Data wrangling

Using dplyr to transform your data I.

Henry W J Reeve

henry.reeve@bristol.ac.uk

Statistical Computing & Empirical Methods

What will we cover today?

- We will explore some foundational concepts of tabular data

- We will then introduce and explore the basics of data wrangling using the dplyr library.

    - Extracting subsets
    - Adding new columns
    - Rearranging your rows
    - Summarizing your data
    - Fusing together data frames.

The Palmer penguins data set

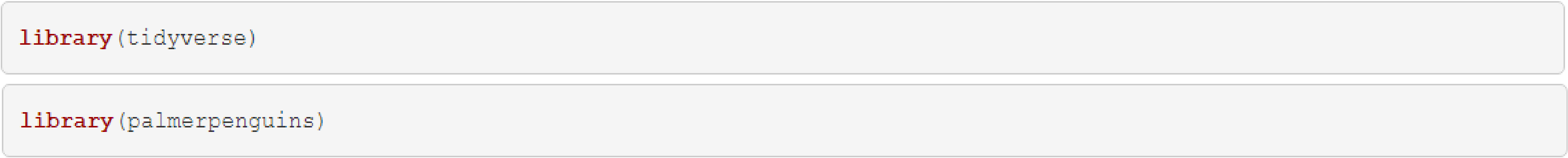
- We will also make use of the Palmer penguin data set.

Adélie Penguin
Gentoo Penguin
Chinstrap Penguin

- Introduced by Alison Hill, Allison Horst, Kristen Gorman.

The Palmer penguins data set

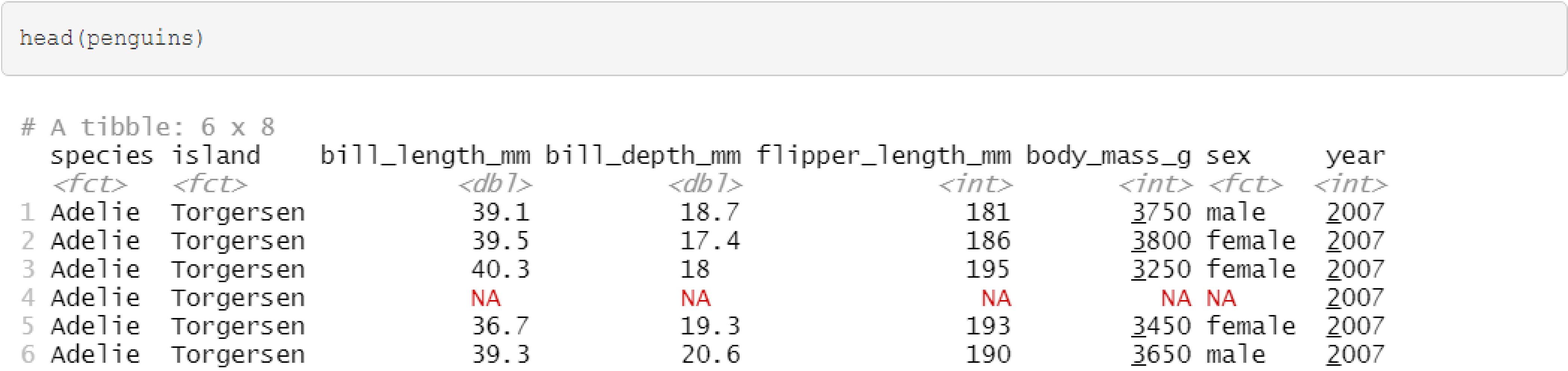
- Load the Tidy verse + the Palmer penguins data set.



```
library(tidyverse)
```

```
library(palmerpenguins)
```

- We can take a look at the data set by using the head function.

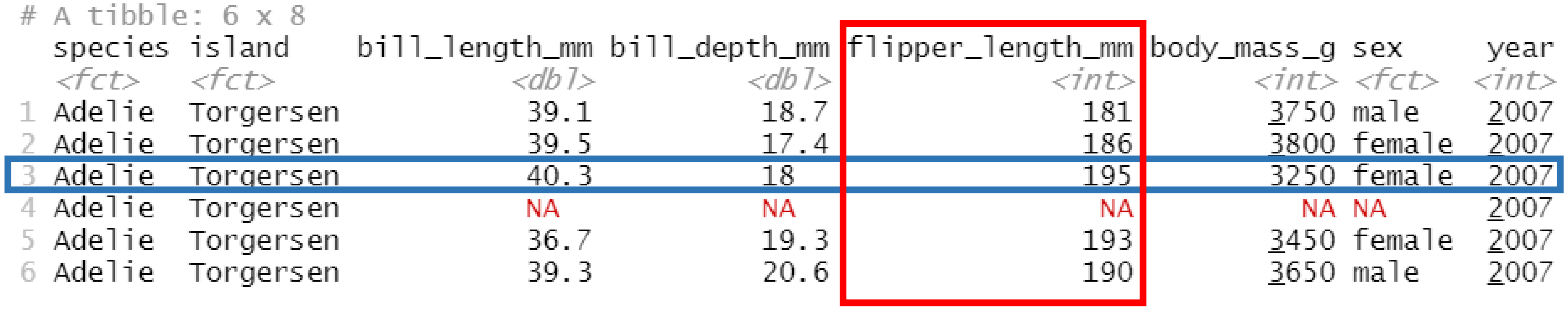


```
head(penguins)
```

```
# A tibble: 6 x 8
  species island    bill_length_mm bill_depth_mm flipper_length_mm body_mass_g 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 female  2007
3 Adelie  Torgersen           40.3          18                 195        3250 female  2007
4 Adelie  Torgersen           NA            NA                  NA          NA NA      2007
5 Adelie  Torgersen           36.7          19.3               193        3450 female  2007
6 Adelie  Torgersen           39.3          20.6               190        3650 male    2007
```

Tabular data

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



```
# A tibble: 6 x 8
  species island    bill_length_mm bill_depth_mm flipper_length_mm body_mass_g 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 female  2007
3 Adelie  Torgersen           40.3          18                 195        3250 female  2007
4 Adelie  Torgersen           NA            NA                  NA          NA NA      2007
5 Adelie  Torgersen           36.7          19.3               193        3450 female  2007
6 Adelie  Torgersen           39.3          20.6               190        3650 male    2007
```

Rows

Correspond to an instance of a specific type of thing, in this case an individual penguin.

Known as examples, observations or cases.

Columns

Correspond to a property or quality of the individual examples.

Known as features, variables or covariates.

What is data wrangling?

- Data wrangling is the process of transforming data from one form to another.
- Extracting, transforming, fusing and aggregating information from existing data.
- We can do this all in R with the Tidyverse, especially the Hadley Wickham’s dplyr.

A grammar for data wrangling

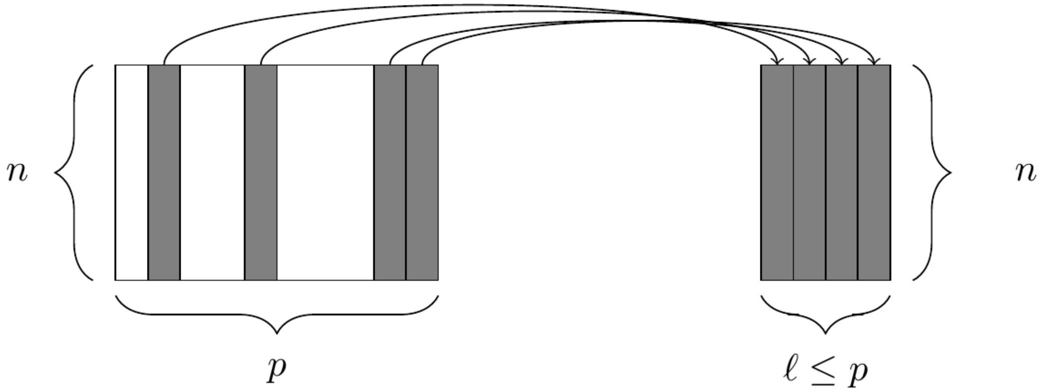
- The “nouns” of data wrangling are the data frames.

- Hadley Wickam identified five key “verbs” which can be applied to data frames:

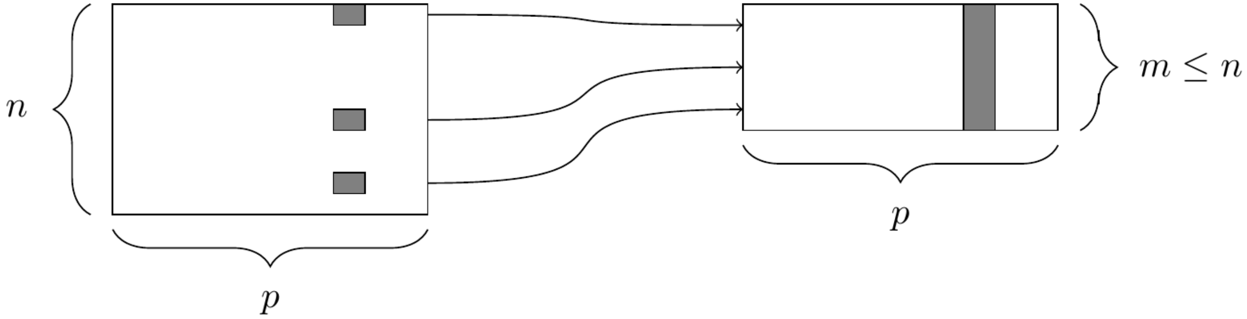
select()	-	Take a subset of columns.
filter()	-	Take a subset of rows.
mutate()	-	Add or modify existing columns.
arrange()	-	Sort rows.
summarize()	-	Aggregate data across existing rows.

The select and filter functions

Select



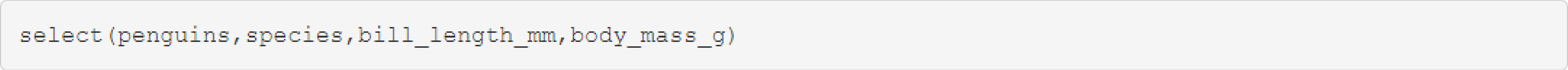
Filter



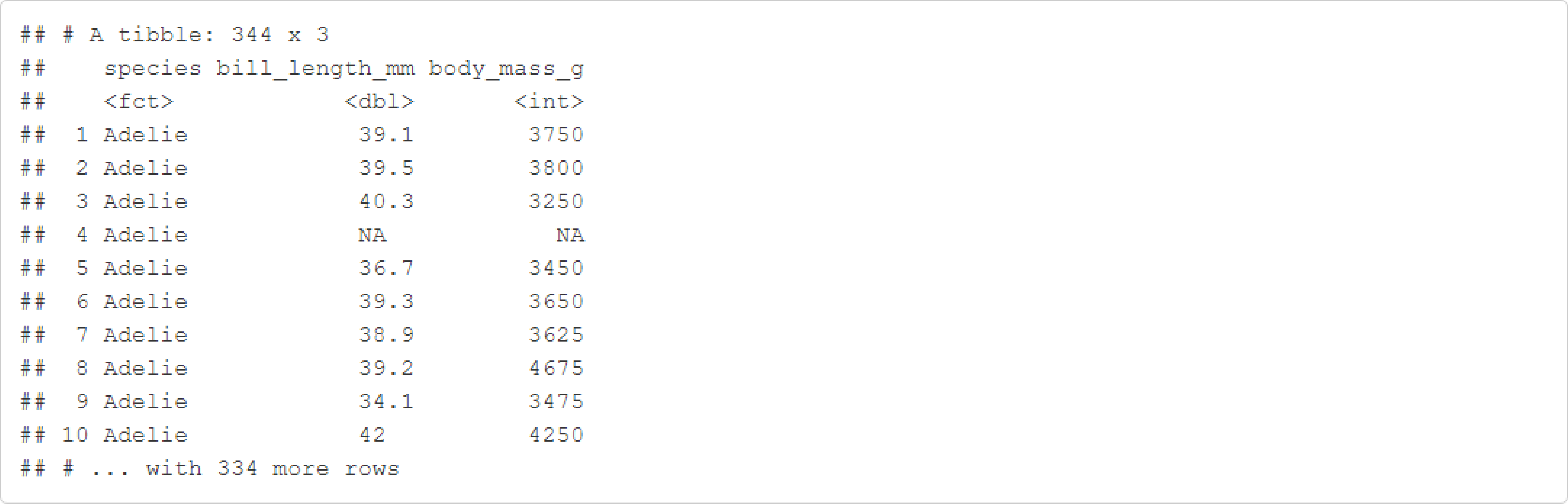
Diagrams from Baumer et al. Modern Data Science with R, 2017.

