Data Wrangling in R

QuantArch Week 2 | 14-02-2022

Universiteit Leiden

Summary

Data types in R:

logical: `TRUE`, `FALSE`, `NA`
integer: `1`, `2`, ..., infinity ($\aleph_0$)
double: `NaN`, `1.0`, `1.1`, `1.2`, ..., more infinity ($\aleph_1$)
NULL: `NULL`

Data structures in R

Vector: `c("7","ate", "9")`
Matrix: Table with one data type
Data frame: Table containing different data types
List: Ordered collection of any R objects

Function: `print()`, `mean()`, `sum()`, `c()`

Summary

Subsetting

Position:

`vector[3]` show third element
`vector[-3]` show all EXCEPT third element
`vector[c(1,2,3)]` show first three elements

Condition:

`vector[vector == "condition"]` show elements matching condition
`vector[vector < 5]` show elements less than 5

Uploading Data

Dataset

House pits at the Mississippian Snodgrass site in Butler County, Missouri, U.S.A.

Download *House Pits Missouri.xlsx* from Brightspace and store the file in a folder `data/raw_data`.

☠️ **DO NOT TOUCH THIS FILE** ☠️
Make a copy of the file and call it `house-pits_cleaned.xlsx`

```
project
|   README.md
|
└──data
|   |
|   └──raw_data
|       |   House Pits Missouri.xlsx
|       |   house-pits_cleaned.csv
|
└──scripts
|   |
|   |
|   └──lab1.R
```

Exercise

Fix problems with the data

What issues might you encounter with the data as is?

Is it machine-readable?

Is it human-readable?

Save data

Save the cleaned data as `house-pits_cleaned.csv`

Why `*.csv`? (comma separated values)

FA-**I**-R data

- **Interoperability:** non-propritary data format

Uploading data to R

There are multiple ways to upload data to R.

Base R

```
read.table  # for *.txt files
read.csv    # for *.csv files
read.csv2   # for *.csv files separated by ';'
```

tidyverse (specifically the **readr** package)

```
read_delim  # for multiple file-types
read_csv    # for *.csv files
read_csv2   # for *.csv files separated by ';'
read_tsv    # for *.tsv files (tab separated values)
```

There are also functions to upload *.xlsx* files, but we won't be using those.

tidyverse

> The tidyverse is an opinionated collection of R packages designed for data science. All packages share an underlying design philosophy, grammar, and data structures.

In this class we will mostly rely on the tidyverse (vs. base R)

tidyverse

data visualisation

reading data

modern data frames

data wrangling

```
install.packages("tidyverse")
```

Installing/loading **tidyverse** installs the following packages (and dependencies):

- **ggplot2**, for data visualisation.
- **dplyr**, for data manipulation.
- **tidyr**, for data tidying.
- **readr**, for data import.
- **purrr**, for functional programming.
- **tibble**, a modern re-imagining of data frames.
- **stringr**, for strings.
- **forcats**, for factors.

data tidying

string manipulation

functional programming

Exercise

Install the **here** and **tidyverse** packages

Note: linux users may have to install the following dependencies:

```
libcurl4-openssl-dev libssl-dev libxml2-dev
```

Solution

```
install.packages("here")
install.packages("tidyverse")
# or
install.packages(c("here", "tidyverse"))
```

Loading packages

To upload the data, we will use the **here** and **readr** packages

readr is installed with **tidyverse**

Every time we start R(Studio), only essential packages are loaded

This allows R to be light-weight, since it only loads a few packages on startup

We can use the `library` function to load packages

```
library(here)
```

(Not) Loading packages

We can also use functions without loading the package by explicitly calling the package that contains the function

```
# library(readr)
readr::read_csv(here("<path/to/cleaned/data.csv>"))
```

This is useful if we only need a single function from a package

Uploading our data

Now we can use **here** and **readr** to upload our data

```
pits_data_raw <- readr::read_csv(here("data/raw_data/house_pits-clean.csv"))
```

The `here` function ensures that our filepaths always start at the project root,

no matter where we are in the project directory

Note: when uploading data with **tidyverse** functions,

the resulting data will be in a *tibble* (a special type of data frame)

Data uploaded

it can actually be a bit of a headache…

Let's explore!

Metadata

variables	description
East	East grid location of house in feet (excavation grid system)
South	South grid location of house in feet (excavation grid system)
Length	House length in feet
Width	House width in feet
Segment	Three areas within the site 1, 2, 3
Inside	Location within or outside the 'white wall' Inside, Outside
Area	Area in square feet
Points	Number of projectile points
Abraders	Number of abraders
Discs	Number of discs
Earplugs	Number of earplugs
Effigies	Number of effigies
Ceramics	Number of ceramics

