <- function(file){
read_delim(file,
           delim = "\t",
           col_type = cols()) %>%
mutate(datetime =
  as.POSIXct(strptime(x = current_fire_datetime,
                      format = "%Y/%m/%d %T"))) %>%
rename(babs = "current_fire_Babs_bkg_fit",
       bscat = "current_fire_Bscat_bkg_fit") %>%
select(datetime, babs, bscat)
}
```

Loading multiple files

We can use our own function the same way we use an existing function

```
data_pax <- map(files, read_pax) %>%
  bind_rows()
head(data_pax, 3)
```

```
## # A tibble: 3 x 3
##   datetime             babs bscat
##   <dttm>              <dbl> <dbl>
## 1 2016-10-06 10:31:30 -1.54  54.3
## 2 2016-10-06 10:31:31  7.05 123.
## 3 2016-10-06 10:31:32  3.22 117.
```

Exploring parameter space

Exploring parameter space

The `cross_*` family of functions are useful for this

```
library(purrr)
cross_df(.l, ...)
```

Exploring parameter space

Start with a list of the the parameters

```
var_list <- list (distance = seq(1, 5, 1),
                  time = seq(10, 1000, 5))
```

Exploring parameter space

Use the `cross_df()` function to create a dataframe of all the combinations

```
cross_df(var_list)
```

```
## # A tibble: 995 x 2
##    distance  time
##       <dbl> <dbl>
##  1        1    10
##  2        2    10
##  3        3    10
##  4        4    10
##  5        5    10
##  6        1    15
##  7        2    15
##  8        3    15
##  9        4    15
## 10        5    15
```

Exploring parameter space

Create a function to run our analysis

```
library(tidyr)

speed_f <- function(distance, time){distance * time}
```

Exploring parameter space

Use `cross_df()` and `map2()` to run our function on every combination of variables

```
df <- cross_df(var_list) %>%
mutate(speed = map2(.x = distance,
                    .y = time,
                    .f = speed_f)) %>%
unnest(speed)