The select function

The select function allows us to extract several columns.



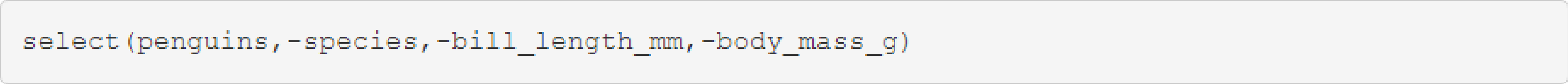
```
select(penguins,species,bill_length_mm,body_mass_g)
```



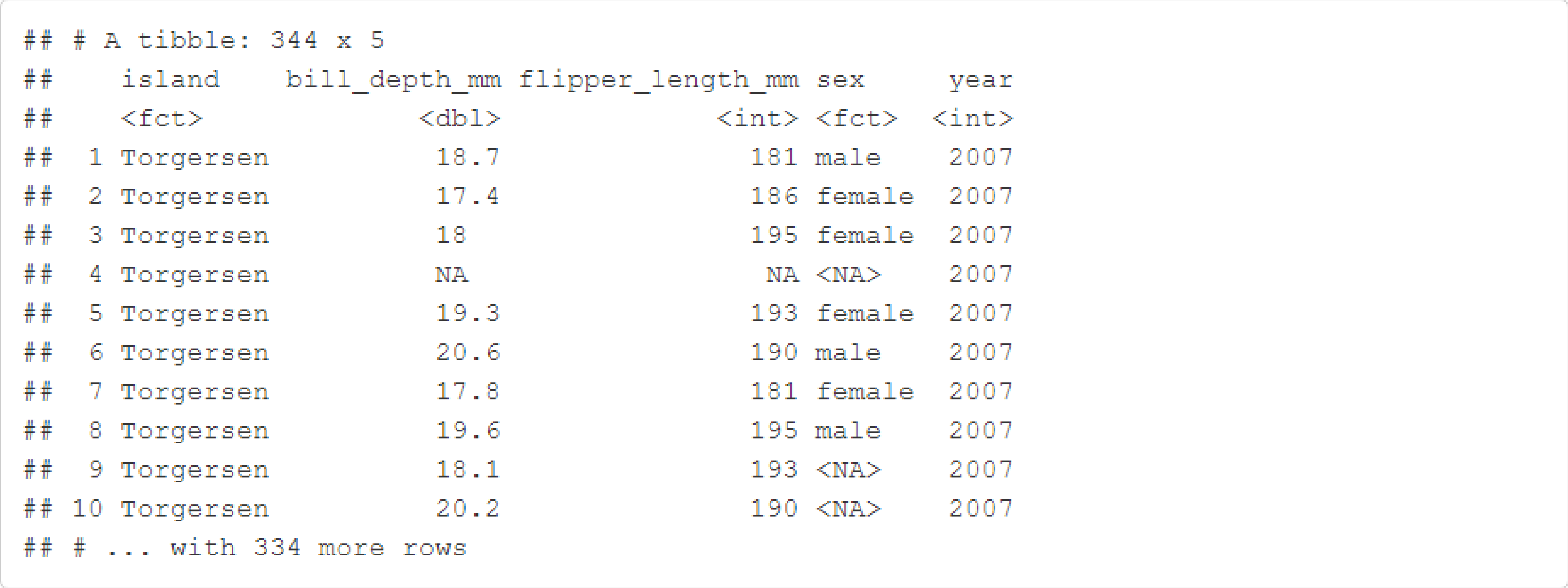
```
## # A tibble: 344 x 3
##    species bill_length_mm body_mass_g
##    <fct>            <dbl>       <int>
##  1 Adelie            39.1        3750
##  2 Adelie            39.5        3800
##  3 Adelie            40.3        3250
##  4 Adelie            NA            NA
##  5 Adelie            36.7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

The select function

The select function also allows us to remove several columns.



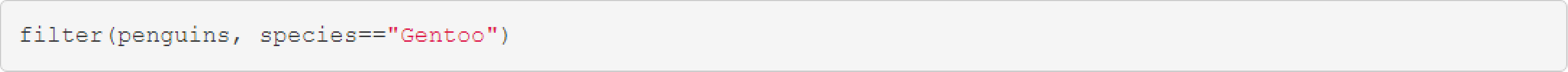
```
select(penguins,-species,-bill_length_mm,-body_mass_g)
```



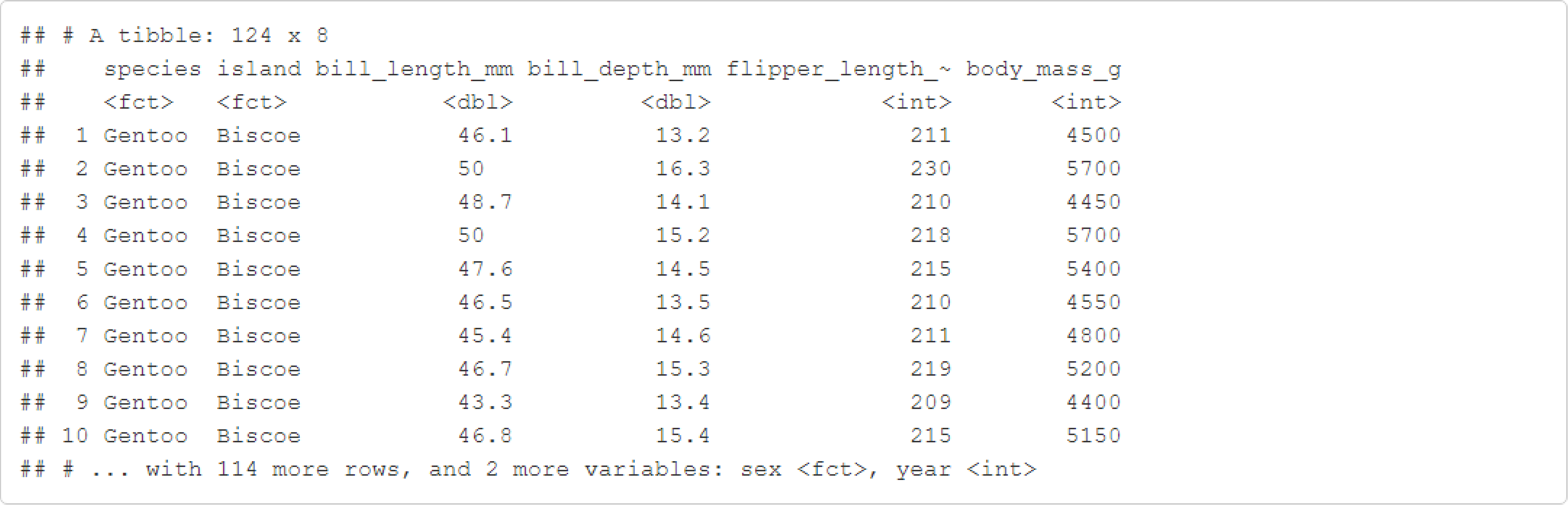
```
## # A tibble: 344 x 5
##    island    bill_depth_mm flipper_length_mm sex     year
##    <fct>             <dbl>             <int> <fct>  <int>
##  1 Torgersen          18.7               181 male    2007
##  2 Torgersen          17.4               186 female  2007
##  3 Torgersen          18                 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
```

The filter function

The filter function allows us to extract a subset of rows.



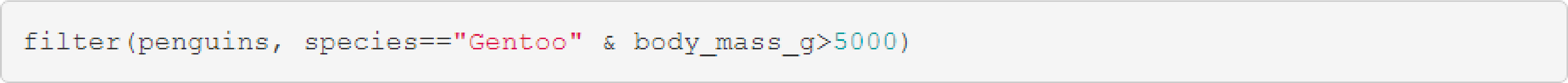
```
filter(penguins, species=="Gentoo")
```



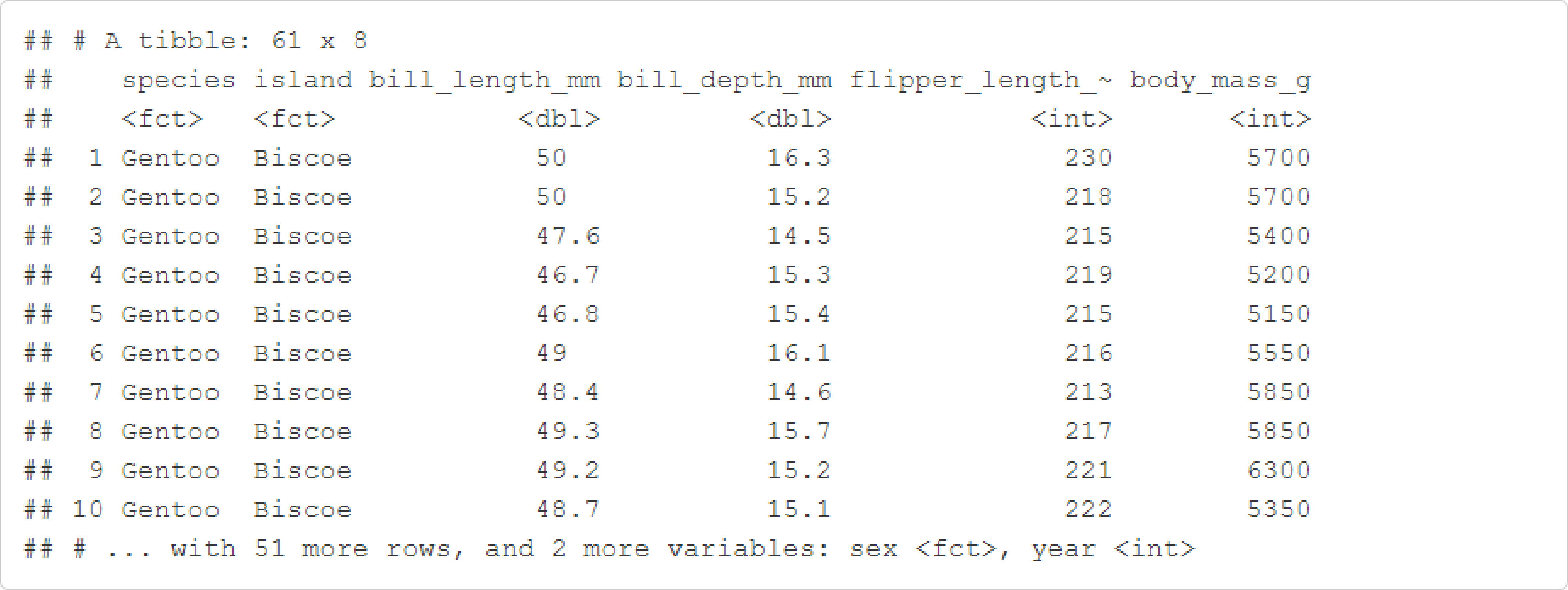
```
## # A tibble: 124 x 8
##    species island bill_length_mm bill_depth_mm flipper_length_~ body_mass_g
##    <fct>   <fct>           <dbl>         <dbl>            <int>       <int>
##  1 Gentoo  Biscoe           46.1          13.2              211        4500
##  2 Gentoo  Biscoe           50            16.3              230        5700
##  3 Gentoo  Biscoe           48.7          14.1              210        4450
##  4 Gentoo  Biscoe           50            15.2              218        5700
##  5 Gentoo  Biscoe           47.6          14.5              215        5400
##  6 Gentoo  Biscoe           46.5          13.5              210        4550
##  7 Gentoo  Biscoe           45.4          14.6              211        4800
##  8 Gentoo  Biscoe           46.7          15.3              219        5200
##  9 Gentoo  Biscoe           43.3          13.4              209        4400
## 10 Gentoo  Biscoe           46.8          15.4              215        5150
## # ... with 114 more rows, and 2 more variables: sex <fct>, year <int>
```

The filter function

We can also combine two or more conditions within the filter function.



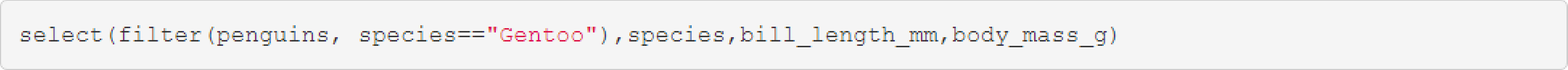
```
filter(penguins, species=="Gentoo" & body_mass_g>5000)
```



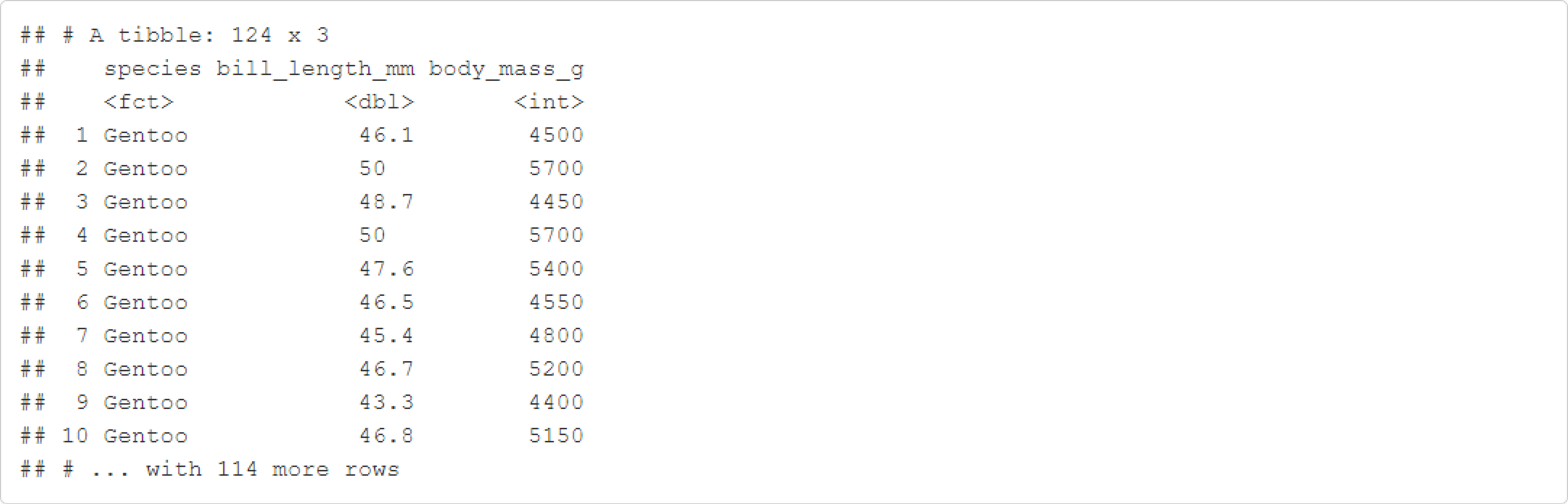
```
## # A tibble: 61 x 8
##    species island bill_length_mm bill_depth_mm flipper_length_~ body_mass_g
##    <fct>   <fct>           <dbl>         <dbl>            <int>       <int>
##  1 Gentoo  Biscoe           50            16.3              230        5700
##  2 Gentoo  Biscoe           50            15.2              218        5700
##  3 Gentoo  Biscoe           47.6          14.5              215        5400
##  4 Gentoo  Biscoe           46.7          15.3              219        5200
##  5 Gentoo  Biscoe           46.8          15.4              215        5150
##  6 Gentoo  Biscoe           49            16.1              216        5550
##  7 Gentoo  Biscoe           48.4          14.6              213        5850
##  8 Gentoo  Biscoe           49.3          15.7              217        5850
##  9 Gentoo  Biscoe           49.2          15.2              221        6300
## 10 Gentoo  Biscoe           48.7          15.1              222        5350
## # ... with 51 more rows, and 2 more variables: sex <fct>, year <int>
```

Combining filter & select functions

We often combine filter with select to get a sub table.

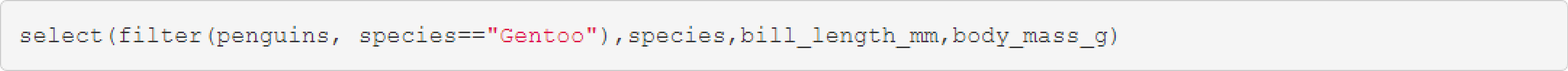