First we can take a look at the structure of our data

```
str(pits_data_raw)
```

```
## spec_tbl_df [91 × 13] (S3: spec_tbl_df/tbl_df/tbl/data.frame)
##  $ East    : chr [1:91] "901.39" "973.01" "889.71" "924.16" ...
##  $ South   : num [1:91] 75.1 81.3 163.2 193.1 216.6 ...
##  $ Length  : chr [1:91] "12" "16" "17" "21" ...
##  $ Width   : num [1:91] 12 16 18 21.5 20 16 19 19 8 15.5 ...
##  $ Segment : num [1:91] 2 2 1 1 1 1 1 1 2 2 ...
##  $ Inside  : chr [1:91] "Outside" "Outside" "Inside" "Inside" ...
##  $ Area    : chr [1:91] "144" "256" "306" "451.5" ...
##  $ Points  : chr [1:91] "0" "0" "1" "2" ...
##  $ Abraders: chr [1:91] "1" "0" "0" "1" ...
##  $ Discs   : chr [1:91] "0" "0" "1" "1" ...
##  $ Earplugs: chr [1:91] "0" "0" "0" "1" ...
##  $ Effigies: num [1:91] 0 1 1 0 0 0 0 2 0 1 ...
##  $ Ceramics: chr [1:91] "0" "0" "1" "5" ...
##  - attr(*, "spec")=
##   .. cols(
##   ..   East = col_character(),
##   ..   South = col_double(),
##   ..   Length = col_character(),
##   ..   Width = col_double(),
##   ..   Segment = col_double(),
##   ..   Inside = col_character(),
##   ..   Area = col_character(),
```

See Any issues?

💡 **Hint:** Recall the conversion of vectors.

```
str(pits_data_raw)
```

```
## spec_tbl_df [91 × 13] (S3: spec_tbl_df/tbl_df/tbl/data.frame)
##  $ East    : chr [1:91] "901.39" "973.01" "889.71" "924.16" ...
##  $ South   : num [1:91] 75.1 81.3 163.2 193.1 216.6 ...
##  $ Length  : chr [1:91] "12" "16" "17" "21" ...
##  $ Width   : num [1:91] 12 16 18 21.5 20 16 19 19 8 15.5 ...
##  $ Segment : num [1:91] 2 2 1 1 1 1 1 1 2 2 ...
##  $ Inside  : chr [1:91] "Outside" "Outside" "Inside" "Inside" ...
##  $ Area    : chr [1:91] "144" "256" "306" "451.5" ...
##  $ Points  : chr [1:91] "0" "0" "1" "2" ...
##  $ Abraders: chr [1:91] "1" "0" "0" "1" ...
##  $ Discs   : chr [1:91] "0" "0" "1" "1" ...
##  $ Earplugs: chr [1:91] "0" "0" "0" "1" ...
##  $ Effigies: num [1:91] 0 1 1 0 0 0 0 2 0 1 ...
##  $ Ceramics: chr [1:91] "0" "0" "1" "5" ...
##  - attr(*, "spec")=
##   .. cols(
##   ..   East = col_character(),
##   ..   South = col_double(),
##   ..   Length = col_character(),
##   ..   Width = col_double(),
##   ..   Segment = col_double(),
```

To fix this, we can add an argument to our upload:

```
pits_data_raw <- readr::read_csv("assets/data/house_pits_missouri.csv", na = "N/a")
str(pits_data_raw)
```

```
## spec_tbl_df [91 × 13] (S3: spec_tbl_df/tbl_df/tbl/data.frame)
##  $ East    : num [1:91] 901 973 890 924 912 ...
##  $ South   : num [1:91] 75.1 81.3 163.2 193.1 216.6 ...
##  $ Length  : num [1:91] 12 16 17 21 20.5 16.5 18 21 7.5 19 ...
##  $ Width   : num [1:91] 12 16 18 21.5 20 16 19 19 8 15.5 ...
##  $ Segment : num [1:91] 2 2 1 1 1 1 1 1 2 2 ...
##  $ Inside  : chr [1:91] "Outside" "Outside" "Inside" "Inside" ...
##  $ Area    : num [1:91] 144 256 306 452 410 ...
##  $ Points  : num [1:91] 0 0 1 2 3 0 5 13 0 1 ...
##  $ Abraders: num [1:91] 1 0 0 1 2 0 0 0 0 0 ...
##  $ Discs   : num [1:91] 0 0 1 1 2 0 5 0 0 0 ...
##  $ Earplugs: num [1:91] 0 0 0 1 1 0 0 2 0 0 ...
##  $ Effigies: num [1:91] 0 1 1 0 0 0 0 2 0 1 ...
##  $ Ceramics: num [1:91] 0 0 1 5 4 0 4 12 0 0 ...
##  - attr(*, "spec")=
##   .. cols(
##   ..   East = col_double(),
##   ..   South = col_double(),
##   ..   Length = col_double(),
##   ..   Width = col_double(),
##   ..   Segment = col_double(),
##   ..   Inside = col_character(),
```

The `na` argument in the `readr::read_csv("house_pits_missouri.csv", na = "N/a")` code lets us indicate how missing values are recorded in the raw data

and converts them to the proper format for missing values in R

which is `NA`.

That took care of most of the type issues.

Changing variables

The `Segment` and `Inside` variables are currently *numeric* and *character*,

and should probably be *categorical*.

