Data wrangling

Using dplyr to transform your data

Statistical Computing and Empirical Methods
Unit EMATM0061, Data Science MSc

Rihuan Ke

rihuan.ke@bristol.ac.uk

Teaching Block 1, 2024

University of BRISTOL

What we will cover in this lecture

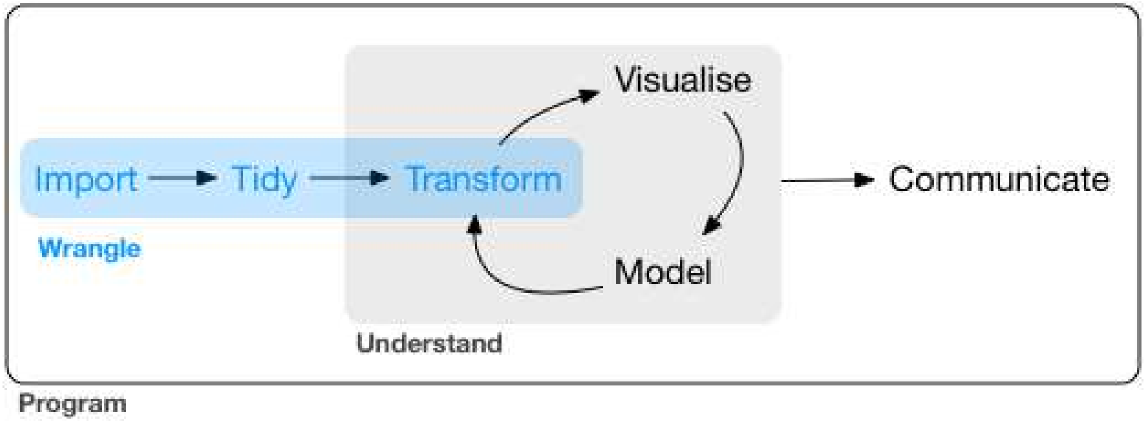
- We will introduce concepts in basic data wrangling operations
  - Select, filter, mutate, arrange, summarize, …

- We will learn how to perform data wrangling operations using tools in the programming language R
  - The package dplyr

What is data wrangling

Data wrangling: the process of transforming data from one form to another in preparation for another downstream task (e.g., visualisation, modelling)

Transforming your data with basic data wrangling operations: selecting, filtering, mutating, arranging, summarizing, joining…



source: r4ds.had.co.nz

Learning data wrangling with examples in R

We will learn data wrangling with examples in R

We will do the examples with two important R packages

1. *The dplyr package*
   - An R package designed for data wrangling
   - Effective data wrangling APIs, such as
     - *select(), filter(), mutate(), …*

2. *The tidyverse package*
   - A collection of R packages that are designed for data science
   - Including
     - *ggplot2*: a package for visualisation
     - *tidyr*: a package for tidying data
     - *dplyr*
     - *purrr*: functional programming

Install and load the packages:

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

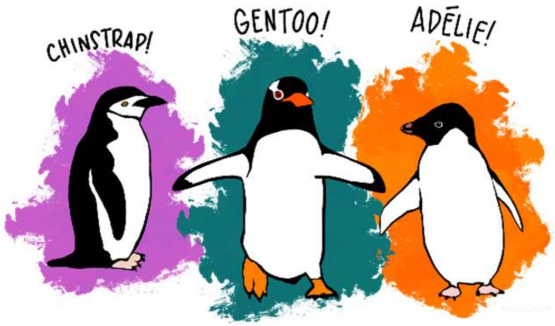
The dplyr package is included in the tidyverse package (so it is loaded when tidyverse is loaded)

Case study: the Palmer penguins data set

Examples will be demonstrated using the *Palmer penguins data set*, which was introduced by Alison Hill, Allison Horst, Kristen Gorman

To use the Palmer penguins data set:

```r
install.packages("palmerpenguins")
library(palmerpenguins)
```



This is what the *penguins* data looks like:

```r
head(penguins)
```

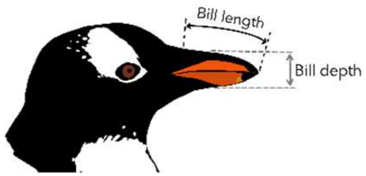
```
## # A tibble: 6 × 8
##   species island    bill_length_mm bill_depth_mm flipper_l…¹ body_…² sex    year
##   <fct>   <fct>              <dbl>         <dbl>       <int>   <int> <fct> <int>
## 1 Adelie  Torgersen           39.1          18.7         181    3750 male   2007
## 2 Adelie  Torgersen           39.5          17.4         186    3800 fema…  2007
## 3 Adelie  Torgersen           40.3          18           195    3250 fema…  2007
## 4 Adelie  Torgersen           NA            NA            NA      NA <NA>   2007
## 5 Adelie  Torgersen           36.7          19.3         193    3450 fema…  2007
## 6 Adelie  Torgersen           39.3          20.6         190    3650 male   2007
## # … with abbreviated variable names ¹flipper_length_mm, ²body_mass_g
```