head(df, 3)
```

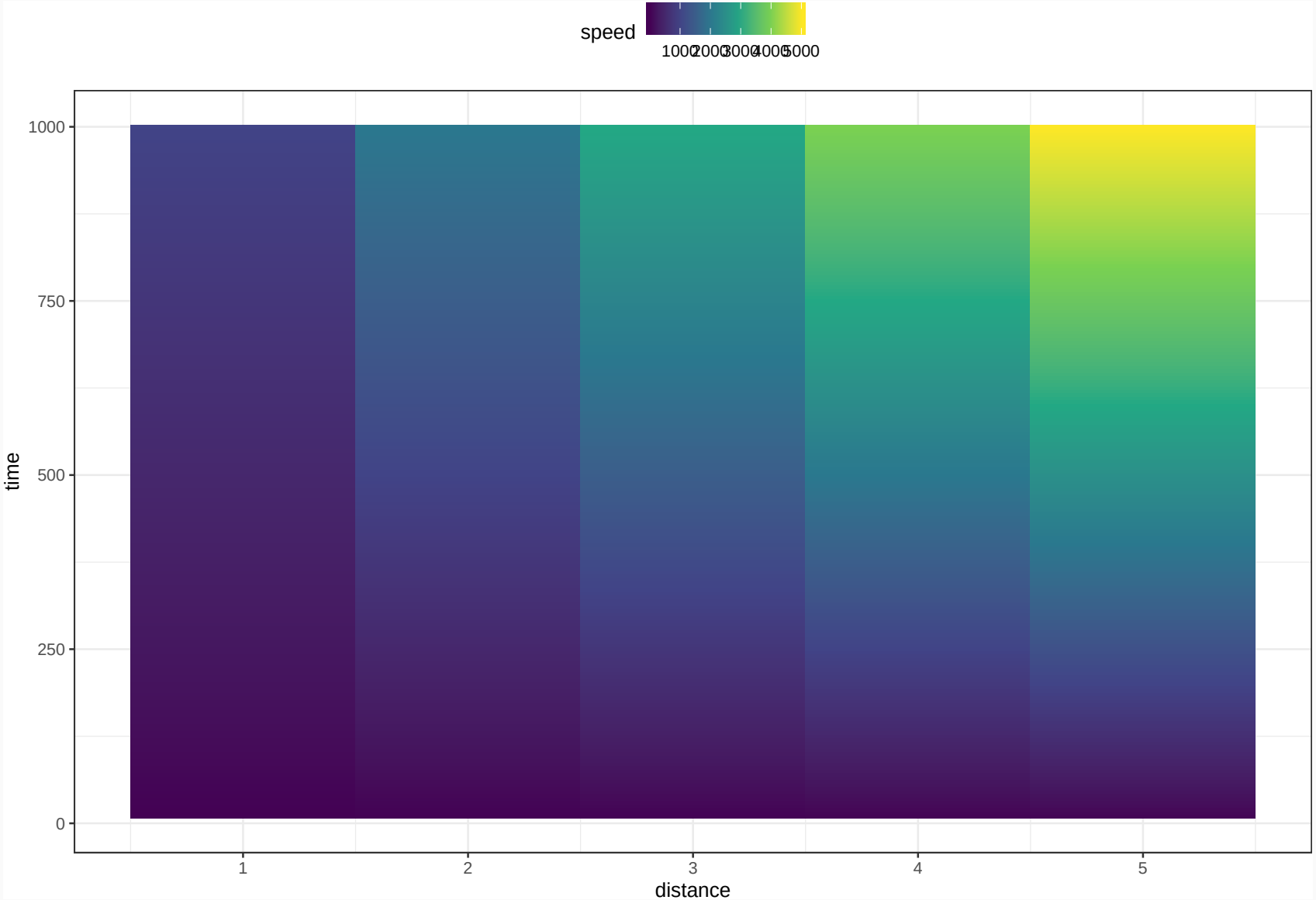
```
## # A tibble: 3 x 3
##   distance  time speed
##      <dbl> <dbl> <dbl>
## 1        1    10    10
## 2        2    10    20
## 3        3    10    30
```

Exploring parameter space

```
library(ggplot2)

p <- ggplot(df, aes(x = distance,
                    y = time,
                    fill = speed)) +
  geom_tile() +
  theme_bw() +
  scale_fill_continuous(type = "viridis") +
  theme(legend.position = "top")
```

Exploring parameter space



Deeply nested lists

Deeply nested lists

- download the .rds file
- add it to your working directory

https://tinyurl.com/deeplynested-list

Deeply nested lists

Read in data

```
library(readr)
result <- read_rds("../data/result.rds")
```

Deeply nested lists

Work out what the object is

```
class(result)
```

```
## [1] "response"
```

Deeply nested lists

Use the `httr` library to extract the `content of the object`

```
library(httr)
content <- content(result)
class(content)
```

```
## [1] "list"
```

Deeply nested lists

Start making it look more like a dataframe

```
library(tibble)
tbl <- as_tibble(content)
```

Deeply nested lists

```
## # A tibble: 288 x 1
##    data
##    <list>
##  1 <named list [4]>
##  2 <named list [4]>
##  3 <named list [4]>
##  4 <named list [4]>
##  5 <named list [4]>
##  6 <named list [4]>
##  7 <named list [4]>
##  8 <named list [4]>
##  9 <named list [4]>
## 10 <named list [4]>
## # ... with 278 more rows
```

Deeply nested lists

When you see lists in dataframes, think `nesting`