```
select(filter(penguins, species=="Gentoo"),species,bill_length_mm,body_mass_g)
```



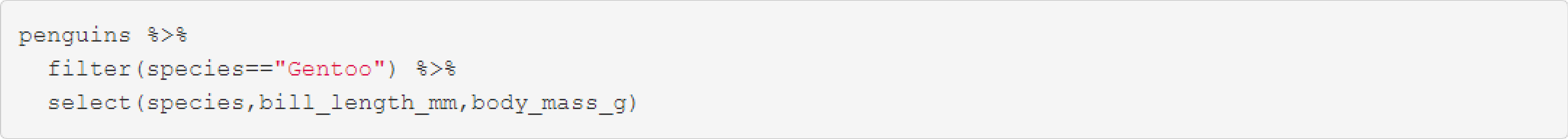
```
## # A tibble: 124 x 3
##    species bill_length_mm body_mass_g
##    <fct>            <dbl>       <int>
##  1 Gentoo            46.1        4500
##  2 Gentoo            50          5700
##  3 Gentoo            48.7        4450
##  4 Gentoo            50          5700
##  5 Gentoo            47.6        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
```

The pipe operator

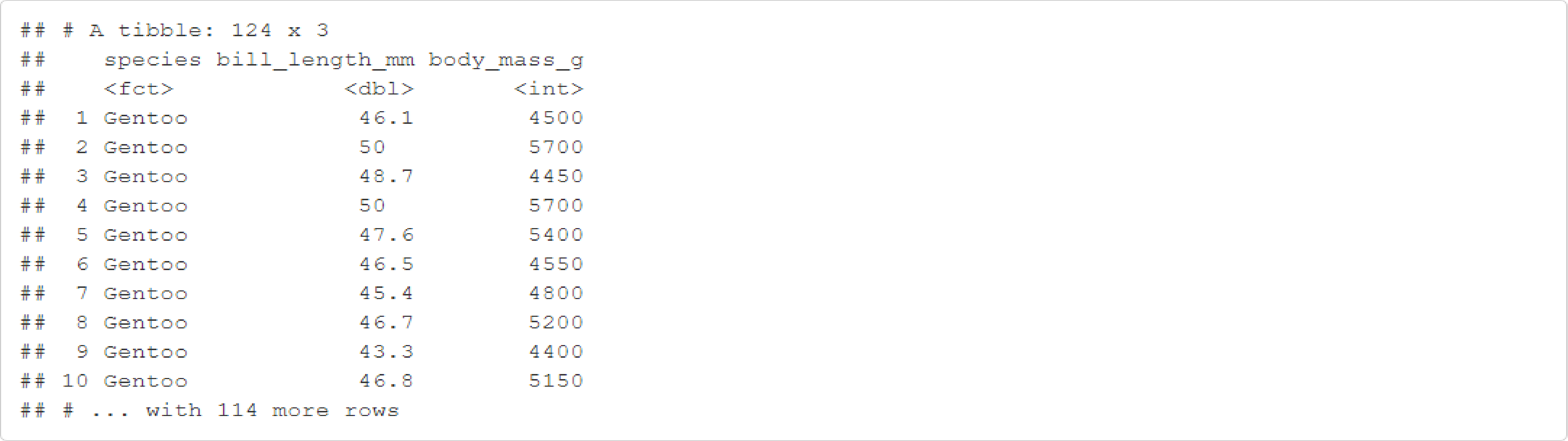


```
select(filter(penguins, species=="Gentoo"),species,bill_length_mm,body_mass_g)
```

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



```
penguins %>%
  filter(species=="Gentoo") %>%
  select(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
```

The pipe operator

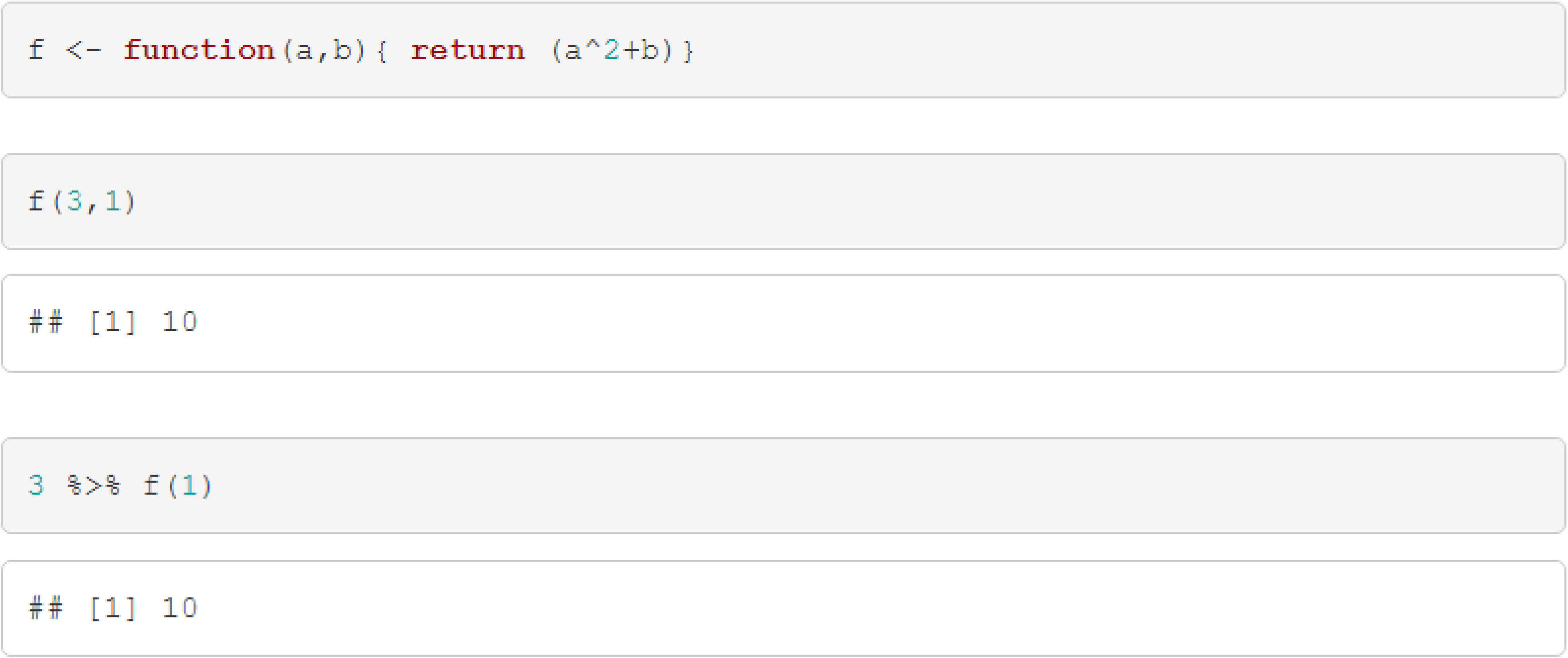
The pipe operator %>% is taken from the magrittr package which is also part of the tidyverse.

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

Image from renemagritte.org and magrittr.tidyverse.org

The pipe operator

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)}
```