In R, data sets are often stored as data frames

Tabular data

Penguins is an example of a tabular data set represented by an R data frame.

```
## # A tibble: 6 × 8
##   species island    bill_length_mm bill_depth_mm flipper_l…¹ body_…² sex    year
##   <fct>   <fct>              <dbl>         <dbl>       <int>   <int> <fct> <int>
## 1 Adelie  Torgersen           39.1          18.7         181    3750 male   2007
## 2 Adelie  Torgersen           39.5          17.4         186    3800 fema…  2007
## 3 Adelie  Torgersen           40.3          18           195    3250 fema…  2007
## 4 Adelie  Torgersen           NA            NA            NA      NA <NA>   2007
## 5 Adelie  Torgersen           36.7          19.3         193    3450 fema…  2007
## 6 Adelie  Torgersen           39.3          20.6         190    3650 male   2007
## # … with abbreviated variable names ¹flipper_length_mm, ²body_mass_g
```



rows

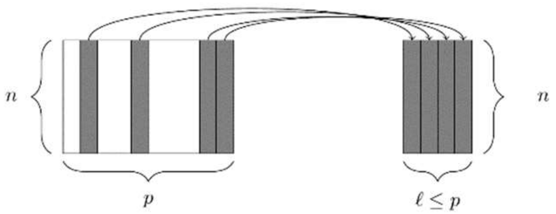
Each row corresponds to an instance of a specific type of thing, in this case, an individual penguin. Known as examples, observations or cases.

columns

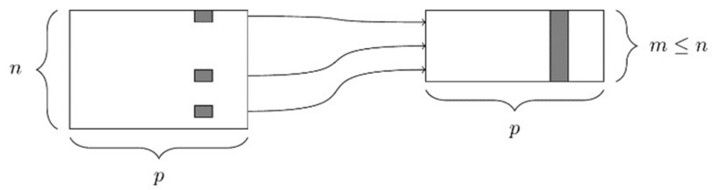
Each column (also called a variable) corresponds to a property or quality of the individual examples. Known as features, or variables.

In the Penguins data set, we have 8 columns, corresponding to different properties of a penguin: *"species"*, *"island"*, *"bill_length_mm"*, *"bill_depth_mm"*, *"flipper_length_mm"*, *"body_mass_g"*, *"sex"*, *"year"*

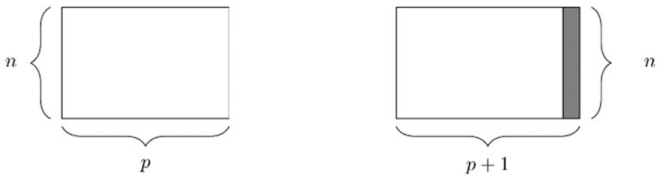
Data wrangling operations



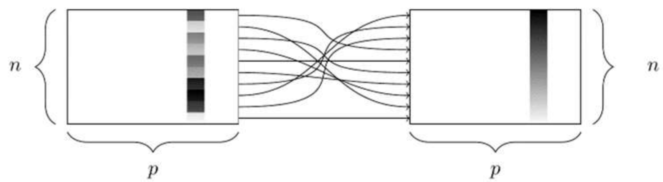
select



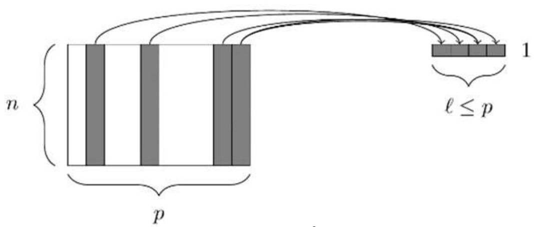
filter



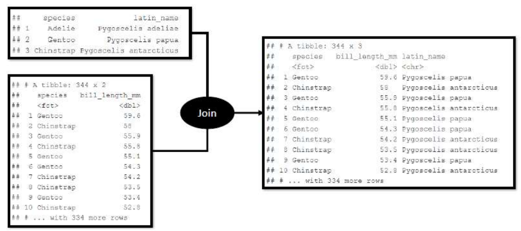
Create new columns



sort

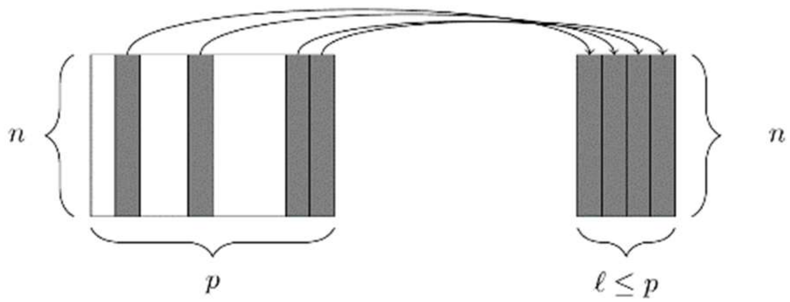


summarize



joint

1. Selecting columns



Selecting a subset of columns and generating a new dataset (with fewer columns)

