University of BRISTOL

Tidy data and iteration

Using the tidyverse to transform your data II

Henry W J Reeve

henry.reeve@bristol.ac.uk

Statistical Computing & Empirical Methods (EMATM0061)

MSc in Data Science, Teaching block 1, 2021.

What will we cover today?

- We will learn about TidyData!
- We will see how to reshape our data frames with the pivot functions.
- We will also look at uniting and separating columns within data.
- We will see how to use the map function for efficient iteration in R.
- We will also look at some basic methods for handling missing data.

What is tidy data?

Species	Island	Bill length (mm)	Bill depth (mm)	Flipper length (mm)	Body mass (g)
Adelie	Dream	39.70	17.90	193.00	4250
Adelie	Dream	39.60	18.80	190.00	4600
Adelie	Dream	39.20	21.10	196.00	4150
Adelie	Biscoe	35.30	18.90	187.00	3800
Adelie	Dream	36.50	18.00	182.00	3150
	Average	38.06	18.94	189.60	3990
Chinstrap	Dream	51.30	19.20	193.00	3650
Chinstrap	Dream	46.50	17.90	192.00	3500
Chinstrap	Dream	49.00	19.60	212.00	4300
Chinstrap	Dream	50.80	19.00	210.00	4100
Chinstrap	Dream	45.90	17.10	190.00	3575
	Average	48.70	18.56	199.40	3825
Gentoo	Biscoe	44.40	17.30	219.00	5250
Gentoo	Biscoe	50.80	17.30	228.00	5600
Gentoo	Biscoe	50.40	15.70	222.00	5750
Gentoo	Biscoe	45.80	14.20	219.00	4700
Gentoo	Biscoe	55.90	17.00	228.00	5600
	Average	49.46	16.30	223.20	5380
	Overall average	45.41	17.93	204.07	4398

What is tidy data?

Tidy data has two important features:

1. Each row corresponds *to a* specific and unique observation representing a similar sort of thing.

2. Columns correspond to single variables with the same sort of value for each observation.

What is tidy data?

- This is tidy data!

species	island	bill_length_mm	bill_depth_mm	flipper_length_mm	body_mass_g	sex	year
Adelie	Torgersen	40.9	16.8	191	3700	female	2008
Adelie	Biscoe	37.8	20	190	4250	male	2009
Adelie	Dream	36.9	18.6	189	3500	female	2008
Adelie	Torgersen	34.6	17.2	189	3200	female	2008
Adelie	Dream	38.8	20	190	3950	male	2007
Chinstrap	Dream	46.4	17.8	191	3700	female	2008
Chinstrap	Dream	58	17.8	181	3700	female	2007
Chinstrap	Dream	45.6	19.4	194	3525	female	2009
Chinstrap	Dream	52	20.7	210	4800	male	2008
Chinstrap	Dream	52.7	19.8	197	3725	male	2007
Gentoo	Biscoe	43.5	14.2	220	4700	female	2008
Gentoo	Biscoe	45.4	14.6	211	4800	female	2007
Gentoo	Biscoe	46.3	15.8	215	5050	male	2007
Gentoo	Biscoe	50.5	15.9	225	5400	male	2008
Gentoo	Biscoe	49	16.1	216	5550	male	2007

1. Each row corresponds *to a* specific and unique observation representing a similar sort of thing.
2. Columns correspond to single variables with the same sort of value for each observation.

What is tidy data?

- This is NOT tidy data!

Species	Island	Bill length (mm)	Bill depth (mm)	Flipper length (mm)	Body mass (g)
Adelie	Dream	39.70	17.90	193.00	4250
Adelie	Dream	39.60	18.80	190.00	4600
Adelie	Dream	39.20	21.10	196.00	4150
Adelie	Biscoe	35.30	18.90	187.00	3800
Adelie	Dream	36.50	18.00	182.00	3150
	Average	38.06	18.94	189.60	3990
Chinstrap	Dream	51.30	19.20	193.00	3650
Chinstrap	Dream	46.50	17.90	192.00	3500
Chinstrap	Dream	49.00	19.60	212.00	4300
Chinstrap	Dream	50.80	19.00	210.00	4100
Chinstrap	Dream	45.90	17.10	190.00	3575
	Average	48.70	18.56	199.40	3825
Gentoo	Biscoe	44.40	17.30	219.00	5250
Gentoo	Biscoe	50.80	17.30	228.00	5600
Gentoo	Biscoe	50.40	15.70	222.00	5750
Gentoo	Biscoe	45.80	14.20	219.00	4700
Gentoo	Biscoe	55.90	17.00	228.00	5600
	Average	49.46	16.30	223.20	5380
	Overall average	45.41	17.93	204.07	4398

1. Each row corresponds *to a* specific and unique observation representing a similar sort of thing.
2. Columns correspond to single variables with the same sort of value for each observation.