```
library(tidyr)
unnested <- unnest_wider(tbl, data)
```

Deeply nested lists

```
## # A tibble: 288 x 4
##    timestamp                score sensors    indices
##    <chr>                    <dbl> <list>     <list>
##  1 2019-11-11T23:55:00.000Z    83 <list [7]> <list [5]>
##  2 2019-11-11T23:50:00.000Z    83 <list [7]> <list [5]>
##  3 2019-11-11T23:45:00.000Z    83 <list [7]> <list [5]>
##  4 2019-11-11T23:40:00.000Z    83 <list [7]> <list [5]>
##  5 2019-11-11T23:35:00.000Z    83 <list [7]> <list [5]>
##  6 2019-11-11T23:30:00.000Z    83 <list [7]> <list [5]>
##  7 2019-11-11T23:25:00.000Z    83 <list [7]> <list [5]>
##  8 2019-11-11T23:20:00.000Z    83 <list [7]> <list [5]>
##  9 2019-11-11T23:15:00.000Z    83 <list [7]> <list [5]>
## 10 2019-11-11T23:10:00.000Z    83 <list [7]> <list [5]>
## # ... with 278 more rows
```

Deeply nested lists

Let's take a look at one of the `sensor` cells:

```
sensors_1 <- unnested$sensors[[1]]
sensors_1
```

```
## [[1]]
## [[1]]$comp
## [1] "temp"
##
## [[1]]$value
## [1] 19.51567
##
##
## [[2]]
## [[2]]$comp
## [1] "humid"
##
## [[2]]$value
```

Deeply nested lists

When we see lists of lists or a list column think `map`

Use `map_df()` to extract the valves to a dataframe

```
library(purrr)
map_df(sensors_1, magrittr::extract, c("comp", "value"))
```

```
## # A tibble: 7 x 2
##   comp  value
##   <chr> <dbl>
## 1 temp   19.5
## 2 humid  16.0
## 3 co2   456.
## 4 voc   162.
## 5 pm25    1
## 6 lux    61.6
## 7 spl_a  51.8
```

Deeply nested lists

But!!! That's just one cell of a list column

```
list_of_tbl <- unnested %>%
mutate(df = map(sensors,
                ~ .x %>% map_df(magrittr::extract,
                                c("comp", "value"))))
```

Deeply nested lists

Which gives us:

```
## # A tibble: 288 x 3
##    timestamp                score df
##    <chr>                    <dbl> <list>
##  1 2019-11-11T23:55:00.000Z    83 <tibble [7 x 2]>
##  2 2019-11-11T23:50:00.000Z    83 <tibble [7 x 2]>
##  3 2019-11-11T23:45:00.000Z    83 <tibble [7 x 2]>
##  4 2019-11-11T23:40:00.000Z    83 <tibble [7 x 2]>
##  5 2019-11-11T23:35:00.000Z    83 <tibble [7 x 2]>
##  6 2019-11-11T23:30:00.000Z    83 <tibble [7 x 2]>
##  7 2019-11-11T23:25:00.000Z    83 <tibble [7 x 2]>
##  8 2019-11-11T23:20:00.000Z    83 <tibble [7 x 2]>
##  9 2019-11-11T23:15:00.000Z    83 <tibble [7 x 2]>
## 10 2019-11-11T23:10:00.000Z    83 <tibble [7 x 2]>
## # ... with 278 more rows
```

Deeply nested lists

Use `unnest()` to flatten the list-column back to a regular column

```
long_df <- list_of_tbl %>%
select(-sensors, -indices) %>%
unnest(df)
```

Deeply nested lists

```
## # A tibble: 2,016 x 4
##    timestamp                score comp  value
##    <chr>                    <dbl> <chr> <dbl>
##  1 2019-11-11T23:55:00.000Z    83 temp   19.5
##  2 2019-11-11T23:55:00.000Z    83 humid  16.0
##  3 2019-11-11T23:55:00.000Z    83 co2   456.
##  4 2019-11-11T23:55:00.000Z    83 voc   162.
##  5 2019-11-11T23:55:00.000Z    83 pm25    1
##  6 2019-11-11T23:55:00.000Z    83 lux    61.6
##  7 2019-11-11T23:55:00.000Z    83 spl_a  51.8
##  8 2019-11-11T23:50:00.000Z    83 temp   19.6
##  9 2019-11-11T23:50:00.000Z    83 humid  16.0
## 10 2019-11-11T23:50:00.000Z    83 co2   458.
## # ... with 2,006 more rows
```

Deeply nested lists

Finally we can `pivot` the long object back to a wider format:

```
wider_df <- long_df %>%
pivot_wider(names_from = "comp", values_from = "value")
```

```
## # A tibble: 288 x 9
##    timestamp                score  temp humid   co2   voc  pm
##    <chr>                    <dbl> <dbl> <dbl> <dbl> <dbl> <db
##  1 2019-11-11T23:55:00.000Z    83  19.5  16.0  456.  162. 1
##  2 2019-11-11T23:50:00.000Z    83  19.6  16.0  458.  164. 0.8
##  3 2019-11-11T23:45:00.000Z    83  19.6  16.0  460.  165. 1.0
##  4 2019-11-11T23:40:00.000Z    83  19.6  16.0  461.  165. 0.8
##  5 2019-11-11T23:35:00.000Z    83  19.6  16.0  461.  167. 0.7
##  6 2019-11-11T23:30:00.000Z    83  19.6  16.0  458.  170. 0.7
##  7 2019-11-11T23:25:00.000Z    83  19.7  16.0  461.  170. 0.9
##  8 2019-11-11T23:20:00.000Z    83  19.7  15.9  460.  170. 0.8
##  9 2019-11-11T23:15:00.000Z    83  19.7  15.9  463.  171. 0.8
## 10 2019-11-11T23:10:00.000Z    83  19.7  15.9  465.  172. 0.8
```

Deeply nested lists

Fix the timestamp!

```
p_data <- wider_df %>%
  mutate(timestamp = as.POSIXct(strptime(timestamp,
                                format = "%FT%H:%M:%S",
                                tz = "GMT")),
         timestamp = format(timestamp,
                            tz = "US/Mountain",
                            usetz = TRUE),
         timestamp = as.POSIXct(timestamp))
```

Deeply nested lists

```
## # A tibble: 288 x 9
##    timestamp           score  temp humid   co2   voc  pm25    
##    <dttm>              <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <d
##  1 2019-11-11 16:55:00    83  19.5  16.0  456.  162. 1       6
##  2 2019-11-11 16:50:00    83  19.6  16.0  458.  164. 0.833   8
##  3 2019-11-11 16:45:00    83  19.6  16.0  460.  165. 1.07    8
##  4 2019-11-11 16:40:00    83  19.6  16.0  461.  165. 0.897   8
##  5 2019-11-11 16:35:00    83  19.6  16.0  461.  167. 0.767   8
##  6 2019-11-11 16:30:00    83  19.6  16.0  458.  170. 0.767   8
##  7 2019-11-11 16:25:00    83  19.7  16.0  461.  170. 0.900   8
##  8 2019-11-11 16:20:00    83  19.7  15.9  460.  170. 0.833   9
##  9 2019-11-11 16:15:00    83  19.7  15.9  463.  171. 0.867   9
## 10 2019-11-11 16:10:00    83  19.7  15.9  465.  172. 0.800   9
## # ... with 278 more rows
```

Deeply nested lists

Finally, we can plot the data:

```
p_data <- p_data %>%
 pivot_longer(-timestamp, names_to = "var", values_to = "val")

p <- ggplot(data = p_data,
            mapping = aes(x = timestamp, y = val, color = var)) +
  geom_point() +
  geom_line() +
  theme_bw() +
  facet_wrap(~var, ncol = 1, scales = "free_y") +
  theme(legend.position = "none") +
  xlab("") + ylab("")
```

Deeply nested lists