In R, this can be done with the select() function (from the dplyr package), e.g.,

```
penguinsv2 <- select(penguins, species, bill_length_mm, body_mass_g, flipper_length_mm )
print(penguinsv2)
```

```
## # A tibble: 344 x 4
##    species bill_length_mm body_mass_g flipper_length_mm
##    <fct>            <dbl>       <int>             <int>
##  1 Adelie            39.1        3750               181
##  2 Adelie            39.5        3800               186
##  3 Adelie            40.3        3250               195
##  4 Adelie            NA            NA                NA
##  5 Adelie            36.7        3450               193
##  6 Adelie            39.3        3650               190
##  7 Adelie            38.9        3625               181
##  8 Adelie            39.2        4675               195
##  9 Adelie            34.1        3475               193
## 10 Adelie            42          4250               190
## # … with 334 more rows
```

The result *penguinsv2* is a new data frame (with 4 columns), which we will use frequently in the following examples

1. Selecting columns

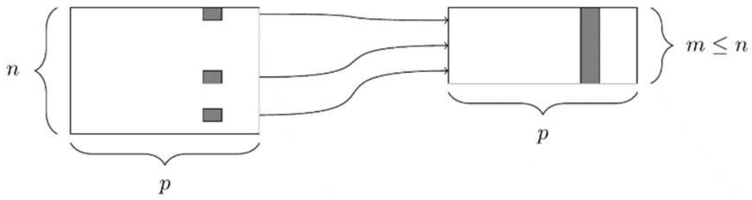
The select function also allows us to remove several columns (invert selection) using the symbol '-' , e.g.,

```
select(penguins, -species, -bill_length_mm, -body_mass_g)
```

```
## # A tibble: 344 x 5
##    island    bill_depth_mm flipper_length_mm sex     year
##    <fct>             <dbl>             <int> <fct>  <int>
##  1 Torgersen          18.7               181 male    2007
##  2 Torgersen          17.4               186 female  2007
##  3 Torgersen          18                 195 female  2007
##  4 Torgersen          NA                  NA <NA>    2007
##  5 Torgersen          19.3               193 female  2007
##  6 Torgersen          20.6               190 male    2007
##  7 Torgersen          17.8               181 female  2007
##  8 Torgersen          19.6               195 male    2007
##  9 Torgersen          18.1               193 <NA>    2007
## 10 Torgersen          20.2               190 <NA>    2007
## # … with 334 more rows
```

So we have 5 columns (after removing the three columns)

2. Filtering rows

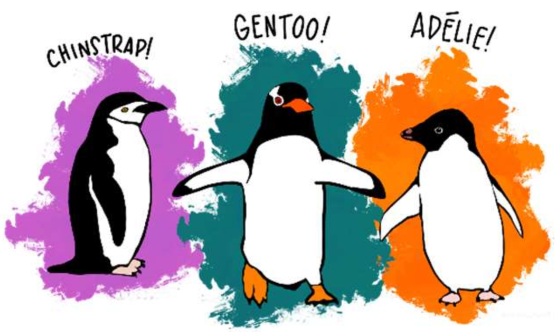


Extracting a subset of rows (while the columns are unchanged)

In R, this can be done with the filter() function (from the dplyr package), e.g.,

```
filter(penguinsv2, species=='Gentoo')
```

```
## # A tibble: 124 × 4
##    species bill_length_mm body_mass_g flipper_length_mm
##    <fct>            <dbl>       <int>             <int>
##  1 Gentoo            46.1        4500               211
##  2 Gentoo            50          5700               230
##  3 Gentoo            48.7        4450               210
##  4 Gentoo            50          5700               218
##  5 Gentoo            47.6        5400               215
##  6 Gentoo            46.5        4550               210
##  7 Gentoo            45.4        4800               211
##  8 Gentoo            46.7        5200               219
##  9 Gentoo            43.3        4400               209
## 10 Gentoo            46.8        5150               215
## # … with 114 more rows
```



So we get rows associated with penguins that are of ‘Gentoo’ species

2. Filtering rows

We can select the rows that satisfy multiple conditions (using the expression '&')

For example, to select penguins that are of the Gentoo species and has body mass bigger than 5kg:

```
filter(penguinsv2, species=='Gentoo' & body_mass_g>5000)
```

```
## # A tibble: 61 x 4
##    species bill_length_mm body_mass_g flipper_length_mm
##    <fct>            <dbl>       <int>             <int>
##  1 Gentoo            50          5700               230
##  2 Gentoo            50          5700               218
##  3 Gentoo            47.6        5400               215
##  4 Gentoo            46.7        5200               219
##  5 Gentoo            46.8        5150               215
##  6 Gentoo            49          5550               216
##  7 Gentoo            48.4        5850               213
##  8 Gentoo            49.3        5850               217
##  9 Gentoo            49.2        6300               221
## 10 Gentoo            48.7        5350               222
## # … with 51 more rows
```

Combining filter & select functions

The functions *select* and *filter* can be used together (to select a subset of columns and a subset of rows)

```
select(filter(penguinsv2, species=='Gentoo'), species, bill_length_mm, body_mass_g)
```

```
## # A tibble: 124 x 3
##    species bill_length_mm body_mass_g
##    <fct>            <dbl>       <int>
##  1 Gentoo            46.1        4500
##  2 Gentoo            50          5700
##  3 Gentoo            48.7        4450
##  4 Gentoo            50          5700
##  5 Gentoo            47.6        5400
##  6 Gentoo            46.5        4550
##  7 Gentoo            45.4        4800
##  8 Gentoo            46.7        5200
##  9 Gentoo            43.3        4400
## 10 Gentoo            46.8        5150
## # … with 114 more rows
```