What is tidy data?

1. Each row corresponds *to a* specific and unique observation representing a similar sort of thing.

2. Columns correspond to variables with the same sort of value in each row.

Tidy data is typically far easier to manipulate and apply statistical analysis to

Note: “Tidy data” here is a technical term ie. not just “data that is tidy”/ “data that is well presented”.

Non-tidy data

Tidy data is typically far easier to manipulate and apply statistical analysis to in R.

In other contexts, non-tidy data has several advantages:

- Non-tidy data can be more accessible visually for non-specialists.
- Non-tidy can offer substantial performance and space advantages in certain contexts.
- Specialist fields e.g. computer vision often have unique standards for storing data.

Reshaping data

Narrow data

```
## # A tibble: 9 x 3
##   species   property  value
##   <fct>     <chr>     <dbl>
## 1 Adelie    bill       38.8
## 2 Adelie    flipper   190.
## 3 Adelie    weight   3701.
## 4 Chinstrap bill       48.8
## 5 Chinstrap flipper   196.
## 6 Chinstrap weight   3733.
## 7 Gentoo    bill       47.5
## 8 Gentoo    flipper   217.
## 9 Gentoo    weight   5076.
```

Wide data

```
## # A tibble: 3 x 4
##   species    bill flipper weight
##   <fct>     <dbl>   <dbl>  <dbl>
## 1 Adelie     38.8    190.  3701.
## 2 Chinstrap  48.8    196.  3733.
## 3 Gentoo     47.5    217.  5076.
```

Reshaping data

Let’s suppose we have data represented within a narrow format.

```
penguins_summary_narrow
```

```
## # A tibble: 9 x 3
##   species   property  value
##   <fct>     <chr>     <dbl>
## 1 Adelie    bill       38.8
## 2 Adelie    flipper   190.
## 3 Adelie    weight   3701.
## 4 Chinstrap bill       48.8
## 5 Chinstrap flipper   196.
## 6 Chinstrap weight   3733.
## 7 Gentoo    bill       47.5
## 8 Gentoo    flipper   217.
## 9 Gentoo    weight   5076.
```

Is this tidy data?

Reshaping data

```
penguins_summary_narrow
```

```
## # A tibble: 9 x 3
##   species   property  value
##   <fct>     <chr>     <dbl>
## 1 Adelie    bill       38.8
## 2 Adelie    flipper   190.
## 3 Adelie    weight   3701.
## 4 Chinstrap bill       48.8
## 5 Chinstrap flipper   196.
## 6 Chinstrap weight   3733.
## 7 Gentoo    bill       47.5
## 8 Gentoo    flipper   217.
## 9 Gentoo    weight   5076.
```

```
penguins_summary_wide<-penguins_summary_narrow%>%
  pivot_wider(names_from=property,values_from=value)
```

```
penguins_summary_wide
```

```
## # A tibble: 3 x 4
##   species    bill flipper weight
##   <fct>     <dbl>   <dbl>  <dbl>
## 1 Adelie     38.8    190.  3701.
## 2 Chinstrap  48.8    196.  3733.
## 3 Gentoo     47.5    217.  5076.
```

We can use pivot functions to efficiently reshape our data

Reshaping data

We can use pivot functions to efficiently reshape our data

```
penguins_summary_wide<-penguins_summary_narrow%>%
  pivot_wider(names_from=property,values_from=value)
```

```
penguins_summary_wide
```

```
## # A tibble: 3 x 4
##   species    bill flipper weight
##   <fct>     <dbl>   <dbl>  <dbl>
## 1 Adelie     38.8    190.  3701.
## 2 Chinstrap  48.8    196.  3733.
## 3 Gentoo     47.5    217.  5076.
```

```
penguins_summary_wide%>%
  pivot_longer(cols=c("bill","flipper","weight"), names_to = "property", values_to = "value")
```

```
## # A tibble: 9 x 3
##   species   property  value
##   <fct>     <chr>     <dbl>
## 1 Adelie    bill       38.8
## 2 Adelie    flipper   190.
## 3 Adelie    weight   3701.
## 4 Chinstrap bill       48.8
## 5 Chinstrap flipper   196.
## 6 Chinstrap weight   3733.
## 7 Gentoo    bill       47.5
## 8 Gentoo    flipper   217.
## 9 Gentoo    weight   5076.
```