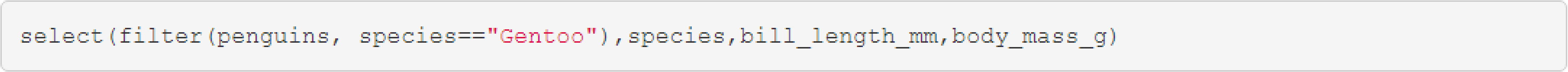
```
f(3,1)
```

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

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

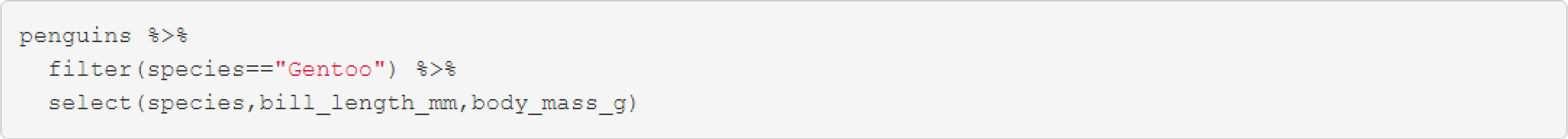
```
## [1] 10
```

The pipe operator

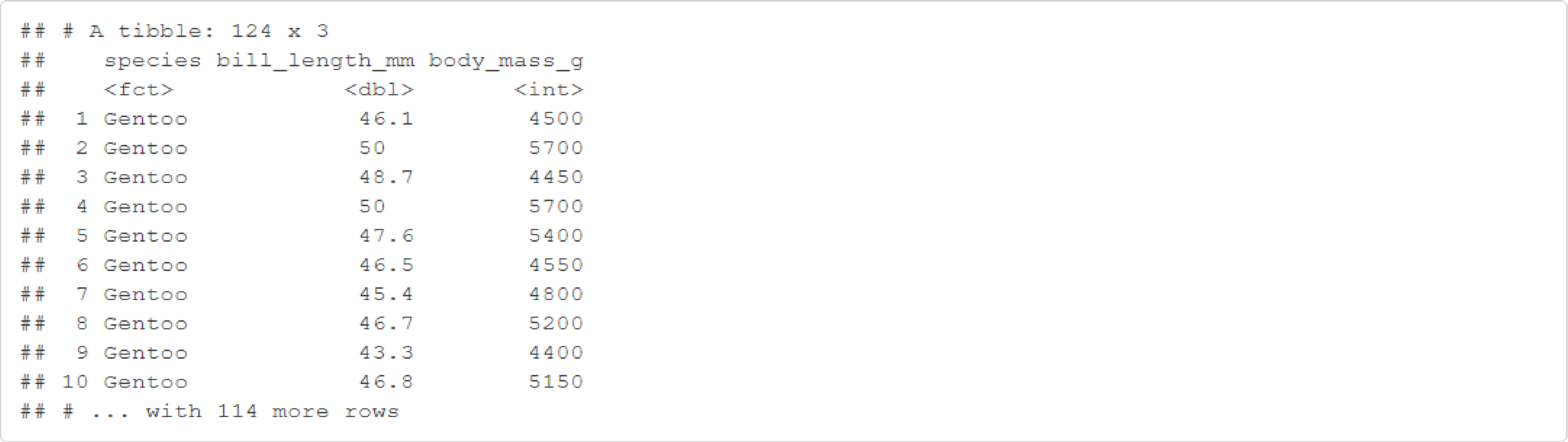


```
select(filter(penguins, species=="Gentoo"),species,bill_length_mm,body_mass_g)
```

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



```
penguins %>%
  filter(species=="Gentoo") %>%
  select(species,bill_length_mm,body_mass_g)
```

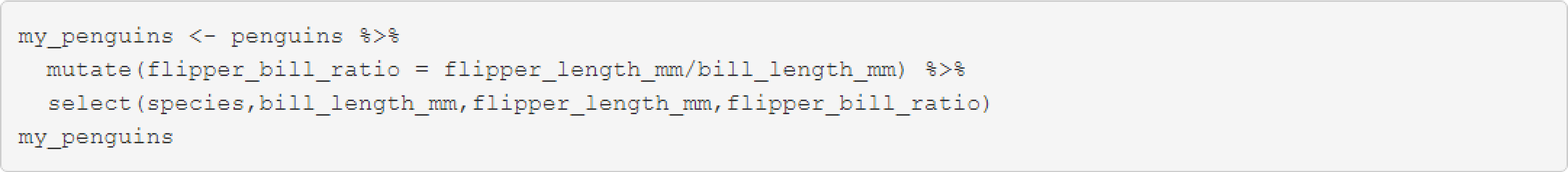


```
## # A tibble: 124 x 3
##    species bill_length_mm body_mass_g
##    <fct>            <dbl>       <int>
##  1 Gentoo            46.1        4500
##  2 Gentoo            50          5700
##  3 Gentoo            48.7        4450
##  4 Gentoo            50          5700
##  5 Gentoo            47.6        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
```

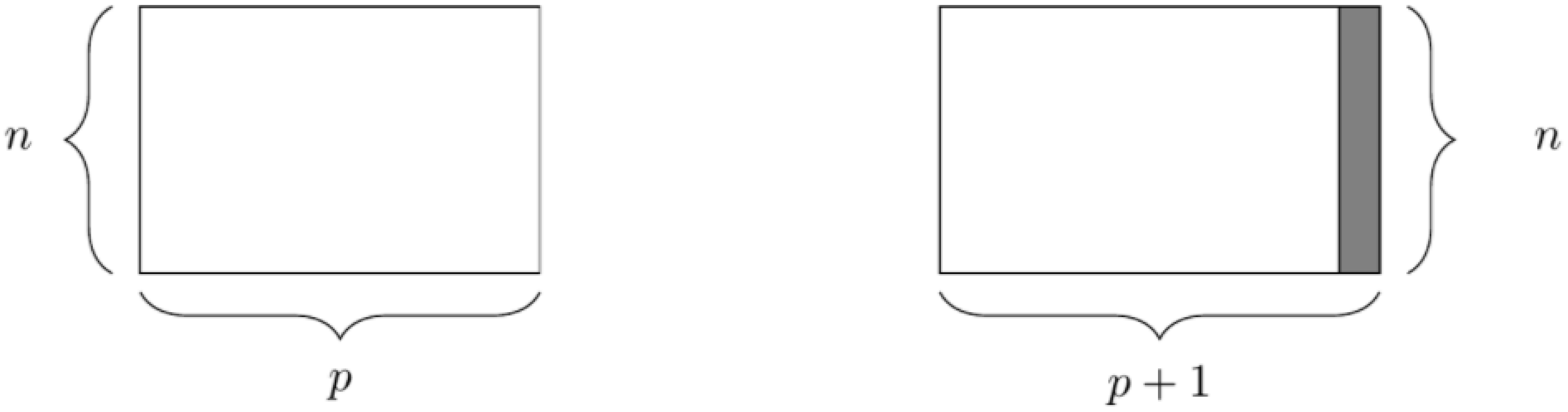
Now take a break!

The mutate function

The mutate function allows us to create a new column as a function of existing columns.

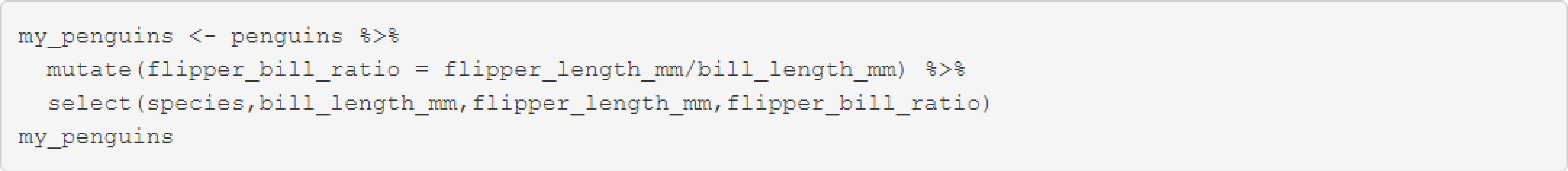


```
my_penguins <- penguins %>%
  mutate(flipper_bill_ratio = flipper_length_mm/bill_length_mm) %>%
  select(species,bill_length_mm,flipper_length_mm,flipper_bill_ratio)
my_penguins
```

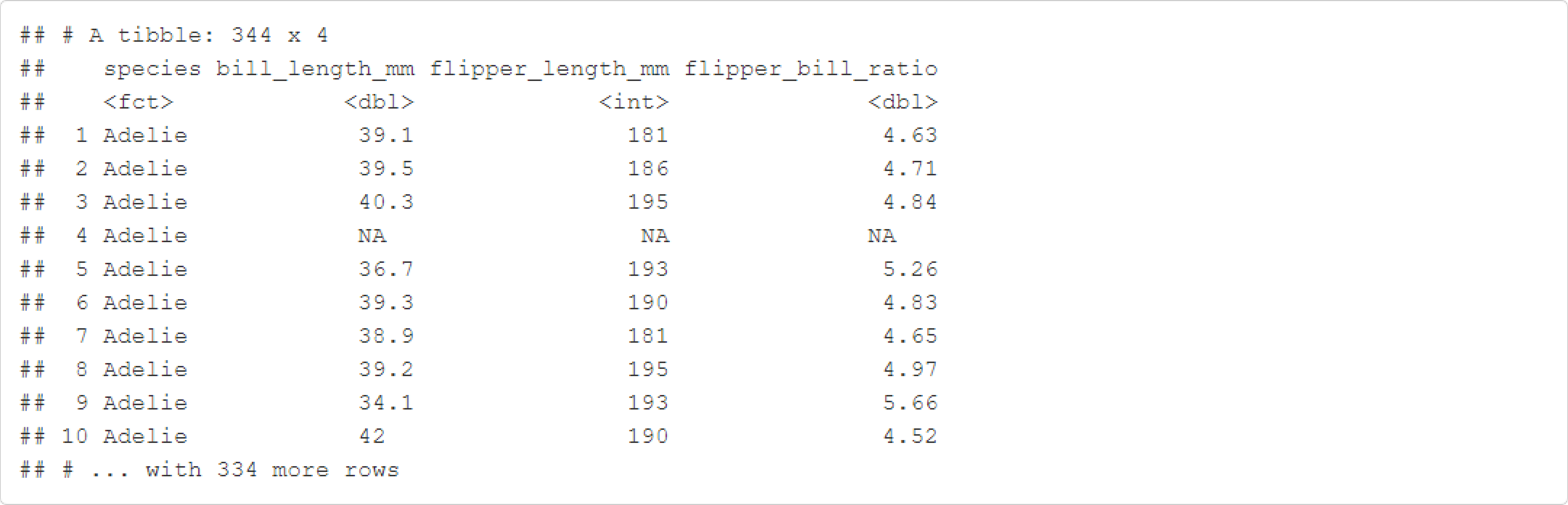


The mutate function

The mutate function allows us to create a new column as a function of existing coloumns.

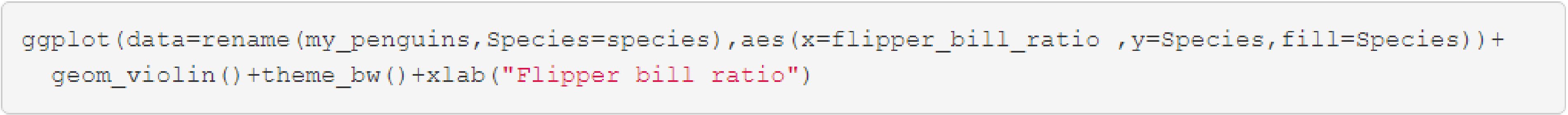


```
my_penguins <- penguins %>%
  mutate(flipper_bill_ratio = flipper_length_mm/bill_length_mm) %>%
  select(species,bill_length_mm,flipper_length_mm,flipper_bill_ratio)
my_penguins
```