So we get only three columns & rows that are associated with “Gentoo” species

Simplifying codes with the pipe operator

We can also chain multiple operations with the pipe operator %>%

The following statements are equivalent:

```
select(filter(penguinsv2, species=='Gentoo'), species, bill_length_mm, body_mass_g)
```

```
penguinsv2 %>%
   filter(species=='Gentoo') %>%
   select(species, bill_length_mm, body_mass_g)
```

The pipe operator %>% allows arguments to be implicitly passed as objects to the function after the pipe.

```
f <- function(a,b) {return (a^2 + b) }
print(f(3,1))
```

```
## [1] 10
```

```
print( 3 %>% f(1) )
```

```
## [1] 10
```

Simplifying codes with the pipe operator

To chain multiple operations (e.g., f1, f2, f3), we have

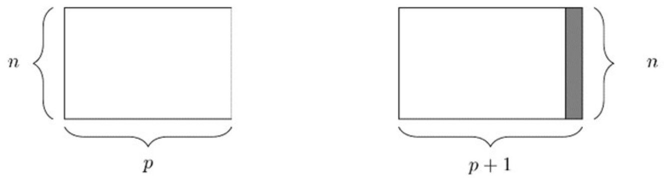
x %>% f1(a) %>% f2(b) %>% f3(c) means **f3(f2(f1(x, a), b), c)**

The pipe operator %>% is taken from the *magrittr* package which is also part of the *tidyverse*

The *magrittr* package was developed by Stefan Milton Bache and Hadley Wickham.

3. Creating and renaming columns

The aim is to create a new column as a function of existing columns.



In R, this can be done with the mutate() function (from the dplyr package), e.g.,

```
penguinsv2 %>%
  mutate(flipper_bill_ratio=flipper_length_mm/bill_length_mm)
```

```
## # A tibble: 344 x 5
##    species bill_length_mm body_mass_g flipper_length_mm flipper_bill_ratio
##    <fct>            <dbl>       <int>             <int>              <dbl>
##  1 Adelie            39.1        3750               181               4.63
##  2 Adelie            39.5        3800               186               4.71
##  3 Adelie            40.3        3250               195               4.84
##  4 Adelie            NA            NA                NA              NA
##  5 Adelie            36.7        3450               193               5.26
##  6 Adelie            39.3        3650               190               4.83
##  7 Adelie            38.9        3625               181               4.65
##  8 Adelie            39.2        4675               195               4.97
##  9 Adelie            34.1        3475               193               5.66
## 10 Adelie            42          4250               190               4.52
## # … with 334 more rows
```

A new column flipper_bill_ratio has been created

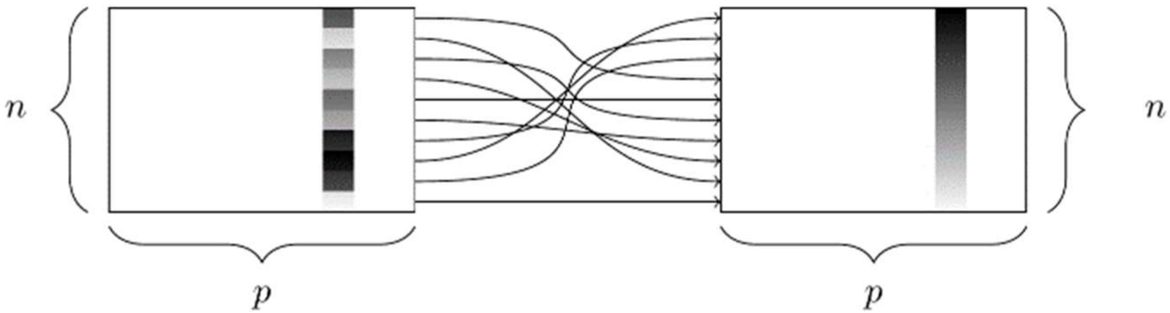
3. Creating and renaming columns

To rename an existing column, we can use the rename () function, e.g.,

```
penguinsv2 %>% rename(f_l_m = flipper_length_mm)
```

```
## # A tibble: 344 x 4
##    species bill_length_mm body_mass_g f_l_m
##    <fct>            <dbl>       <int> <int>
##  1 Adelie            39.1        3750   181
##  2 Adelie            39.5        3800   186
##  3 Adelie            40.3        3250   195
##  4 Adelie            NA            NA    NA
##  5 Adelie            36.7        3450   193
##  6 Adelie            39.3        3650   190
##  7 Adelie            38.9        3625   181
##  8 Adelie            39.2        4675   195
##  9 Adelie            34.1        3475   193
## 10 Adelie            42          4250   190
## # … with 334 more rows
```

4. Sorting the rows



Sorting the rows of a data frame according to the values of a column

In R, this can be done with the arrange() function, e.g.,

```
penguinsv2 %>% arrange(bill_length_mm)
```

```
## # A tibble: 344 x 4
##    species bill_length_mm body_mass_g flipper_length_mm
##    <fct>            <dbl>       <int>             <int>
##  1 Adelie            32.1        3050               188
##  2 Adelie            33.1        2900               178
##  3 Adelie            33.5        3600               190
##  4 Adelie            34          3400               185
##  5 Adelie            34.1        3475               193
##  6 Adelie            34.4        3325               184
##  7 Adelie            34.5        2900               187
##  8 Adelie            34.6        4400               198
##  9 Adelie            34.6        3200               189
## 10 Adelie            35          3450               190
## # … with 334 more rows
```