This can be done in base R with:

```
# We don't want to change the raw data frame...
pits_data <- pits_data_raw # ...so we create a copy
# index the target variable and assign to it the corrected variable
pits_data$Inside <- as.factor(pits_data$Inside)
```

```
pits_data$Inside # print just the 'Inside' variable
```

```
##  [1] Outside Outside Inside  Inside  Inside  Inside  Inside  Inside  Outside
## [10] Outside Outside Outside Outside Outside Inside  Inside  Inside  Inside
## [19] Inside  Inside  Inside  Inside  Outside Outside Outside Outside Outside
## [28] Outside Outside Outside Outside Outside Outside Outside Outside Outside
## [37] Outside Outside Inside  Inside  Inside  Inside  Inside  Inside  Inside
## [46] Inside  Inside  Inside  Inside  Inside  Inside  Inside  Inside  Inside
## [55] Inside  Inside  Inside  Outside Outside Outside Outside Inside  Outside
## [64] Outside Outside Outside Outside Outside Outside Outside Outside Outside
```

Changing variables

The `Segment` and `Inside` variables are currently *numeric* and *character*,

and should probably be *categorical*.

Or with the **tidyverse** using `mutate()`:

```
library(tidyverse)
pits_data <- mutate(pits_data_raw, Inside = as.factor(Inside))
```

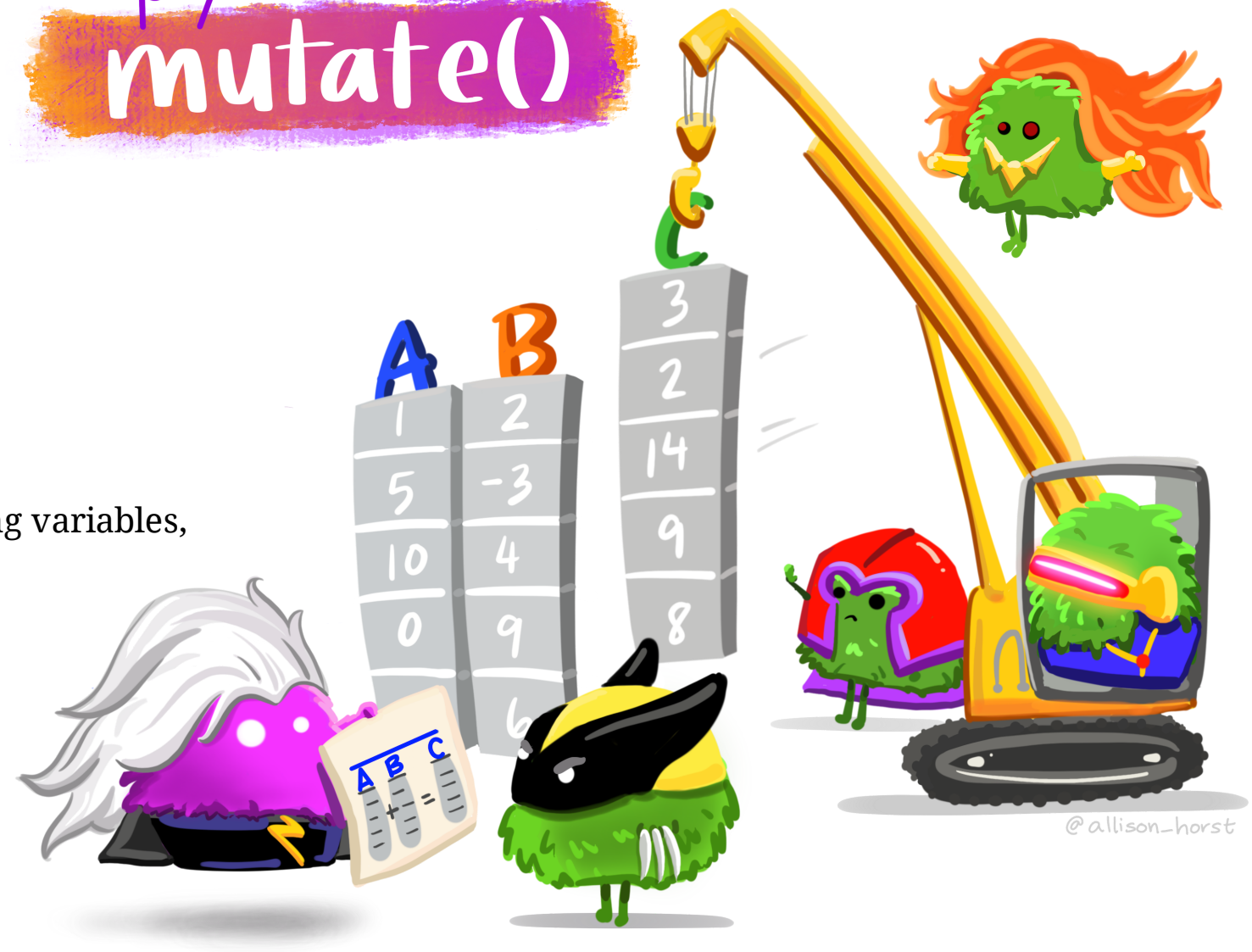
```
pits_data$Inside
```

```
##  [1] Outside Outside Inside  Inside  Inside  Inside  Inside  Inside  Outside
## [10] Outside Outside Outside Outside Outside Inside  Inside  Inside  Inside
## [19] Inside  Inside  Inside  Inside  Outside Outside Outside Outside Outside
## [28] Outside Outside Outside Outside Outside Outside Outside Outside Outside
## [37] Outside Outside Inside  Inside  Inside  Inside  Inside  Inside  Inside
## [46] Inside  Inside  Inside  Inside  Inside  Inside  Inside  Inside  Inside
## [55] Inside  Inside  Inside  Outside Outside Outside Outside Inside  Outside
## [64] Outside Outside Outside Outside Outside Outside Outside Outside Outside
## [73] Outside Outside Outside Outside Outside Outside Outside Outside Outside
## [82] Inside  Inside  Inside  Outside Outside Outside Outside Outside Outside
```