Reshaping data

```
penguins_summary_wide%>%
  pivot_longer(cols=c("bill","flipper","weight"), names_to = "property", values_to = "value")
```

```
## # A tibble: 9 x 3
##   species   property  value
##   <fct>     <chr>     <dbl>
## 1 Adelie    bill       38.8
## 2 Adelie    flipper   190.
## 3 Adelie    weight   3701.
## 4 Chinstrap bill       48.8
## 5 Chinstrap flipper   196.
## 6 Chinstrap weight   3733.
## 7 Gentoo    bill       47.5
## 8 Gentoo    flipper   217.
## 9 Gentoo    weight   5076.
```

```
penguins_summary_wide%>%
  pivot_longer(cols=!species, names_to = "property", values_to = "value")
```

```
## # A tibble: 9 x 3
##   species   property  value
##   <fct>     <chr>     <dbl>
## 1 Adelie    bill       38.8
## 2 Adelie    flipper   190.
## 3 Adelie    weight   3701.
## 4 Chinstrap bill       48.8
## 5 Chinstrap flipper   196.
## 6 Chinstrap weight   3733.
## 7 Gentoo    bill       47.5
## 8 Gentoo    flipper   217.
## 9 Gentoo    weight   5076.
```

Reshaping data

Narrow data

```
## # A tibble: 9 x 3
##   species   property value
##   <fct>     <chr>    <dbl>
## 1 Adelie    bill      38.8
## 2 Adelie    flipper  190.
## 3 Adelie    weight  3701.
## 4 Chinstrap bill      48.8
## 5 Chinstrap flipper  196.
## 6 Chinstrap weight  3733.
## 7 Gentoo    bill      47.5
## 8 Gentoo    flipper  217.
## 9 Gentoo    weight  5076.
```

Pivot wider

Pivot longer

Wide data

```
## # A tibble: 3 x 4
##   species    bill flipper weight
##   <fct>     <dbl>   <dbl>  <dbl>
## 1 Adelie     38.8    190.  3701.
## 2 Chinstrap  48.8    196.  3733.
## 3 Gentoo     47.5    217.  5076.
```

Pivoting for tidy data

Suppose we have access to data about chess tournaments spread across two data frames:

```
wins_df_wide
```

```
## # A tibble: 3 x 3
##   name    `2018` `2019`
##   <chr>    <dbl>  <dbl>
## 1 Alice        5      9
## 2 Bob          9      7
## 3 Charlie      3      2
```

```
losses_df_wide
```

```
## # A tibble: 3 x 3
##   name    `2018` `2019`
##   <chr>    <dbl>  <dbl>
## 1 Alice        7      8
## 2 Bob         15      4
## 3 Charlie     12     10
```

Data stored in this way is very common within spreadsheets.

However, this format makes analysis difficult.

<u>Example</u> How can we compute the win rate per participant?

Pivoting for tidy data

```
wins_df_wide
```

```
## # A tibble: 3 x 3
##   name    `2018` `2019`
##   <chr>    <dbl>  <dbl>
## 1 Alice        5      9
## 2 Bob          9      7
## 3 Charlie      3      2
```

```
wins_df_narrow<-wins_df_wide%>%
  pivot_longer(!name,names_to="year",values_to="wins")
```

```
wins_df_narrow
```

```
## # A tibble: 6 x 3
##   name    year   wins
##   <chr>   <chr> <dbl>
## 1 Alice   2018      5
## 2 Alice   2019      9
## 3 Bob     2018      9
## 4 Bob     2019      7
## 5 Charlie 2018      3
## 6 Charlie 2019      2
```

Pivoting for tidy data

```
losses_df_wide
```

```
## # A tibble: 3 x 3
##   name    `2018` `2019`
##   <chr>    <dbl>  <dbl>
## 1 Alice        7      8
## 2 Bob         15      4
## 3 Charlie     12     10
```

```
losses_df_narrow<-losses_df_wide%>%
  pivot_longer(!name,names_to="year",values_to="losses")
```

```
losses_df_narrow
```

```
## # A tibble: 6 x 3
##   name    year  losses
##   <chr>   <chr>  <dbl>
## 1 Alice   2018       7
## 2 Alice   2019       8
## 3 Bob     2018      15
## 4 Bob     2019       4
## 5 Charlie 2018      12
## 6 Charlie 2019      10
```