Now the rows are sorted according to bill_length_mm (in ascending order)

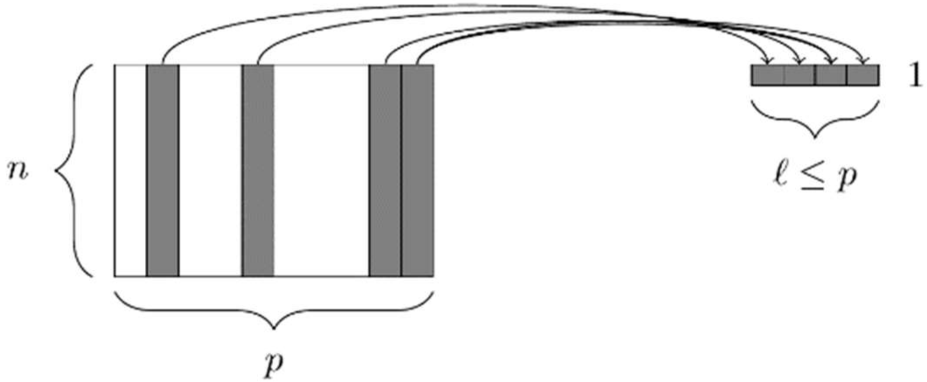
4. Sorting the rows

We can also sort in descending order, with desc()

```
penguinsv2 %>% arrange(desc(bill_length_mm))
```

```
## # A tibble: 344 x 4
##    species   bill_length_mm body_mass_g flipper_length_mm
##    <fct>              <dbl>       <int>             <int>
##  1 Gentoo              59.6        6050               230
##  2 Chinstrap           58          3700               181
##  3 Gentoo              55.9        5600               228
##  4 Chinstrap           55.8        4000               207
##  5 Gentoo              55.1        5850               230
##  6 Gentoo              54.3        5650               231
##  7 Chinstrap           54.2        4300               201
##  8 Chinstrap           53.5        4500               205
##  9 Gentoo              53.4        5500               219
## 10 Chinstrap           52.8        4550               205
## # … with 334 more rows
```

5. Summarizing data



Summarizing a data frame into just one value or a vector (e.g., compute the mean, median, sum, standard deviation, … of a column)

In R, this can be done with the summarize() function, e.g.,

```
penguinsv2 %>%
  summarize(num_rows=n(), avg_weight_kg=mean(body_mass_g/1000, na.rm=TRUE), avg_flipper_bill_ratio=mean(flipper_length_mm/bi
ll_length_mm, na.rm=TRUE))
```

```
## # A tibble: 1 x 3
##   num_rows avg_weight_kg avg_flipper_bill_ratio
##      <int>         <dbl>                  <dbl>
## 1      344          4.20                   4.62
```

Here we have extracted three statistics including the *number of rows, average weights, and average flipper bill ratio* (which are contained in the three columns of the output)

Group the rows when summarizing

We can use the function group_by() to group the rows of the data frames according to some given criteria, e.g. species

```
penguinsv2 %>%
  group_by(species)
```

```
## # A tibble: 344 x 4
## # Groups:   species [3]
##    species bill_length_mm body_mass_g flipper_length_mm
##    <fct>            <dbl>       <int>             <int>
##  1 Adelie            39.1        3750               181
##  2 Adelie            39.5        3800               186
##  3 Adelie            40.3        3250               195
##  4 Adelie            NA            NA                NA
##  5 Adelie            36.7        3450               193
##  6 Adelie            39.3        3650               190
##  7 Adelie            38.9        3625               181
##  8 Adelie            39.2        4675               195
##  9 Adelie            34.1        3475               193
## 10 Adelie            42          4250               190
## # … with 334 more rows
```

Summarize by group

Group and then summarize:

```
penguinsv2 %>%
  group_by(species) %>%
  summarize(num_rows=n(), avg_weight_kg=mean(body_mass_g/1000, na.rm=TRUE), avg_flipper_bill_ratio=mean(flipper_length_mm/bi
ll_length_mm, na.rm=TRUE))
```

```
## # A tibble: 3 x 4
##   species   num_rows avg_weight_kg avg_flipper_bill_ratio
##   <fct>        <int>         <dbl>                  <dbl>
## 1 Adelie         152          3.70                   4.92
## 2 Chinstrap       68          3.73                   4.02
## 3 Gentoo         124          5.08                   4.58
```

Now we have extracted the three statistics for individual groups (instead of the whole data frame)

Column-wise operations with across()

Suppose that we want to compute the number of NA (not available) values in each column, which can be done via

```
Num_NAs <- penguinsv2 %>% summarize(species=sum(is.na(species)), bill_length_mm=sum(is.na(bill_length_mm)), body_mass_g=sum
(is.na(body_mass_g)), flipper_length_mm=sum(is.na(flipper_length_mm)))
print(Num_NAs)
```

```
## # A tibble: 1 × 4
##   species bill_length_mm body_mass_g flipper_length_mm
##     <int>          <int>       <int>             <int>
## 1       0              2           2                 2
```