Mutate

`mutate()` allows you to modify existing variables,

and make new variables.



Mutate

We can also change the `Segment` variable in the same function:

```
pits_data <- mutate(pits_data_raw,
                            Inside = as.factor(Inside),
                            Segment = as.factor(Segment))
pits_data$Segment
```

```
##  [1] 2 2 1 1 1 1 1 1 2 2 2 3 3 3 1 1 1 1 1 1 1 1 2 2 2 3 3 3 3 3 3 2 2 2 3 3
## [37] 3 3 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 3 3 3 2 1 2 2 2 2 2 2 2 3 3 3
## [73] 3 3 2 2 2 2 3 3 2 1 1 1 3 3 2 2 2 2 1
## Levels: 1 2 3
```

Mutate

We can also change the `Segment` variable in the same function:

```
pits_data <- mutate(pits_data_raw,
                    Inside = as.factor(Inside),
                    Segment = as.factor(Inside))
pits_data$Segment
```

```
##  [1] Outside Outside Inside  Inside  Inside  Inside  Inside  Inside  Outside
## [10] Outside Outside Outside Outside Outside Inside  Inside  Inside  Inside
## [19] Inside  Inside  Inside  Inside  Outside Outside Outside Outside Outside
## [28] Outside Outside Outside Outside Outside Outside Outside Outside Outside
## [37] Outside Outside Inside  Inside  Inside  Inside  Inside  Inside  Inside
## [46] Inside  Inside  Inside  Inside  Inside  Inside  Inside  Inside  Inside
## [55] Inside  Inside  Inside  Outside Outside Outside Outside Inside  Outside
## [64] Outside Outside Outside Outside Outside Outside Outside Outside Outside
## [73] Outside Outside Outside Outside Outside Outside Outside Outside Outside
## [82] Inside  Inside  Inside  Outside Outside Outside Outside Outside Outside
## [91] Inside
## Levels: Inside Outside
```

Mutate

We can also change the `Segment` variable in the same function:

```
pits_data <- mutate(pits_data_raw,
                    Inside = as.factor(Inside),
                    Segment = as.factor(Inside))
pits_data$Segment
```

So what's going on here? 🧐

```
pits_data <- mutate( <data frame with variables> ,
                     <new variable> = <modification> ,
                     <new variable> = <modification> ,
                     <new variable> = ...etc )
```

Mutate

We can also change the `Segment` variable in the same function:

```
pits_data <- mutate(pits_data_raw,
                    Inside = as.factor(Inside),
                    Segment = as.factor(Inside))
pits_data$Segment
```

We could also have assigned the changes to a new variable:

```
pits_data <- mutate(pits_data_raw,
                    Inside_fct = as.factor(Inside),
                    Segment_fct = as.factor(Inside))
```

Indexing data frames/tibbles

You may have noticed we have been using the 'dollar symbol', $

This is a way to index data frames (and tibbles) by variable name

```
pits_data$Segment
```

```
##  [1] Outside Outside Inside  Inside  Inside  Inside  Inside  Inside  Outside
## [10] Outside Outside Outside Outside Outside Inside  Inside  Inside  Inside
## [19] Inside  Inside  Inside  Inside  Outside Outside Outside Outside Outside
## [28] Outside Outside Outside Outside Outside Outside Outside Outside Outside
## [37] Outside Outside Inside  Inside  Inside  Inside  Inside  Inside  Inside
## [46] Inside  Inside  Inside  Inside  Inside  Inside  Inside  Inside  Inside
## [55] Inside  Inside  Inside  Outside Outside Outside Outside Inside  Outside
## [64] Outside Outside Outside Outside Outside Outside Outside Outside Outside
## [73] Outside Outside Outside Outside Outside Outside Outside Outside Outside
## [82] Inside  Inside  Inside  Outside Outside Outside Outside Outside Outside
## [91] Inside
## Levels: Inside Outside
```

Indexing data frames/tibbles

We can also use square brackets: `dataframe[<rows>,<columns>]`

```
pits_data[,5] # index all rows and 5th column
```

```
## # A tibble: 91 × 1
##    Segment
##    <fct>
##  1 Outside
##  2 Outside
##  3 Inside
##  4 Inside
##  5 Inside
##  6 Inside
##  7 Inside
##  8 Inside
##  9 Outside
## 10 Outside
## # … with 81 more rows
```

Indexing data frames/tibbles

We can also use square brackets: `dataframe[<rows>,<columns>]`

```
pits_data[,5] # index all rows and 5th column
```

We needed to know that the `Segment` variable was the 5th column,

so sometimes it's easier to index by name

What's the difference between the two outputs? (hint: use `class` function)

Summary statistics

We can summarise our data to get an overview.