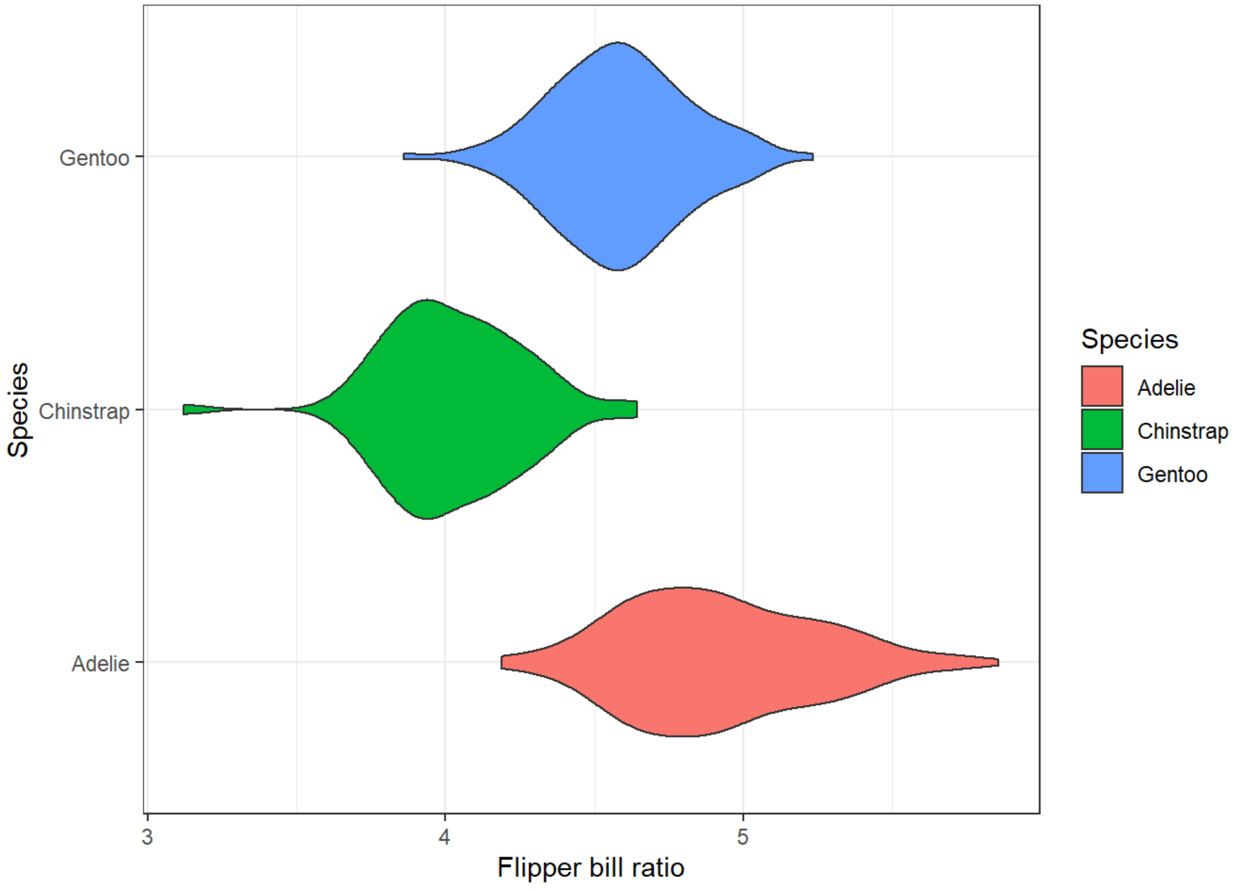


```
## # A tibble: 344 x 4
##    species bill_length_mm flipper_length_mm flipper_bill_ratio
##    <fct>            <dbl>             <int>              <dbl>
##  1 Adelie            39.1               181               4.63
##  2 Adelie            39.5               186               4.71
##  3 Adelie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

The mutate function

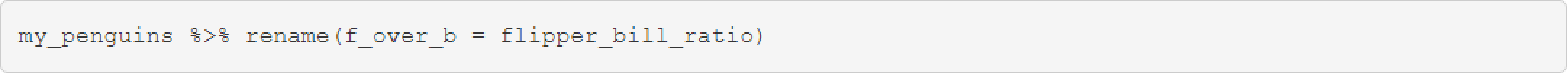


```r
ggplot(data=rename(my_penguins,Species=species),aes(x=flipper_bill_ratio ,y=Species,fill=Species))+
  geom_violin()+theme_bw()+xlab("Flipper bill ratio")
```

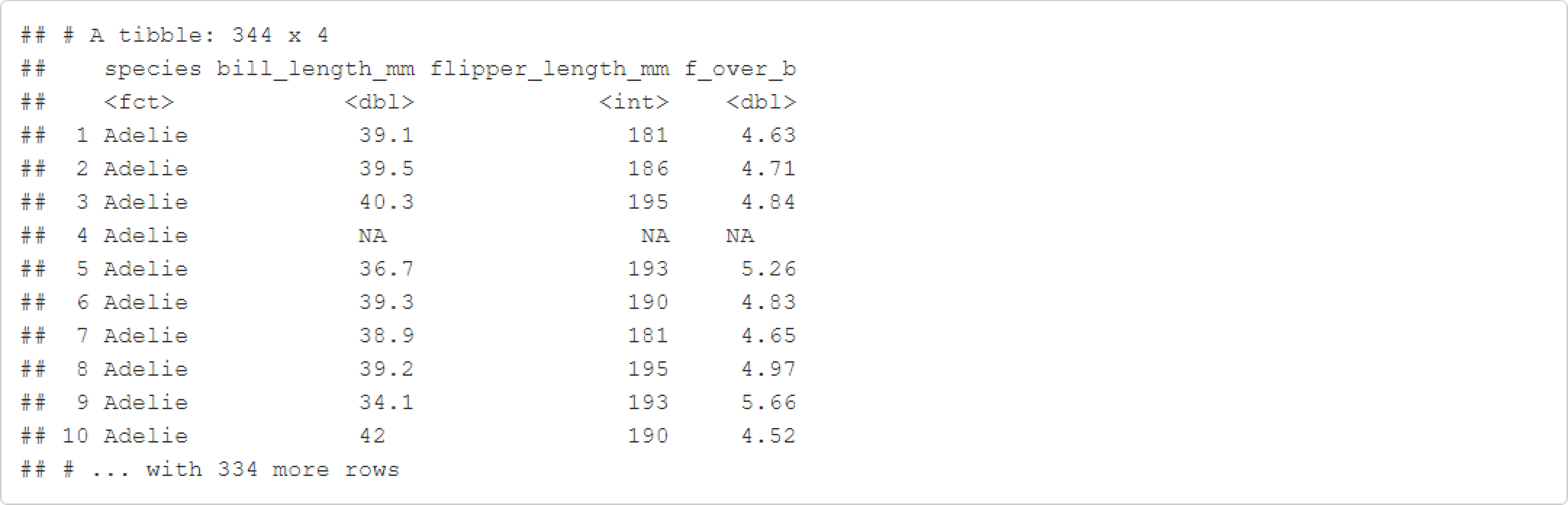


The rename function

The rename function allows us to rename an existing column.



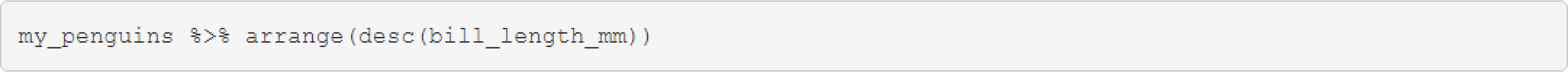
```
my_penguins %>% rename(f_over_b = flipper_bill_ratio)
```



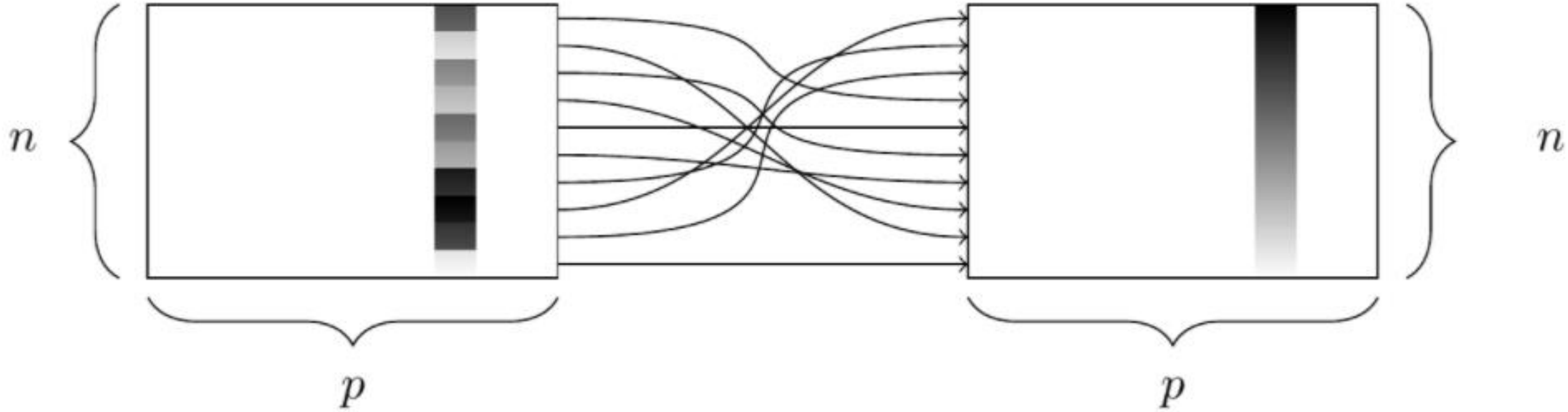
```
## # A tibble: 344 x 4
##    species bill_length_mm flipper_length_mm f_over_b
##    <fct>            <dbl>             <int>    <dbl>
##  1 Adelie            39.1               181     4.63
##  2 Adelie            39.5               186     4.71
##  3 Adelie            40.3               195     4.84
##  4 Adelie            NA                  NA    NA   
##  5 Adelie            36.7               193     5.26
##  6 Adelie            39.3               190     4.83
##  7 Adelie            38.9               181     4.65
##  8 Adelie            39.2               195     4.97
##  9 Adelie            34.1               193     5.66
## 10 Adelie            42                 190     4.52
## # ... with 334 more rows
```

The arrange function

We can sort the rows of a table via the arrange function.

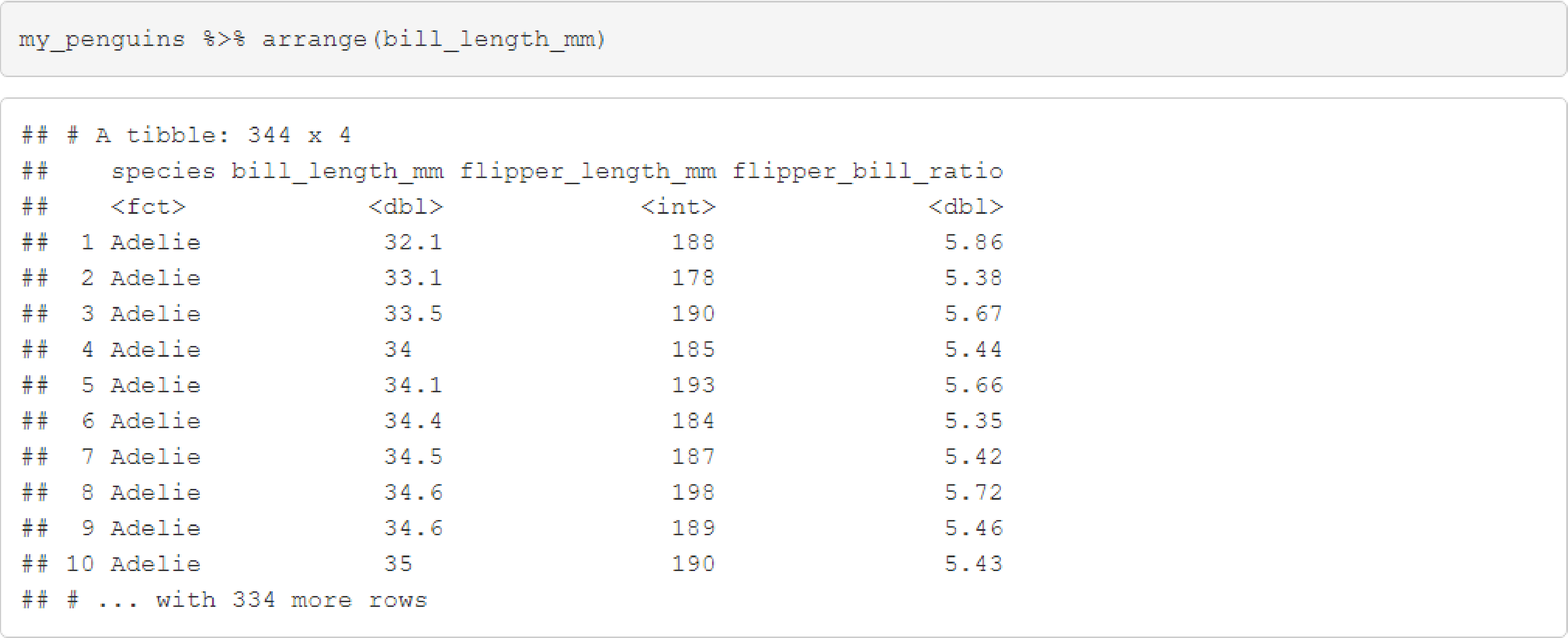


```
my_penguins %>% arrange(desc(bill_length_mm))
```



The arrange function

We can sort the rows of a table via the arrange function.

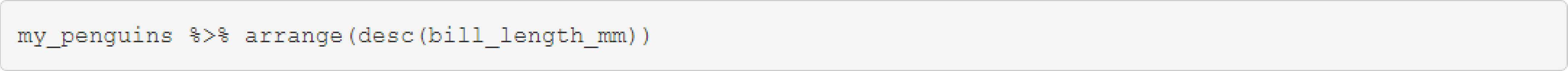