Use across to perform column-wise operations (for all columns), without copying and pasting the same code (e.g., sum(is.na(species)), …)

```
Num_NAs <- penguinsv2 %>% summarize(across(everything(), ~sum(is.na(.x))))
print(Num_NAs)
```

```
## # A tibble: 1 × 4
##   species bill_length_mm body_mass_g flipper_length_mm
##     <int>          <int>       <int>             <int>
## 1       0              2           2                 2
```

Here, **~sum(is.na(.x))** is equivalent to **function(x){(sum(is.na(x)))}**

Column-wise operations on a subset of columns

We can combine across () and where() to perform column-wise operations for a subset of columns (for example, that is of numeric type)

```
penguinsv2 %>% summarize(across(where(is.numeric), ~mean(.x, na.rm=TRUE)))
```

```
## # A tibble: 1 × 3
##   bill_length_mm body_mass_g flipper_length_mm
##            <dbl>       <dbl>             <dbl>
## 1           43.9       4202.              201.
```

Column-wise summarizing by groups

We can combine the summarize, group_by and across functions to perform *Column-wise summarizing by groups*

```
penguinsv2 %>%
  select(-bill_length_mm) %>%
  group_by(species) %>%
  summarize(across(where(is.numeric), ~mean(.x, na.rm=TRUE)))
```

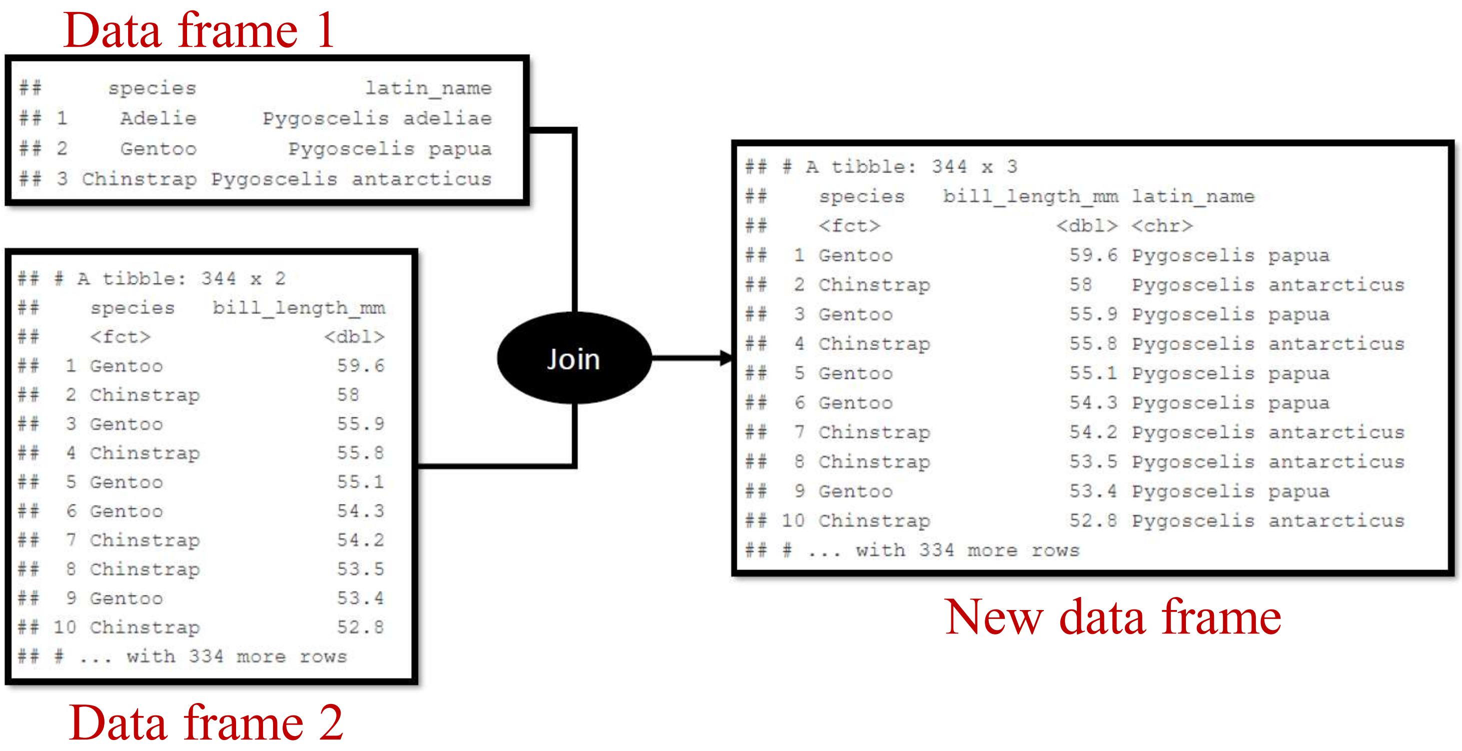
```
## # A tibble: 3 x 3
##   species   body_mass_g flipper_length_mm
##   <fct>           <dbl>             <dbl>
## 1 Adelie          3701.              190.
## 2 Chinstrap       3733.              196.
## 3 Gentoo          5076.              217.
```