Let's see what the `mean` value is for the `Area` variable:

```
#summarise(pits_data, mean = mean(Area))
mean(pits_data$Area)
```

```
## [1] NA
```

`NA`? That doesn't seem right...

```
mean(pits_data$Area, na.rm = T) # that's better
```

```
## [1] 237.5678
```

Summary statistics

```
mean(pits_data$Area, na.rm = T)
```

```
## [1] 237.5678
```

This is not particularly informative.

Summary statistics continued

More interesting would be to see if there is a difference between houses located `Inside` or `Outside` the "white wall".

To achieve this, we could `filter` the data before taking the `mean`:

```
filter(pits_data, Inside == "Inside") # filter with conditional statement
```

```
## # A tibble: 38 × 13
##     East South Length Width Segment Inside  Area Points Abraders Discs
##    <dbl> <dbl>  <dbl> <dbl> <fct>   <fct>  <dbl>  <dbl>    <dbl> <dbl>
##  1  890.  163.   17    18   Inside  Inside  306       1        0     1
##  2  924.  193.   21    21.5 Inside  Inside  452.      2        1     1
##  3  912.  217.   20.5  20   Inside  Inside  410       3        2     2
##  4  940.  251.   16.5  16   Inside  Inside  264       0        0     0
##  5  948.  229.   18    19   Inside  Inside  342       5        0     5
##  6  962.  212.   21    19   Inside  Inside  399      13        0     0
##  7  765.  297.   16.5  17   Inside  Inside  280.      3        1     2
##  8  787.  256.   21    20.5 Inside  Inside  430.      8        1     7
##  9  796.  228.   19    19   Inside  Inside  361       7        1    11
## 10  809.  204.   20    19   Inside  Inside  380       6        7     4
## # … with 28 more rows, and 3 more variables: Earplugs <dbl>, Effigies <dbl>,
```

Summary statistics continued

More interesting would be to see if there is a difference between houses located `Inside` or `Outside` the "white wall".

```
pits_data_inside <- filter(pits_data, Inside == "Inside")
mean(pits_data_inside$Area, na.rm = T)
```

```
## [1] 317.3711
```

Summary statistics continued

More interesting would be to see if there is a difference between houses located `Inside` or `Outside` the "white wall".

And now the houses 'Outside':

```
pits_data_outside <- filter(pits_data, Inside == "Outside")
mean(pits_data_outside$Area, na.rm = T)
```

```
## [1] 179.25
```

Summary statistics continued

More interesting would be to see if there is a difference between houses located `Inside` or `Outside` the "white wall".

Then compare

```
mean(pits_data_outside$Area, na.rm = T)
```

```
## [1] 179.25
```

```
mean(pits_data_inside$Area, na.rm = T)
```

```
## [1] 317.3711
```

There's got to be an easier way to `summarise`... 😉

Break!

Summarise

Enter the `summarise` and `group_by` functions from **dplyr**!

A more convenient way to get summary statistics for different groups,

is to use the `group_by` function.

This allows you to use a group variables (factors) without `filter`ing:

```
pits_data_grouped <- group_by(pits_data, Inside)

summarise(pits_data_grouped,
          mean = mean(Area, na.rm = T)) # summarise with the mean function
```

```
## # A tibble: 2 × 2
##   Inside   mean
##   <fct>   <dbl>
## 1 Inside   317.
## 2 Outside  179.
```

The `summarise` function provides a new *tibble* with the summary statistics.

Summary statistics continued, the final chapter, the sequel

What did we do?

```
pits_data_outside <- group_by( <data to group> ,
                                    <variable(s) to group by> )

summarise(<data to summarise>,
          <new variable for output> = <function>(<variable>) )
```

Exercise

Get the mean values for the `Points` variable for houses and `group_by` the `Inside` variable.

Solution

```
pits_data_grouped <- group_by(pits_data, Inside)
summarise(pits_data_grouped, mean = mean(Points, na.rm = T))
```

```
## # A tibble: 2 × 2
##   Inside   mean
##   <fct>   <dbl>
## 1 Inside  4.92
## 2 Outside 0.804
```

Renaming variables

The `Inside` variable name is really starting to annoy me...