Pivoting for tidy data

```
wins_df_narrow
```

```
## # A tibble: 6 x 3
##   name    year   wins
##   <chr>   <chr> <dbl>
## 1 Alice   2018      5
## 2 Alice   2019      9
## 3 Bob     2018      9
## 4 Bob     2019      7
## 5 Charlie 2018      3
## 6 Charlie 2019      2
```

```
losses_df_narrow
```

```
## # A tibble: 6 x 3
##   name    year  losses
##   <chr>   <chr>  <dbl>
## 1 Alice   2018       7
## 2 Alice   2019       8
## 3 Bob     2018      15
## 4 Bob     2019       4
## 5 Charlie 2018      12
## 6 Charlie 2019      10
```

```
wins_losses_df<-inner_join(wins_df_narrow,losses_df_narrow)%>%
  mutate(win_rate=wins/(wins+losses))
```

```
wins_losses_df
```

```
## # A tibble: 6 x 5
##   name    year   wins losses win_rate
##   <chr>   <chr> <dbl>  <dbl>    <dbl>
## 1 Alice   2018      5      7    0.417
## 2 Alice   2019      9      8    0.529
## 3 Bob     2018      9     15    0.375
## 4 Bob     2019      7      4    0.636
## 5 Charlie 2018      3     12    0.2
## 6 Charlie 2019      2     10    0.167
```

Uniting and separating data

United

```
## # A tibble: 6 x 3
##   name    year  w_over_t
##   <chr>   <chr> <chr>
## 1 Alice   2018  5/12
## 2 Alice   2019  9/17
## 3 Bob     2018  9/24
## 4 Bob     2019  7/11
## 5 Charlie 2018  3/15
## 6 Charlie 2019  2/12
```

Separate

Unite

Separated

```
## # A tibble: 6 x 4
##   name    year   wins totals
##   <chr>   <chr> <int>  <int>
## 1 Alice   2018      5     12
## 2 Alice   2019      9     17
## 3 Bob     2018      9     24
## 4 Bob     2019      7     11
## 5 Charlie 2018      3     15
## 6 Charlie 2019      2     12
```

Multiple variables within a column

We often encounter data with multiple variables within a single column.

```
wins_over_total_df
```

```
## # A tibble: 6 x 3
##   name    year  w_over_t
##   <chr>   <chr> <chr>
## 1 Alice   2018  5/12
## 2 Alice   2019  9/17
## 3 Bob     2018  9/24
## 4 Bob     2019  7/11
## 5 Charlie 2018  3/15
## 6 Charlie 2019  2/12
```

We need to extract the individual variables for tasks such as statistical analysis and visualisation.

The separate function

```
wins_over_total_df
```

```
## # A tibble: 6 x 3
##   name    year  w_over_t
##   <chr>   <chr> <chr>
## 1 Alice   2018  5/12
## 2 Alice   2019  9/17
## 3 Bob     2018  9/24
## 4 Bob     2019  7/11
## 5 Charlie 2018  3/15
## 6 Charlie 2019  2/12
```

```
sep_df<-wins_over_total_df%>%
  separate(w_over_t,into=c("wins","totals"), sep="/")
```

```
sep_df
```

```
## # A tibble: 6 x 4
##   name    year  wins  totals
##   <chr>   <chr> <chr> <chr>
## 1 Alice   2018  5     12
## 2 Alice   2019  9     17
## 3 Bob     2018  9     24
## 4 Bob     2019  7     11
## 5 Charlie 2018  3     15
## 6 Charlie 2019  2     12
```

The separate function

```r
sep_df<-wins_over_total_df%>%
  separate(w_over_t,into=c("wins","totals"), sep="/")
```

```r
sep_df
```

```
## # A tibble: 6 x 4
##   name    year  wins  totals
##   <chr>   <chr> <chr> <chr>
## 1 Alice   2018  5     12
## 2 Alice   2019  9     17
## 3 Bob     2018  9     24
## 4 Bob     2019  7     11
## 5 Charlie 2018  3     15
## 6 Charlie 2019  2     12
```

```r
sep_df%>%
  mutate(losses=totals-wins)%>%
  select(-totals)
```

Error: Subtraction requires numerical variables!

The separate function

```
sep_df<-wins_over_total_df%>%
  separate(w_over_t,into=c("wins","totals"), sep="/")
```

```
sep_df
```

```
## # A tibble: 6 x 4
##   name    year  wins  totals
##   <chr>   <chr> <chr> <chr>
## 1 Alice   2018  5     12
## 2 Alice   2019  9     17
## 3 Bob     2018  9     24
## 4 Bob     2019  7     11
## 5 Charlie 2018  3     15
## 6 Charlie 2019  2     12
```

```
sep_df<-wins_over_total_df%>%
  separate(w_over_t,into=c("wins","totals"), sep="/", convert=TRUE)
```

```
sep_df
```

```
## # A tibble: 6 x 4
##   name    year   wins totals
##   <chr>   <chr> <int>  <int>
## 1 Alice   2018      5     12
## 2 Alice   2019      9     17
## 3 Bob     2018      9     24
## 4 Bob     2019      7     11
## 5 Charlie 2018      3     15
## 6 Charlie 2019      2     12
```