```
my_penguins %>% arrange(bill_length_mm)
```

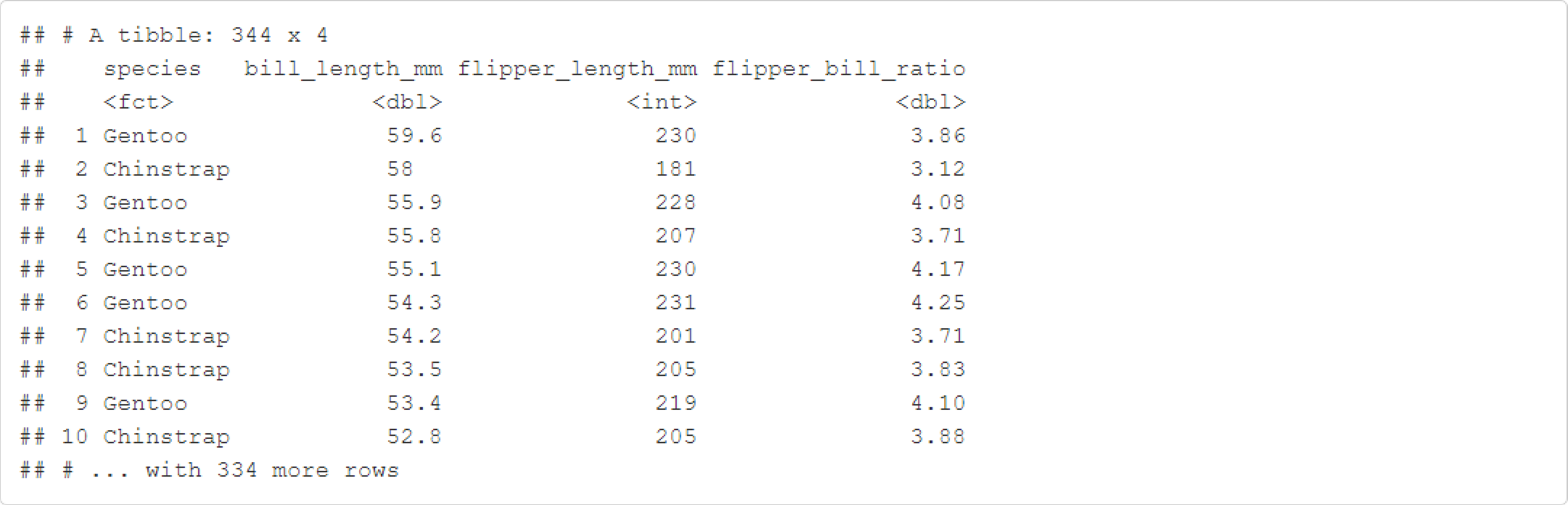
```
## # A tibble: 344 x 4
##    species bill_length_mm flipper_length_mm flipper_bill_ratio
##    <fct>            <dbl>             <int>              <dbl>
##  1 Adelie            32.1               188               5.86
##  2 Adelie            33.1               178               5.38
##  3 Adelie            33.5               190               5.67
##  4 Adelie            34                 185               5.44
##  5 Adelie            34.1               193               5.66
##  6 Adelie            34.4               184               5.35
##  7 Adelie            34.5               187               5.42
##  8 Adelie            34.6               198               5.72
##  9 Adelie            34.6               189               5.46
## 10 Adelie            35                 190               5.43
## # ... with 334 more rows
```

The arrange function

We can sort the rows of a table via the arrange function.



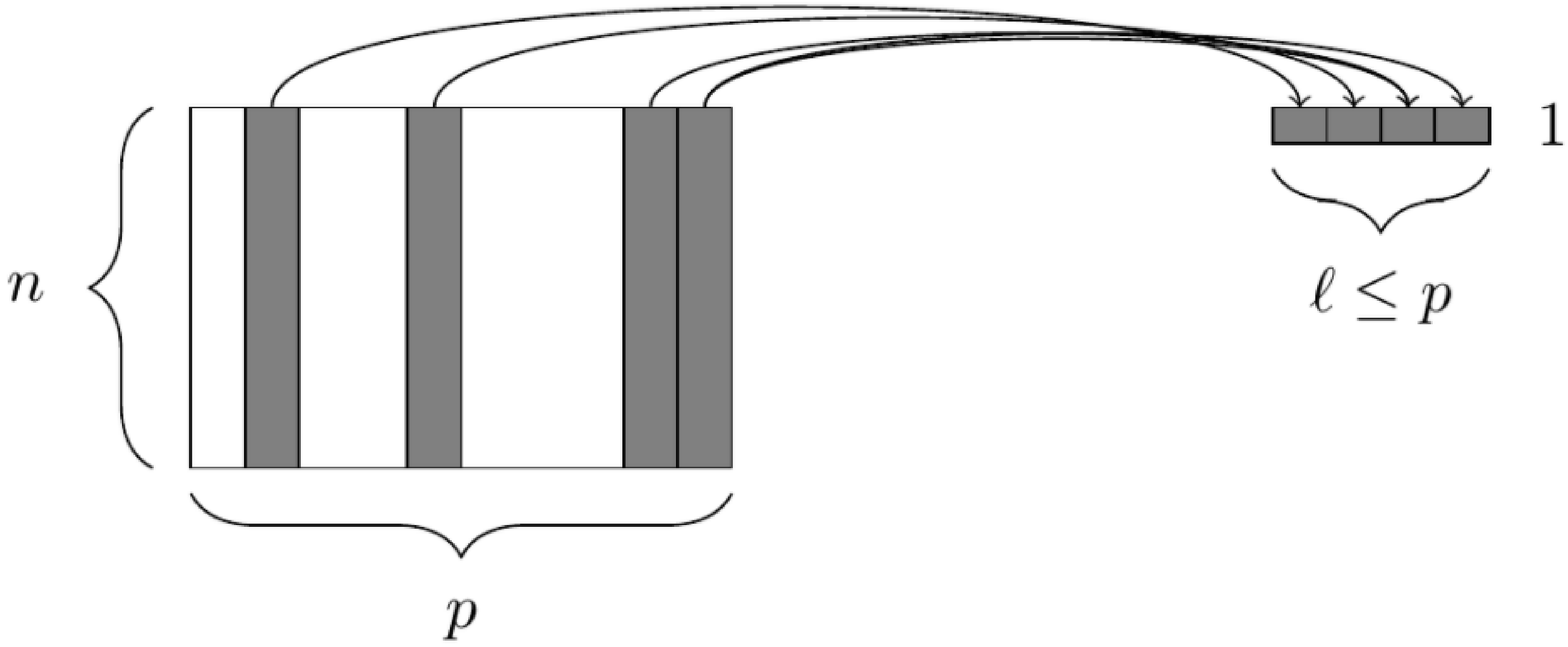
```
my_penguins %>% arrange(desc(bill_length_mm))
```



```
## # A tibble: 344 x 4
##    species   bill_length_mm flipper_length_mm flipper_bill_ratio
##    <fct>              <dbl>             <int>              <dbl>
##  1 Gentoo              59.6               230               3.86
##  2 Chinstrap           58                 181               3.12
##  3 Gentoo              55.9               228               4.08
##  4 Chinstrap           55.8               207               3.71
##  5 Gentoo              55.1               230               4.17
##  6 Gentoo              54.3               231               4.25
##  7 Chinstrap           54.2               201               3.71
##  8 Chinstrap           53.5               205               3.83
##  9 Gentoo              53.4               219               4.10
## 10 Chinstrap           52.8               205               3.88
## # ... with 334 more rows
```

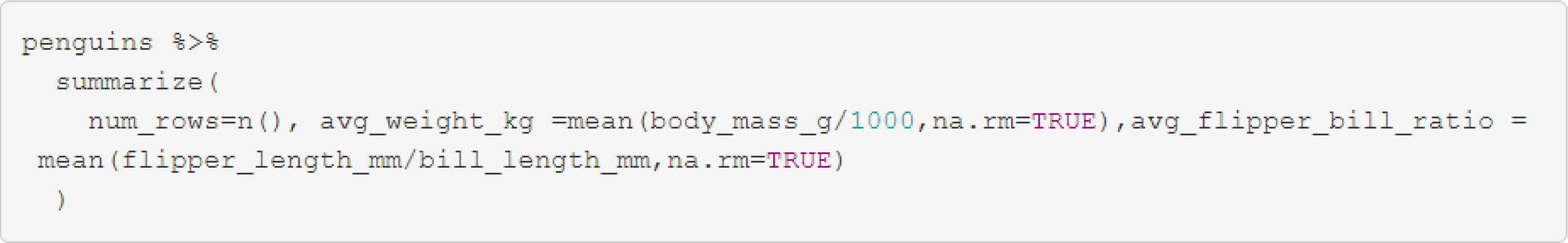
Summarizing data

To understand data we can extract summary statistics from a data frame.

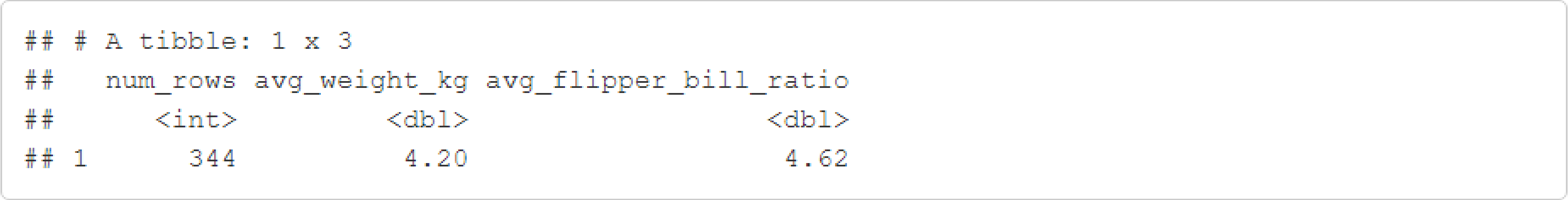


The summarize function

The summarize function computes vector functions across the entire data frame.



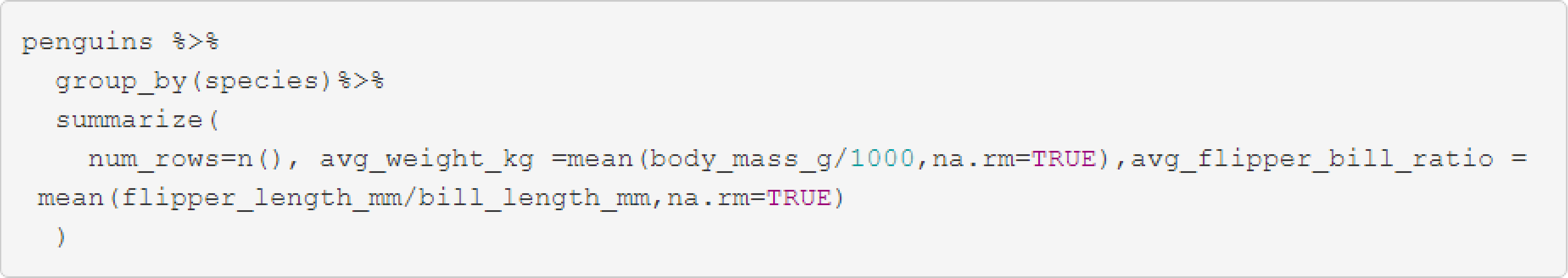
```
penguins %>%
  summarize(
    num_rows=n(), avg_weight_kg =mean(body_mass_g/1000,na.rm=TRUE),avg_flipper_bill_ratio =
 mean(flipper_length_mm/bill_length_mm,na.rm=TRUE)
  )
```



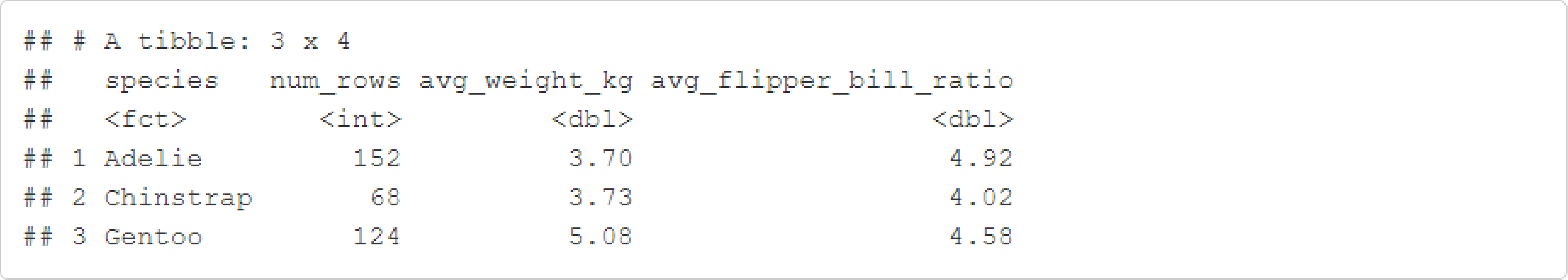
```
## # A tibble: 1 x 3
##   num_rows avg_weight_kg avg_flipper_bill_ratio
##      <int>         <dbl>                  <dbl>
## 1      344          4.20                   4.62
```

The groupby function

To obtain summaries by group we can combine the summarize and groupby functions.



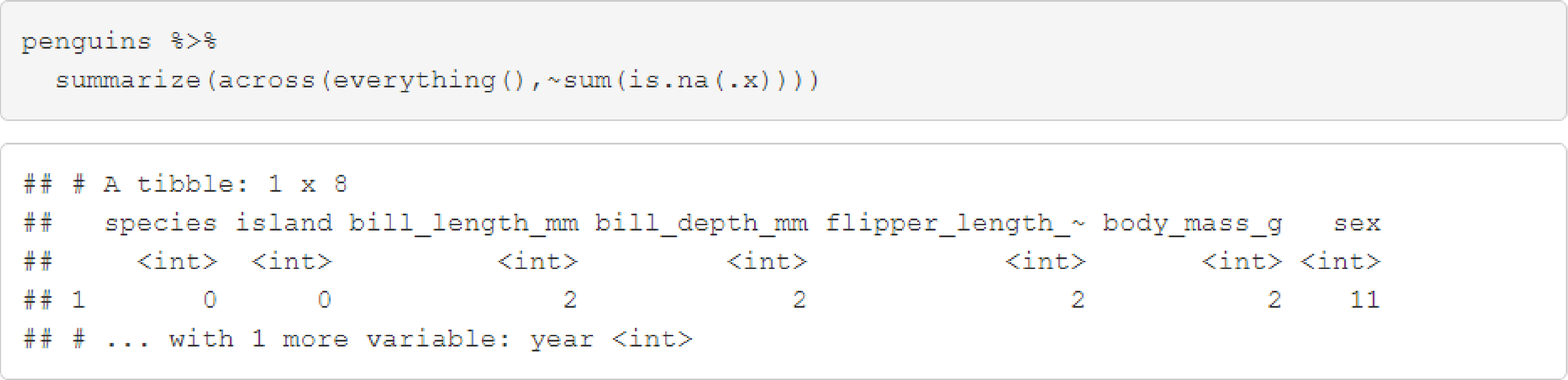
```
penguins %>%
  group_by(species)%>%
  summarize(
    num_rows=n(), avg_weight_kg =mean(body_mass_g/1000,na.rm=TRUE),avg_flipper_bill_ratio =
 mean(flipper_length_mm/bill_length_mm,na.rm=TRUE)
  )
```