💡 The variable name should not be the same as a value within that variable.

let's `rename` it:

```
pits_data_renamed <- rename(pits_data, Location = Inside)
```

Pipes

We have accumulated a few intermediate data frames.

```
ls() # list all elements in the Global Environment
```

```
## [1] "pits_data"         "pits_data_grouped" "pits_data_inside"
## [4] "pits_data_outside" "pits_data_raw"     "pits_data_renamed"
```

This can be avoided (to some extent) with the 'pipe' operator %>%

(loaded with tidyverse).

Pipes

The pipe allows us to perform multiple operations in a single ...

Let's recap what we have done so far:

```
pits_data <- pits_data_raw %>%
  mutate(Segment = as_factor(Segment),
         Inside = as_factor(Inside)) %>%
  rename(Location = Inside)
# We don't want to store the summary output
pits_data %>%
  group_by(Location) %>%
  summarise(mean = mean(Points, na.rm = T))
```

```
## # A tibble: 2 × 2
##   Location  mean
##   <fct>    <dbl>
## 1 Outside  0.804
## 2 Inside   4.92
```

The data `pits_data_raw` is 'piped' through to all subsequent functions.

Pipes

Think of it as saying

```
take 'pits_data', and then
  filter by 'Inside', and then
  take the mean Area of all 'Inside'
```

Exercise

Find the `median` values of `Ceramics` for each `Segment`

Solution

```
pits_data %>%
  group_by(Segment) %>%
  summarise(median = median(Ceramics, na.rm = T))
```

```
## # A tibble: 3 × 2
##   Segment median
##   <fct>    <dbl>
## 1 1            3
## 2 2            0
## 3 3            0
```

New variables

We may be interested in creating a new variable that has the total number of artifacts.

So for each row, we will need the sum of artifact variables.

```
pits_data %>%
  rowwise() %>% # we want to calculate each row separately
  mutate(total = sum(
    c_across(Points:Ceramics), # we want to sum these variables
    na.rm = T))
```

```
## # A tibble: 91 × 10
## # Rowwise:
##      East South  Area Points Abraders Discs Earplugs Effigies Ceramics total
##     <dbl> <dbl> <dbl>  <dbl>    <dbl> <dbl>    <dbl>    <dbl>    <dbl> <dbl>
##  1  901.  75.1  144        0        1     0        0        0        0     1
##  2  973.  81.3  256        0        0     0        0        1        0     1
##  3  890. 163.   306        1        0     1        0        1        1     4
##  4  924. 193.   452.       2        1     1        1        0        5    10
##  5  912. 217.   410        3        2     2        1        0        4    12
##  6  940. 251.   264        0        0     0        0        0        0     0
##  7  948. 229.   342        5        0     5        0        0        4    14
```

New variables

The 'tidyverse' way is not always easiest...

```
pits_data$total <- rowSums(pits_data[,8:13], na.rm = T)
```

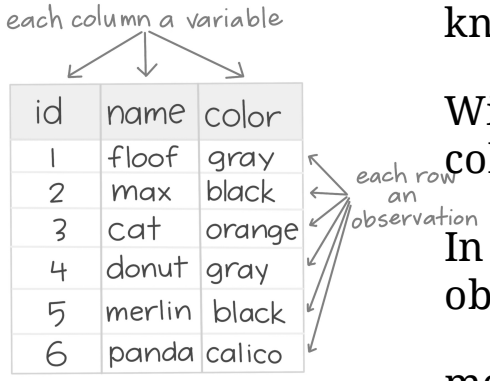
```
## # A tibble: 91 × 9
##     East South  Area Points Abraders Discs Earplugs Effigies Ceramics
##    <dbl> <dbl> <dbl>  <dbl>    <dbl> <dbl>    <dbl>    <dbl>    <dbl>
##  1  901.  75.1  144        0        1     0        0        0        0
##  2  973.  81.3  256        0        0     0        0        1        0
##  3  890. 163.   306        1        0     1        0        1        1
##  4  924. 193.   452.       2        1     1        1        0        5
##  5  912. 217.   410        3        2     2        1        0        4
##  6  940. 251.   264        0        0     0        0        0        0
##  7  948. 229.   342        5        0     5        0        0        4
##  8  962. 212.   399       13        0     0        2        2       12
##  9  979. 194.    60        0        0     0        0        0        0
## 10  992. 153.   217        1        0     0        0        1        0
## # … with 81 more rows
```

"TIDY DATA is a standard way of mapping the meaning of a dataset to its structure."

–HADLEY WICKHAM

In tidy data:

- each variable forms a column
- each observation forms a row
- each cell is a single measurement



Wickham, H. (2014). Tidy Data. Journal of Statistical Software *59* (10). DOI: 10.18637/jss.v059.i10

Tidy data

The current data format we are working with is known as 'wide'.