Here, **~mean(.x, na.rm=TRUE)** is equivalent to **function(x){(mean(x, na.rm=TRUE))}**

Here, we compute the mean of a column which is of numeric type, for each species of penguins

6. Joining multiple data frames

Combining multiple data frames



In R, this can be done with the join functions (a part of the dplyr package)

Example

First, create data frame 1: a data frame of bill lengths and species.

```
penguin_bill_lengths_df <- penguinsv2 %>%
  arrange(desc(bill_length_mm)) %>%
  select(species, bill_length_mm)
penguin_bill_lengths_df
```

```
## # A tibble: 344 × 2
##    species   bill_length_mm
##    <fct>              <dbl>
##  1 Gentoo              59.6
##  2 Chinstrap           58
##  3 Gentoo              55.9
##  4 Chinstrap           55.8
##  5 Gentoo              55.1
##  6 Gentoo              54.3
##  7 Chinstrap           54.2
##  8 Chinstrap           53.5
##  9 Gentoo              53.4
## 10 Chinstrap           52.8
## # … with 334 more rows
```

Here we have used the functions arrange and select that were introduced before.

Example

Second, create data frame 2: a data frame of Latin species names

```
species <- unique(penguinsv2$species)
latin_name <- c('Pygoscelis adeliae', 'Pygoscelis papua', 'Pygoscelis antarcticus')
latin_name_df <- data.frame( species, latin_name  )
print(latin_name_df)
```

```
##     species             latin_name
## 1    Adelie     Pygoscelis adeliae
## 2    Gentoo       Pygoscelis papua
## 3 Chinstrap Pygoscelis antarcticus
```

Example

Data frame 1

```
## # A tibble: 344 × 2
##    species   bill_length_mm
##    <fct>              <dbl>
##  1 Gentoo              59.6
##  2 Chinstrap           58
##  3 Gentoo              55.9
##  4 Chinstrap           55.8
##  5 Gentoo              55.1
##  6 Gentoo              54.3
##  7 Chinstrap           54.2
##  8 Chinstrap           53.5
##  9 Gentoo              53.4
## 10 Chinstrap           52.8
## # … with 334 more rows
```

+

Data frame 2

```
##     species             latin_name
## 1    Adelie      Pygoscelis adeliae
## 2    Gentoo        Pygoscelis papua
## 3 Chinstrap Pygoscelis antarcticus
```

Finally, we can combine these two data frames with a join function.

```
penguin_bill_lengths_df %>% inner_join(latin_name_df)
```

```
## # A tibble: 344 × 3
##    species   bill_length_mm latin_name
##    <fct>              <dbl> <chr>
##  1 Gentoo              59.6 Pygoscelis papua
##  2 Chinstrap           58   Pygoscelis antarcticus
##  3 Gentoo              55.9 Pygoscelis papua
##  4 Chinstrap           55.8 Pygoscelis antarcticus
##  5 Gentoo              55.1 Pygoscelis papua
##  6 Gentoo              54.3 Pygoscelis papua
##  7 Chinstrap           54.2 Pygoscelis antarcticus
##  8 Chinstrap           53.5 Pygoscelis antarcticus
##  9 Gentoo              53.4 Pygoscelis papua
## 10 Chinstrap           52.8 Pygoscelis antarcticus
## # … with 334 more rows
```

The rows from the two data frames are merged, by matching the common columns (which is species here)

Here we have used the function *inner_join()*, which is a type of join function. There are other types of join functions available

Types of join functions

What happens when the set of values on the common column is not the same for both tables? For example:

```
print(band_members)
```

```
## # A tibble: 3 × 2
##   name  band
##   <chr> <chr>
## 1 Mick  Stones
## 2 John  Beatles
## 3 Paul  Beatles
```

+

```
print(band_instruments)
```

```
## # A tibble: 3 × 2
##   name  plays
##   <chr> <chr>
## 1 John  guitar
## 2 Paul  bass
## 3 Keith guitar
```

= ?

band_members and **band_instruments** are two toy datasets given by the dplyr package

“Mick” only appears in “band_members ” and “Keith” only appears in band_instruments

Let’s rename the two data frames as x and y, respectively.

```
x = band_members
y = band_instruments
```

There are four basic join functions, each of which deals with missing rows differently.

Types of join functions 1: Inner join

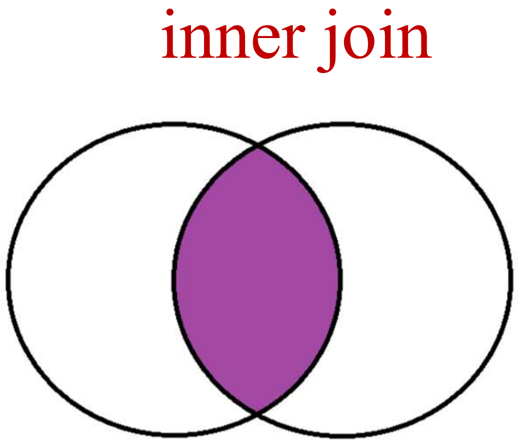
An inner join means only rows with matching keys in both x and y are included in the result.

```
print(x)
```

```
## # A tibble: 3 × 2
##   name  band
##   <chr> <chr>
## 1 Mick  Stones
## 2 John  Beatles
## 3 Paul  Beatles
```

```
print(y)
```

```
## # A tibble: 3 × 2
##   name  plays
##   <chr> <chr>
## 1 John  guitar
## 2 Paul  bass
## 3 Keith guitar
```




```
inner_join(x, y)
```

```
## # A tibble: 2 × 3
##   name  band    plays
##   <chr> <chr>   <chr>
## 1 John  Beatles guitar
## 2 Paul  Beatles bass
```