The separate function

```
sep_df
```

```
## # A tibble: 6 x 4
##   name    year   wins totals
##   <chr>   <chr> <int>  <int>
## 1 Alice   2018      5     12
## 2 Alice   2019      9     17
## 3 Bob     2018      9     24
## 4 Bob     2019      7     11
## 5 Charlie 2018      3     15
## 6 Charlie 2019      2     12
```

```
sep_df%>%
  mutate(losses=totals-wins)%>%
  select(-totals)
```

```
## # A tibble: 6 x 4
##   name    year   wins losses
##   <chr>   <chr> <int>  <int>
## 1 Alice   2018      5      7
## 2 Alice   2019      9      8
## 3 Bob     2018      9     15
## 4 Bob     2019      7      4
## 5 Charlie 2018      3     12
## 6 Charlie 2019      2     10
```

The unite function

```
sep_df
```

```
## # A tibble: 6 x 4
##   name    year   wins totals
##   <chr>   <chr> <int>  <int>
## 1 Alice   2018      5     12
## 2 Alice   2019      9     17
## 3 Bob     2018      9     24
## 4 Bob     2019      7     11
## 5 Charlie 2018      3     15
## 6 Charlie 2019      2     12
```

```
uni_df<-sep_df%>%
  unite(w_over_t,wins,totals,sep="/")
```

```
uni_df
```

```
## # A tibble: 6 x 3
##   name    year  w_over_t
##   <chr>   <chr> <chr>
## 1 Alice   2018  5/12
## 2 Alice   2019  9/17
## 3 Bob     2018  9/24
## 4 Bob     2019  7/11
## 5 Charlie 2018  3/15
## 6 Charlie 2019  2/12
```

Uniting and separating data

United

```
## # A tibble: 6 x 3
##   name    year  w_over_t
##   <chr>   <chr> <chr>
## 1 Alice   2018  5/12
## 2 Alice   2019  9/17
## 3 Bob     2018  9/24
## 4 Bob     2019  7/11
## 5 Charlie 2018  3/15
## 6 Charlie 2019  2/12
```

Separate

Unite

Separated

```
## # A tibble: 6 x 4
##   name    year   wins totals
##   <chr>   <chr> <int>  <int>
## 1 Alice   2018      5     12
## 2 Alice   2019      9     17
## 3 Bob     2018      9     24
## 4 Bob     2019      7     11
## 5 Charlie 2018      3     15
## 6 Charlie 2019      2     12
```

Now take a break!

Iteration

A common paradigm throughout programming is *iteration.*

The standard approach to this is through loops e.g. for, while etc.

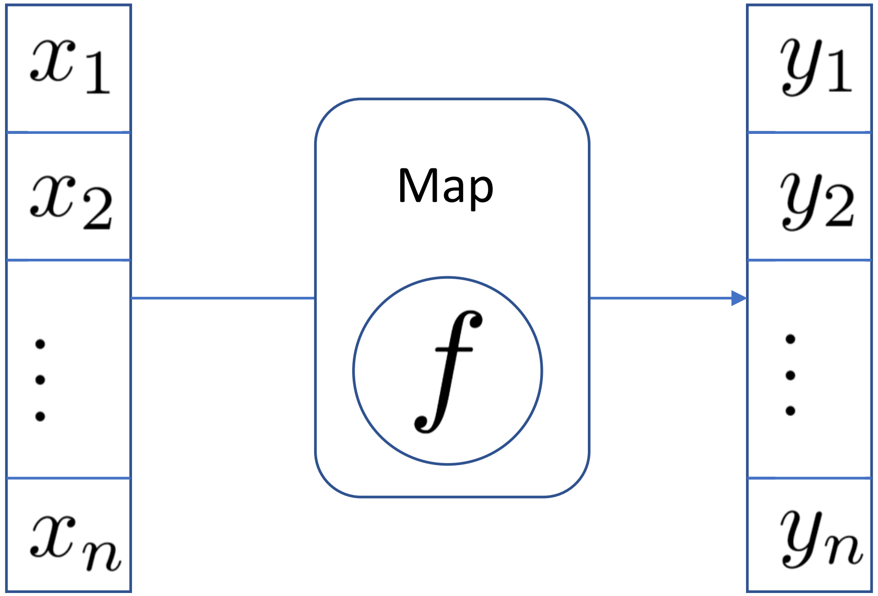
In R we prefer *vectorized* and map operations where possible:

- Historically, this led to efficiency improvements. However, this is often no longer the case.
- Map based approaches are typically more readable.