```
## # 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
```

The across function

The across function allows us to apply a function within summarize to all columns at once.

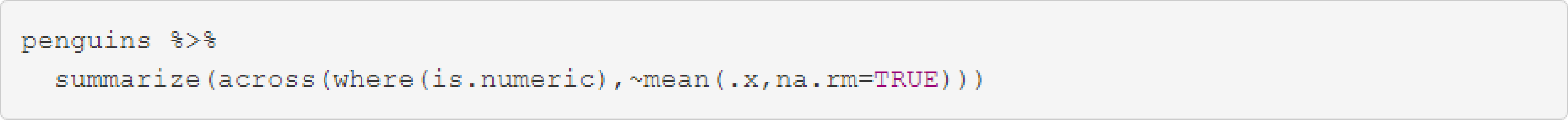


```
penguins %>%
  summarize(across(everything(),~sum(is.na(.x))))
```

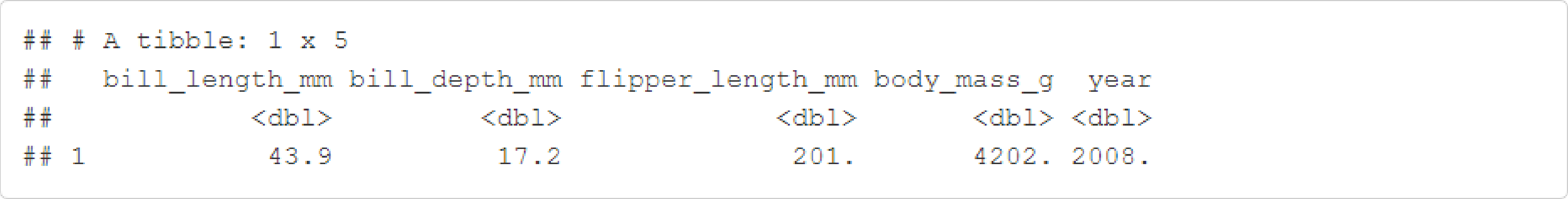
```
## # A tibble: 1 x 8
##   species island bill_length_mm bill_depth_mm flipper_length_~ body_mass_g   sex
##     <int>  <int>          <int>         <int>            <int>       <int> <int>
## 1       0      0              2             2                2           2    11
## # ... with 1 more variable: year <int>
```

The across function combined with where

We can also restrict apply the function to a subset of columns of a prescribed form.



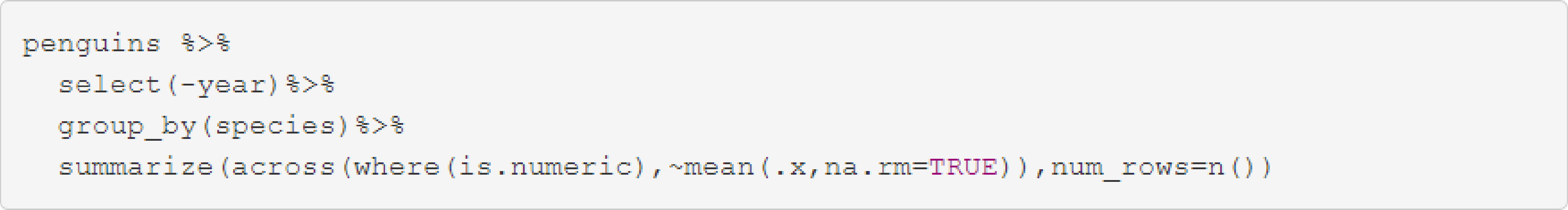
```
penguins %>%
  summarize(across(where(is.numeric),~mean(.x,na.rm=TRUE)))
```



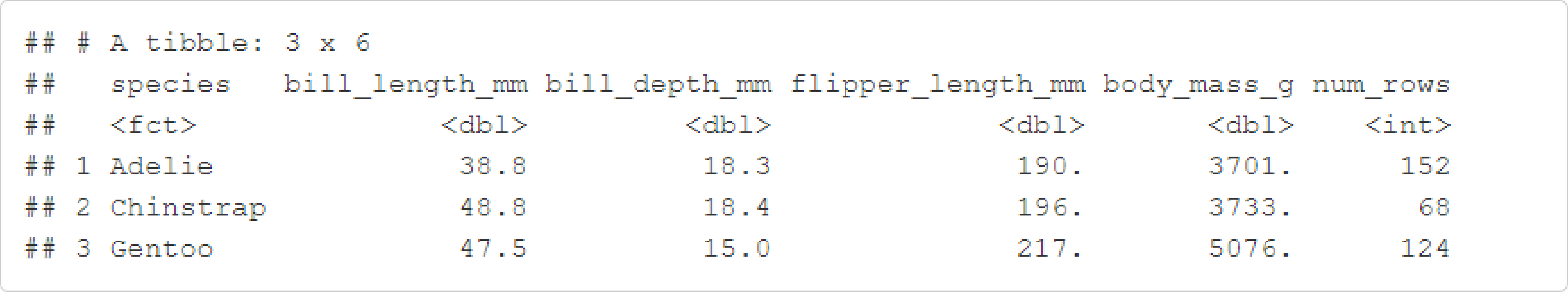
```
## # A tibble: 1 x 5
##   bill_length_mm bill_depth_mm flipper_length_mm body_mass_g  year
##            <dbl>         <dbl>             <dbl>       <dbl> <dbl>
## 1           43.9          17.2              201.       4202. 2008.
```

Combining the summarize, groupby and across functions

To obtain summaries by group we can combine the summarize and groupby functions.



```
penguins %>%
  select(-year)%>%
  group_by(species)%>%
  summarize(across(where(is.numeric),~mean(.x,na.rm=TRUE)),num_rows=n())
```

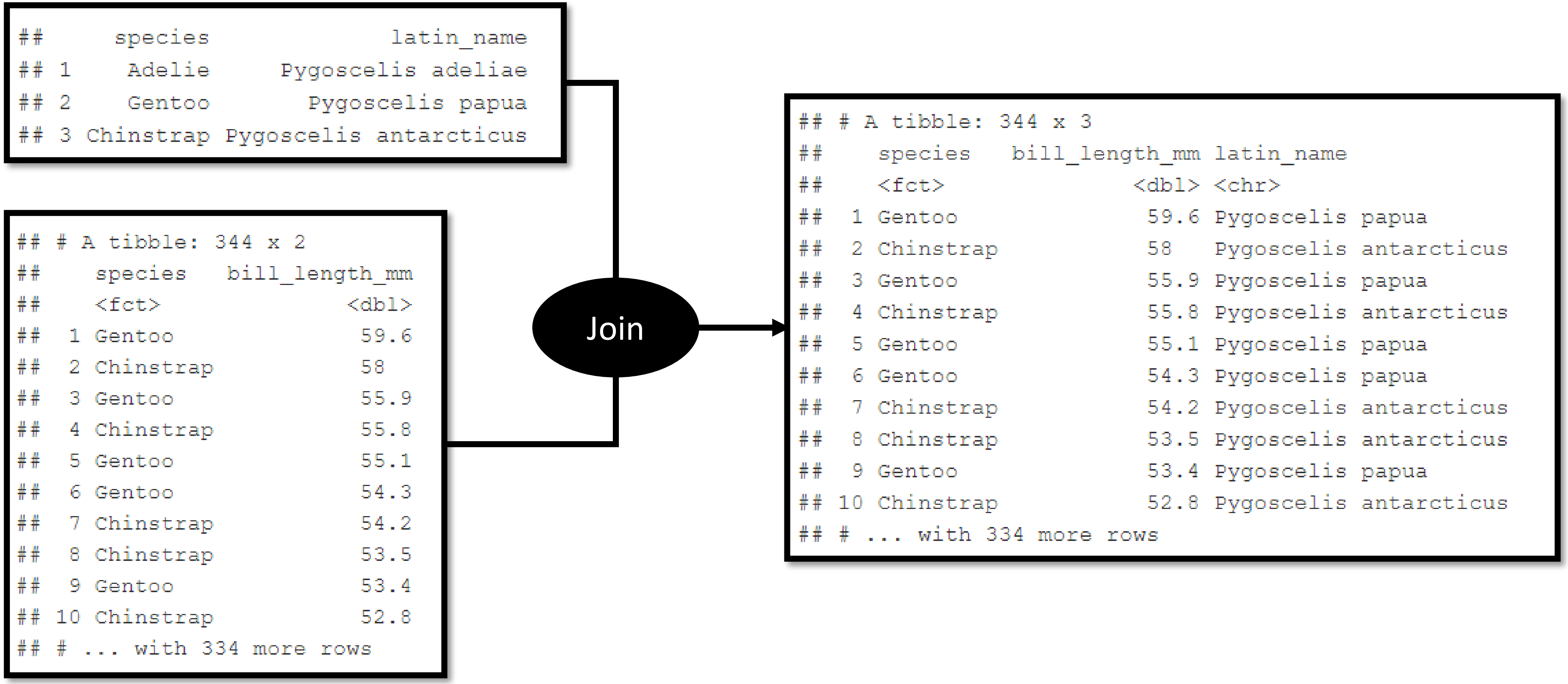