Neither Mick nor Keith is included

Types of join functions 2: Outer join

An outer join (also called a full join) means to include all rows in x with matching columns in y, then the rows of y that don't match x.

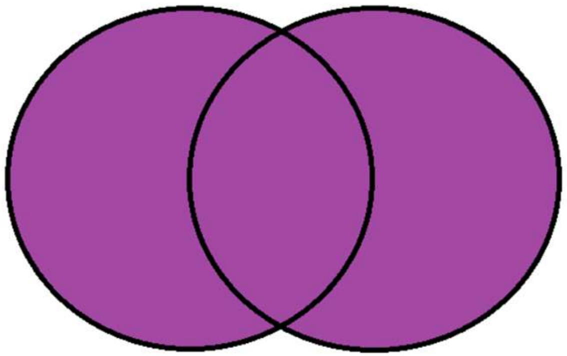
```
print(x)
```

```
## # A tibble: 3 × 2
##   name  band
##   <chr> <chr>
## 1 Mick  Stones
## 2 John  Beatles
## 3 Paul  Beatles
```

```
print(y)
```

```
## # A tibble: 3 × 2
##   name  plays
##   <chr> <chr>
## 1 John  guitar
## 2 Paul  bass
## 3 Keith guitar
```

Outer join



```
full_join(x, y)
```

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

Both Mick and Keith are included

Types of join functions 3: left join

A left join means to include all rows in x, and add matching columns from y.

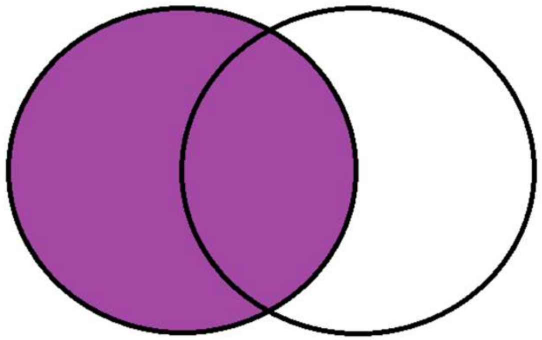
```
print(x)
```

```
## # A tibble: 3 × 2
##   name  band
##   <chr> <chr>
## 1 Mick  Stones
## 2 John  Beatles
## 3 Paul  Beatles
```

```
print(y)
```

```
## # A tibble: 3 × 2
##   name  plays
##   <chr> <chr>
## 1 John  guitar
## 2 Paul  bass
## 3 Keith guitar
```

left join



```
left_join(x, y)
```

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

Mick is included, but Keith is not

Types of join functions 4: right join

A right join means to include all rows in y, and add matching columns from x.

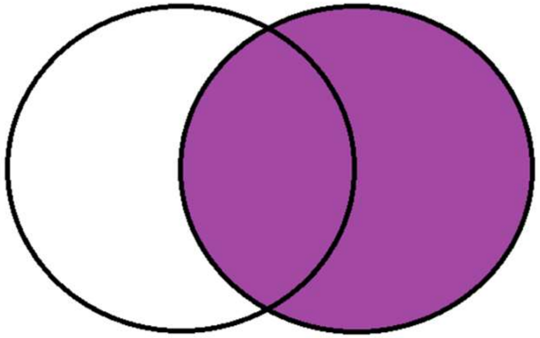
```
print(x)
```

```
## # A tibble: 3 × 2
##   name  band
##   <chr> <chr>
## 1 Mick  Stones
## 2 John  Beatles
## 3 Paul  Beatles
```

```
print(y)
```

```
## # A tibble: 3 × 2
##   name  plays
##   <chr> <chr>
## 1 John  guitar
## 2 Paul  bass
## 3 Keith guitar
```

right join



```
right_join(x, y)
```

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

Keith is included, but Mick is not

What we have covered today

- We introduced basic data wrangling operations, including
  - selecting, filtering, creating and renaming columns, sorting, Summarizing, and joining multiples data frames
- We introduced the *tidyverse* and *dplyr*, basic R packages for data science
- We learned how to perform data wrangling operations using examples with R
- We explored examples with the pip operator %>%
- We explained how to use group_by, and cross functions to obtain summary data
- We discussed different types of join functions (inner join, full join, left join, and right join)

Try the examples yourself?

The illustration, codes, and examples are included in the R Markdown file **LectureDataWrangling.Rmd** which can be downloaded via the course webpage.

Thanks for listening!

Dr Rihuan Ke
rihuan.ke@bristol.ac.uk

Statistical Computing and Empirical Methods
Unit EMATM0061, MSc Data Science