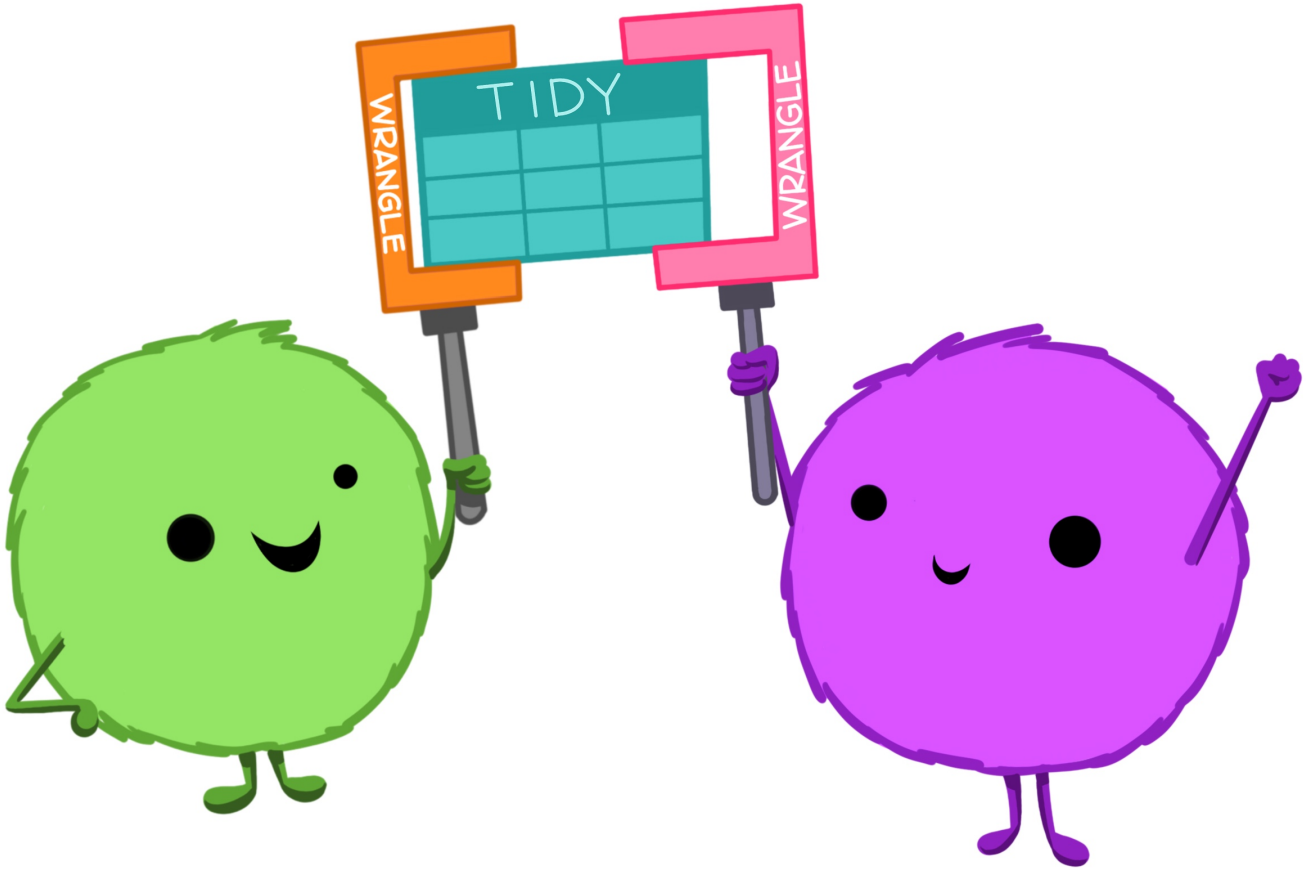
Wide data has individuals as rows, and variables as columns.

In tidy data (a.k.a. 'long' data), the result of a single observation forms a row,

meaning the results from a single individual may span multiple rows.

Tidy data

If all data are structured the same way, it is easier to collaborate! 🥂



Pivoting

There are two pivot functions to switch between 'long' and 'wide'

- `pivot_longer`
- `pivot_wider`

Pretty self-explanatory...

Pivoting

Let's try pivoting our data with `pivot_longer`

- `cols` is a vector of the variables we want to transform
- `names_to` is the name of the new column where we want to store the variable names
- `values_to` is the new column where we want to put the values of our variables

```
pits_data_long <- pits_data %>%
  pivot_longer(cols = Points:Ceramics, names_to = "artifact", values_to = "count")
```

```
## # A tibble: 546 × 9
##     East South Length Width Segment Location  Area artifact count
##    <dbl> <dbl>  <dbl> <dbl> <fct>   <fct>    <dbl> <chr>    <dbl>
##  1  901.  75.1     12    12 2       Outside    144 Points       0
##  2  901.  75.1     12    12 2       Outside    144 Abraders     1
##  3  901.  75.1     12    12 2       Outside    144 Discs        0
##  4  901.  75.1     12    12 2       Outside    144 Earplugs     0
##  5  901.  75.1     12    12 2       Outside    144 Effigies     0
##  6  901.  75.1     12    12 2       Outside    144 Ceramics     0
##  7  973.  81.3     16    16 2       Outside    256 Points       0
```

What just happened?

We have combined all the artifact variables to two columns.

By doing this we have made the data frame longer.