The map function

Suppose you have a function f which takes input x and outputs a value y:

The map function extends this to a vector or a list of values



The map function

```
is_div_2_3<-function(x){
  if(x%%2==0){
    return(TRUE)
  }else if(x%%3==0){
    return(TRUE)
  }else{
    return(FALSE)
  }
}
```

```
is_div_2_3(3)
```

```
## [1] TRUE
```

```
v<-c(1,2,3,4,5)
```

```
is_div_2_3(v)
```

```
## Warning in if (x%%2 == 0) {: the condition has length > 1 and only the first
## element will be used
```

```
## Warning in if (x%%3 == 0) {: the condition has length > 1 and only the first
## element will be used
```

```
## [1] FALSE
```

The map function

```
is_div_2_3<-function(x){
  if(x%%2==0){
    return(TRUE)
  }else if(x%%3==0){
    return(TRUE)
  }else{
    return(FALSE)
  }
}
```

```
v<-c(1,2,3,4,5)
```

```
map(v,is_div_2_3)
```

```
## [[1]]
## [1] FALSE
##
## [[2]]
## [1] TRUE
##
## [[3]]
## [1] TRUE
##
## [[4]]
## [1] TRUE
##
## [[5]]
## [1] FALSE
```

The map function

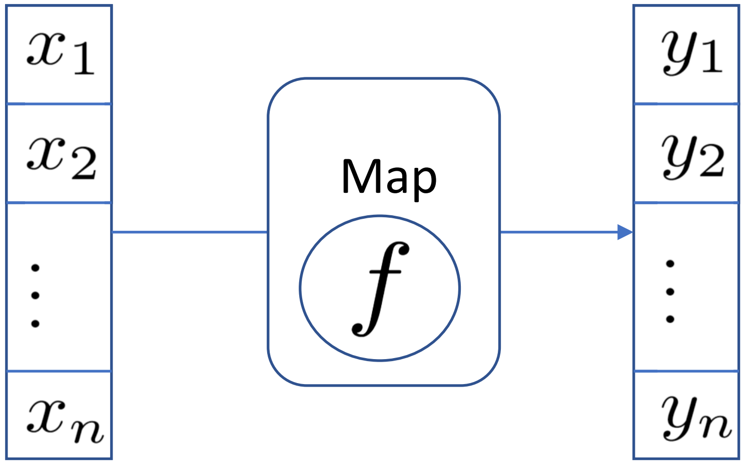
The **map()** function is taken from the is taken from the purrR library which is contained in the Tidyverse.

The output of **map()** is a list of function values.

The output of **map_lgl()** is a vector of Booleans.

The output of **map_dbl()** is a vector of doubles.

The output of **map_chr()** is a vector of strings.



The map function

```
is_div_2_3<-function(x){
  if(x%%2==0){
    return(TRUE)
  }else if(x%%3==0){
    return(TRUE)
  }else{
    return(FALSE)
  }
}
```

```
v<-c(1,2,3,4,5)
```

```
map(v,is_div_2_3)
```

```
## [[1]]
## [1] FALSE
##
## [[2]]
## [1] TRUE
##
## [[3]]
## [1] TRUE
##
## [[4]]
## [1] TRUE
##
## [[5]]
## [1] FALSE
```

The map function

The output of **map_lgl()** is a vector of Booleans.

```
is_div_2_3<-function(x){
  if(x%%2==0){
    return(TRUE)
  }else if(x%%3==0){
    return(TRUE)
  }else{
    return(FALSE)
  }
}
```

```
v<-c(1,2,3,4,5)
```

```
map_lgl(v,is_div_2_3)
```

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

The map function

The output of **map_dbl()** is a vector of doubles.

```
example_f<-function(x){
  if(is_div_2_3(x)){
    return(x)
  }else{
    return(0)
  }
}
```

```
v<-c(1,2,3,4,5)
```

```
map_dbl(v,example_f)
```

```
## [1] 0 2 3 4 0
```

The map function

The output of **map_chr()** is a vector of strings.

```
example_f<-function(x){
  if(is_div_2_3(x)){
    return(x)
  }else{
    return(0)
  }
}
```

```
library(english)

example_f_eng<-function(x){as.character(as.english(example_f(x)))}
```

```
v<-c(1,2,3,4,5)
```

```
map_chr(v,example_f_eng)
```

```
## [1] "zero"  "two"   "three" "four"  "zero"
```

Vectorization

```
is_div_2_3_vect<-function(x){return(x%%2==0|x%%3==0)}

is_div_2_3_vect(v)
```

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

```
example_f_vect<-function(x){return(x*is_div_2_3_vect(x))}

example_f_vect(v)
```

```
## [1] 0 2 3 4 0
```

```
example_f_eng_vect<-function(x){return(as.english(example_f_vect(v)))}

example_f_eng_vect(v)
```

```
## [1] zero  two   three four  zero
```

Nesting and unnesting

```
## # A tibble: 4 x 3
##   name  band    plays
##   <chr> <chr>   <chr>
## 1 Mick  Stones  <NA>
## 2 John  Beatles guitar
## 3 Paul  Beatles bass
## 4 Keith <NA>    guitar
```

Nest

Unnest

```
## # A tibble: 4 x 2
## # Groups:   name [4]
##   name  data
##   <chr> <list>
## 1 Mick  <tibble [1 x 2]>
## 2 John  <tibble [1 x 2]>
## 3 Paul  <tibble [1 x 2]>
## 4 Keith <tibble [1 x 2]>
```

Nesting and unnesting

```
musicians
```

```
## # A tibble: 4 x 3
##   name  band    plays
##   <chr> <chr>   <chr>
## 1 Mick  Stones  <NA>
## 2 John  Beatles guitar
## 3 Paul  Beatles bass
## 4 Keith <NA>    guitar
```

```
musicians_nested<-musicians%>%
  group_by(name)%>%
  nest()
```

```
musicians_nested
```

```
## # A tibble: 4 x 2
## # Groups:   name [4]
##   name  data
##   <chr> <list>
## 1 Mick  <tibble [1 x 2]>
## 2 John  <tibble [1 x 2]>
## 3 Paul  <tibble [1 x 2]>
## 4 Keith <tibble [1 x 2]>
```

Nesting and unnesting

```
musicians_nested
```

```
## # A tibble: 4 x 2
## # Groups:   name [4]
##   name  data
##   <chr> <list>
## 1 Mick  <tibble [1 x 2]>
## 2 John  <tibble [1 x 2]>
## 3 Paul  <tibble [1 x 2]>
## 4 Keith <tibble [1 x 2]>
```

```
musicians_nested%>%
  unnest(cols=data)
```

```
## # A tibble: 4 x 3
## # Groups:   name [4]
##   name  band    plays
##   <chr> <chr>   <chr>
## 1 Mick  Stones  <NA>
## 2 John  Beatles guitar
## 3 Paul  Beatles bass
## 4 Keith <NA>    guitar
```

Example: Finding variables of maximal correlation

Our goal is to create a function which:

1) Takes as input a data frame and a variable name
2) Computes the correlation with all other numeric variables
3) Returns the name of the variable with maximal absolute correlation, and the corresponding correlation.

```r
max_cor_var<-function(df,col_name){ # function to determine the variable with maximal correlation

  v_col<-df%>%select(all_of(col_name)) # extract variable based on col_name

  df_num<-df%>%
    select_if(is.numeric)%>%
    select(-all_of(col_name)) # select all numeric variables excluding col_name

  correlations<-unlist(map(df_num,function(x){cor(x,v_col,use="complete.obs")})) # compute correlations with all
 other numeric variables

  max_abs_cor_var<-names(which(abs(correlations)==max(abs(correlations)))) # extract the  variable name
  cor<-as.double(correlations[max_abs_cor_var]) # compute the correlation

  return(data.frame(var_name=max_abs_cor_var,cor=cor)) # return dataframe
}
```

Example: Finding variables of maximal correlation

```
max_cor_var<-function(df,col_name){ # function to determine the variable with maximal correlation

  v_col<-df%>%select(all_of(col_name)) # extract variable based on col_name

  df_num<-df%>%
    select_if(is.numeric)%>%
    select(-all_of(col_name)) # select all numeric variables excluding col_name

  correlations<-unlist(map(df_num,function(x){cor(x,v_col,use="complete.obs")})) # compute correlations with all
 other numeric variables

  max_abs_cor_var<-names(which(abs(correlations)==max(abs(correlations)))) # extract the  variable name
  cor<-as.double(correlations[max_abs_cor_var]) # compute the correlation

  return(data.frame(var_name=max_abs_cor_var,cor=cor)) # return dataframe
}
```

```
library(palmerpenguins)
```

```
penguins%>%
  max_cor_var("body_mass_g")
```

```
##              var_name       cor
## 1 flipper_length_mm 0.8712018
```

Example: Finding variables of maximal correlation

We can use the **nest()** and **unnest()** functions to compute the variable with maximal correlation.

```
penguins%>%
  group_by(species)%>%
  nest()%>%
  mutate(max_cor=map(data,~max_cor_var(.x,"body_mass_g")))%>%
  select(-data)%>%
  unnest(cols=max_cor)
```

```
## # A tibble: 3 x 3
## # Groups:   species [3]
##   species   var_name            cor
##   <fct>     <chr>             <dbl>
## 1 Adelie    bill_depth_mm     0.576
## 2 Gentoo    bill_depth_mm     0.719
## 3 Chinstrap flipper_length_mm 0.642
```

Missing data

Missing data is remarkably common in practical Data Science applications:

“One of the ironies of working with Big Data is that missing data play an ever more significant role, and often present serious difficulties for analysis.”