East	South	Length	Width	Area	artifact	count
901.39	75.07	12.0	12.0	144.00	Points	0
901.39	75.07	12.0	12.0	144.00	Abraders	1
901.39	75.07	12.0	12.0	144.00	Discs	0
901.39	75.07	12.0	12.0	144.00	Earplugs	0
901.39	75.07	12.0	12.0	144.00	Effigies	0
901.39	75.07	12.0	12.0	144.00	Ceramics	0
973.01	81.33	16.0	16.0	256.00	Points	0
973.01	81.33	16.0	16.0	256.00	Abraders	0
973.01	81.33	16.0	16.0	256.00	Discs	0
973.01	81.33	16.0	16.0	256.00	Earplugs	0
973.01	81.33	16.0	16.0	256.00	Effigies	1
973.01	81.33	16.0	16.0	256.00	Ceramics	0
889.71	163.21	17.0	18.0	306.00	Points	1
889.71	163.21	17.0	18.0	306.00	Abraders	0
889.71	163.21	17.0	18.0	306.00	Discs	1
889.71	163.21	17.0	18.0	306.00	Earplugs	0
889.71	163.21	17.0	18.0	306.00	Effigies	1

What just happened?

We have combined all the artifact variables to two columns.

By doing this we have made the data frame longer.

This gives us a row for every type of artifact within a house.

East	South	Length	Width	Area	artifact	count
901.39	75.07	12.0	12.0	144.00	Points	0
901.39	75.07	12.0	12.0	144.00	Abraders	1
901.39	75.07	12.0	12.0	144.00	Discs	0
901.39	75.07	12.0	12.0	144.00	Earplugs	0
901.39	75.07	12.0	12.0	144.00	Effigies	0
901.39	75.07	12.0	12.0	144.00	Ceramics	0
973.01	81.33	16.0	16.0	256.00	Points	0
973.01	81.33	16.0	16.0	256.00	Abraders	0
973.01	81.33	16.0	16.0	256.00	Discs	0
973.01	81.33	16.0	16.0	256.00	Earplugs	0
973.01	81.33	16.0	16.0	256.00	Effigies	1
973.01	81.33	16.0	16.0	256.00	Ceramics	0
889.71	163.21	17.0	18.0	306.00	Points	1
889.71	163.21	17.0	18.0	306.00	Abraders	0
889.71	163.21	17.0	18.0	306.00	Discs	1
889.71	163.21	17.0	18.0	306.00	Earplugs	0
889.71	163.21	17.0	18.0	306.00	Effigies	1

Pivoting

We can restore it to the wide format using `pivot_wider`

We use `names_from` to re-create the variable names

and `values_from` to re-create the values.

```
pits_data_long %>%
  pivot_wider(names_from = artifact, values_from = count)
```

```
## # A tibble: 91 × 13
##     East South Length Width Segment Location  Area Points Abraders Discs
##    <dbl> <dbl>  <dbl> <dbl> <fct>   <fct>    <dbl>  <dbl>    <dbl> <dbl>
##  1  901.  75.1   12    12    2       Outside   144       0        1     0
##  2  973.  81.3   16    16    2       Outside   256       0        0     0
##  3  890. 163.    17    18    1       Inside    306       1        0     1
##  4  924. 193.    21    21.5  1       Inside    452.      2        1     1
##  5  912. 217.    20.5  20    1       Inside    410       3        2     2
##  6  940. 251.    16.5  16    1       Inside    264       0        0     0
##  7  948. 229.    18    19    1       Inside    342       5        0     5
##  8  962. 212.    21    19    1       Inside    399      13        0     0
##  9  979. 194.     7.5   8    2       Outside    60       0        0     0
```

Summarising long format

We can now easily summarise the artifact counts.

```
pits_data_long %>%
  group_by(artifact) %>%
  summarise(total = sum(count, na.rm = T))
```

```
## # A tibble: 6 × 2
##   artifact total
##   <chr>    <dbl>
## 1 Abraders    32
## 2 Ceramics   206
## 3 Discs       73
## 4 Earplugs    32
## 5 Effigies    26
## 6 Points     228
```

Again, not particularly informative...

Exercise

Find the `mean` number of each `artifact` for all levels of the `Location` variable.

💡 **Hint:** `group_by` accepts multiple arguments.

Solution

```
pits_data_long %>%
  group_by(artifact, Location) %>%
  summarise(mean = mean(count, na.rm = T))
```

```
## `summarise()` has grouped output by 'artifact'. You can override using the
## `.groups` argument.
```

```
## # A tibble: 12 × 3
## # Groups:   artifact [6]
##    artifact Location   mean
##    <chr>    <fct>     <dbl>
##  1 Abraders Outside  0.212
##  2 Abraders Inside   0.553
##  3 Ceramics Outside  0.566
```

Exporting the processed data

We can save our data as a `.csv` file using `write_csv`

```
write_csv(pits_data, file = here("data/derived_data/pits-data.csv"))

write_csv(pits_data_long, file = here("data/derived_data/pits-data_long.csv"))
```