Zhu, Wang and Samworth, 2019.

```
##   Species Sex Wing Weight Culmen Hallux Tail Tarsus WingPitFat KeelFat
## 1      RT      385    920   25.7   30.1  219     NA         NA      NA
## 2      RT      376    930     NA     NA  221     NA         NA      NA
## 3      RT      381    990   26.7   31.3  235     NA         NA      NA
## 4      CH    F 265    470   18.7   23.5  220     NA         NA      NA
## 5      SS    F 205    170   12.5   14.3  157     NA         NA      NA
## 6      RT      412   1090   28.5   32.2  230     NA         NA      NA
## 7      RT      370    960   25.3   30.1  212     NA         NA      NA
## 8      RT      375    855   27.2   30.0  243     NA         NA      NA
```

Missing data

1. Explicit missing data:

   The value of an individual variable replaced with “NA” (not available).

1. Implicit missing data:

   Entire rows missing from a data frame.

```
##      name year wins losses
## 1   Alice 2018    5      7
## 2 Charlie 2018    9     NA
## 3   Alice 2019    3     12
## 4     Bob 2019    9      8
## 5 Charlie 2019    7      4
```

Making missing data explicit

```
w_l
```

```
##      name year wins losses
## 1   Alice 2018    5      7
## 2 Charlie 2018    9     NA
## 3   Alice 2019    3     12
## 4     Bob 2019    9      8
## 5 Charlie 2019    7      4
```

```
w_l%>%
  complete(name,year)
```

```
## # A tibble: 6 x 4
##   name     year  wins losses
##   <chr>   <dbl> <dbl>  <dbl>
## 1 Alice    2018     5      7
## 2 Alice    2019     3     12
## 3 Bob      2018    NA     NA
## 4 Bob      2019     9      8
## 5 Charlie  2018     9     NA
## 6 Charlie  2019     7      4
```

Complete case analysis

```
w_l
```

```
##      name year wins losses
## 1   Alice 2018    5      7
## 2 Charlie 2018    9     NA
## 3   Alice 2019    3     12
## 4     Bob 2019    9      8
## 5 Charlie 2019    7      4
```

```
complete.cases(w_l)
```

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

```
w_l%>%
  filter(complete.cases(.))
```

```
##      name year wins losses
## 1   Alice 2018    5      7
## 2   Alice 2019    3     12
## 3     Bob 2019    9      8
## 4 Charlie 2019    7      4
```

Imputation by mean

```r
impute_by_mean<-function(x){

  mu<-mean(x,na.rm=1) # first compute the mean of x

  impute_f<-function(z){ # coordinate-wise imputation
    if(is.na(z)){
      return(mu) # if z is na replace with mean
    }else{
      return(z) # otherwise leave in place
    }
  }
  return(map_dbl(x,impute_f)) # apply the map function to impute across vector
}
```

```r
x<-c(1,2,NA,4)
impute_by_mean(x)
```

```
## [1] 1.000000 2.000000 2.333333 4.000000
```

Imputation by mean

```
w_l_na
```

```
## # A tibble: 6 x 4
##   name     year  wins losses
##   <chr>   <dbl> <dbl>  <dbl>
## 1 Alice    2018     5      7
## 2 Alice    2019     3     12
## 3 Bob      2018    NA     NA
## 4 Bob      2019     9      8
## 5 Charlie  2018     9     NA
## 6 Charlie  2019     7      4
```

```
w_l_na%>%
  mutate(wins=impute_by_mean(wins),losses=impute_by_mean(losses))
```

```
## # A tibble: 6 x 4
##   name     year  wins losses
##   <chr>   <dbl> <dbl>  <dbl>
## 1 Alice    2018   5     7
## 2 Alice    2019   3    12
## 3 Bob      2018   6.6   7.75
## 4 Bob      2019   9     8
## 5 Charlie  2018   9     7.75
## 6 Charlie  2019   7     4
```

What have we covered?

- We introduced the formal concept of tidy data.
- We saw how to reshape data with the pivot functions.
- We looked at the unite and separate functions and the nest and unnest functions.
- We investigated the map function for iteration within the tidyverse R.
- We also look at some basic methods for handling missing data.

University of BRISTOL

Thanks for listening!

Henry W J Reeve

henry.reeve@bristol.ac.uk

Unit: EMATM0061

Statistical Computing & Empirical Methods