```
## # A tibble: 3 x 6
##   species   bill_length_mm bill_depth_mm flipper_length_mm body_mass_g num_rows
##   <fct>              <dbl>         <dbl>             <dbl>       <dbl>    <int>
## 1 Adelie              38.8          18.3              190.       3701.      152
## 2 Chinstrap           48.8          18.4              196.       3733.       68
## 3 Gentoo              47.5          15.0              217.       5076.      124
```

Now take a break!

Join functions

Join functions allow us to fuse multiple data frames.



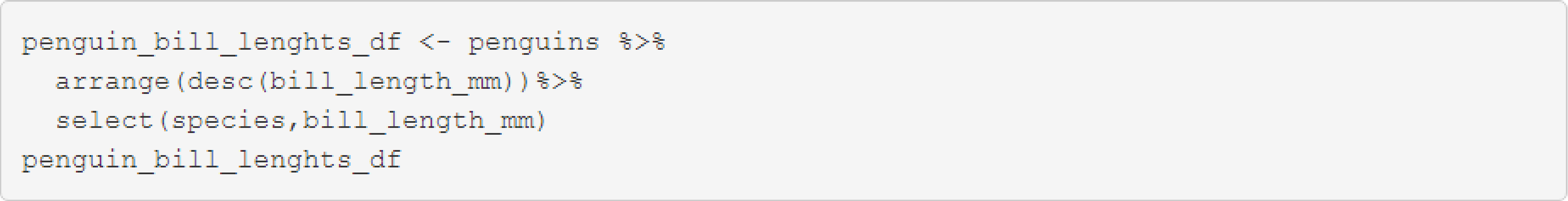
```
##     species             latin_name
## 1    Adelie     Pygoscelis adeliae
## 2    Gentoo       Pygoscelis papua
## 3 Chinstrap Pygoscelis antarcticus
```

```
## # A tibble: 344 x 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
```

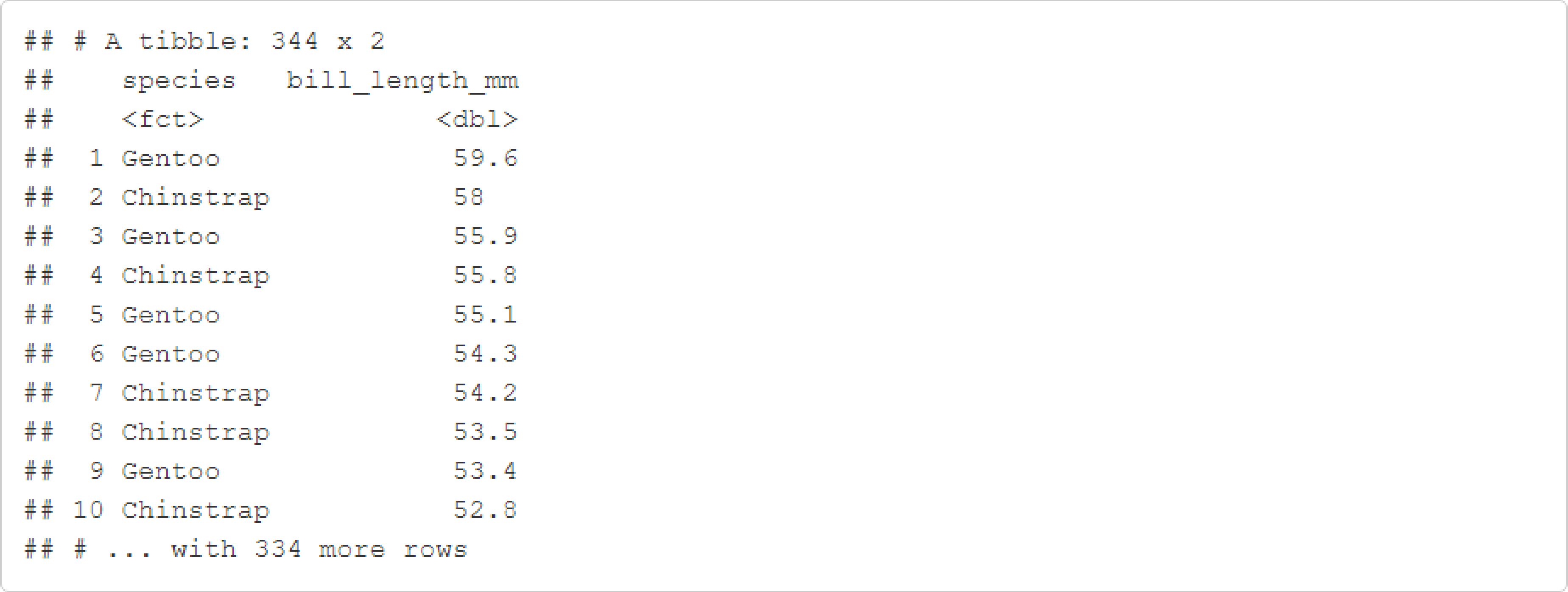
```
## # A tibble: 344 x 3
##    species   bill_length_mm latin_name
##    <fct>              <dbl> <chr>
##  1 Gentoo              59.6 Pygoscelis papua
##  2 Chinstrap           58   Pygoscelis antarcticus
##  3 Gentoo              55.9 Pygoscelis papua
##  4 Chinstrap           55.8 Pygoscelis antarcticus
##  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
```

Join functions

First we extract a data frame of bill lengths by species.



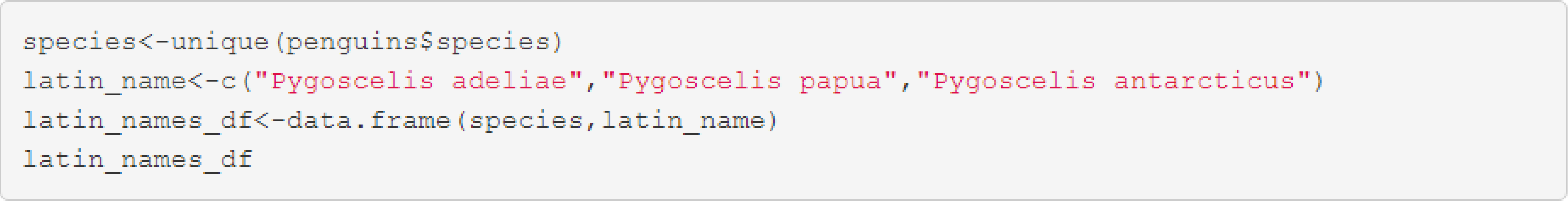
```
penguin_bill_lenghts_df <- penguins %>%
  arrange(desc(bill_length_mm))%>%
  select(species,bill_length_mm)
penguin_bill_lenghts_df
```



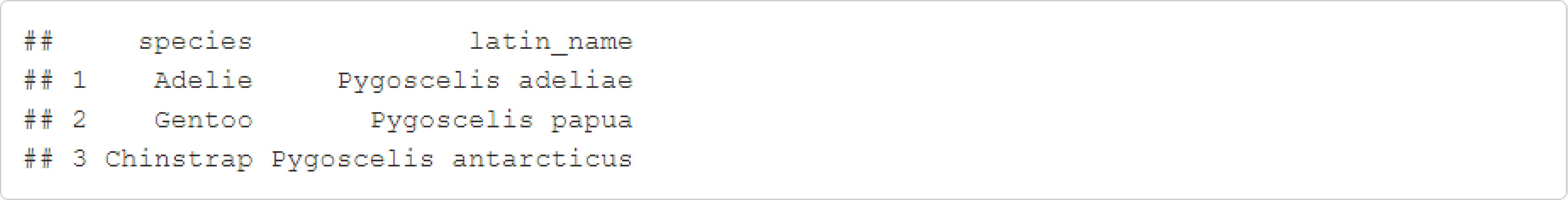
```
## # A tibble: 344 x 2
##    species   bill_length_mm
##    <fct>              <dbl>
##  1 Gentoo              59.6
##  2 Chinstrap           58
##  3 Gentoo              55.9
##  4 Chinstrap           55.8
##  5 Gentoo              55.1
##  6 Gentoo              54.3
##  7 Chinstrap           54.2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

Join functions

Next we create a data frame of latin species names.



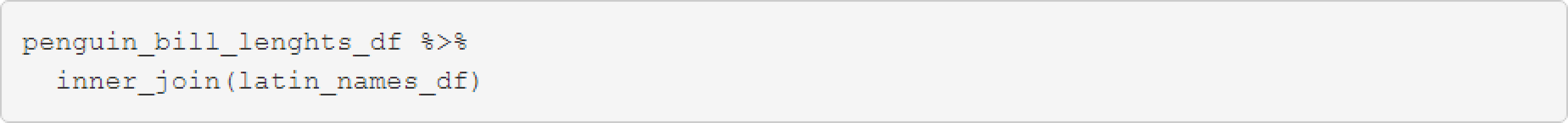
```
species<-unique(penguins$species)
latin_name<-c("Pygoscelis adeliae","Pygoscelis papua","Pygoscelis antarcticus")
latin_names_df<-data.frame(species,latin_name)
latin_names_df
```



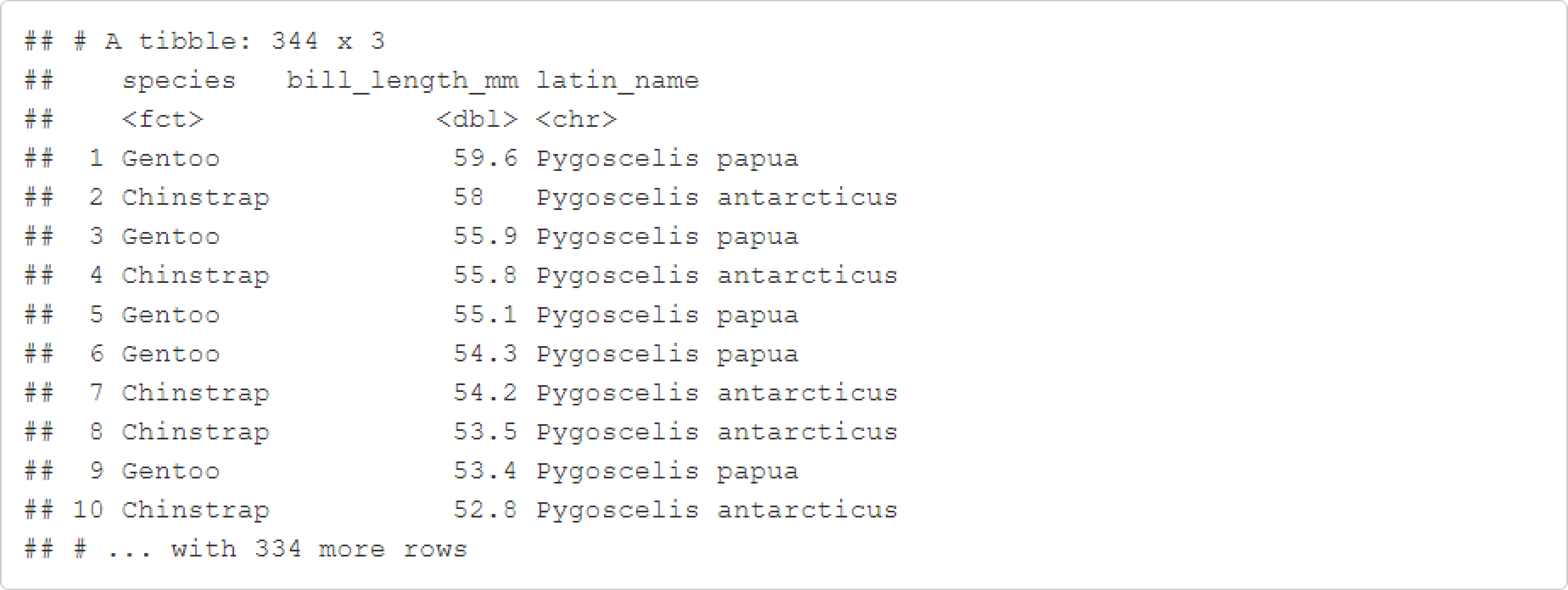
```
##     species             latin_name
## 1    Adelie     Pygoscelis adeliae
## 2    Gentoo       Pygoscelis papua
## 3 Chinstrap Pygoscelis antarcticus
```

Join functions

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



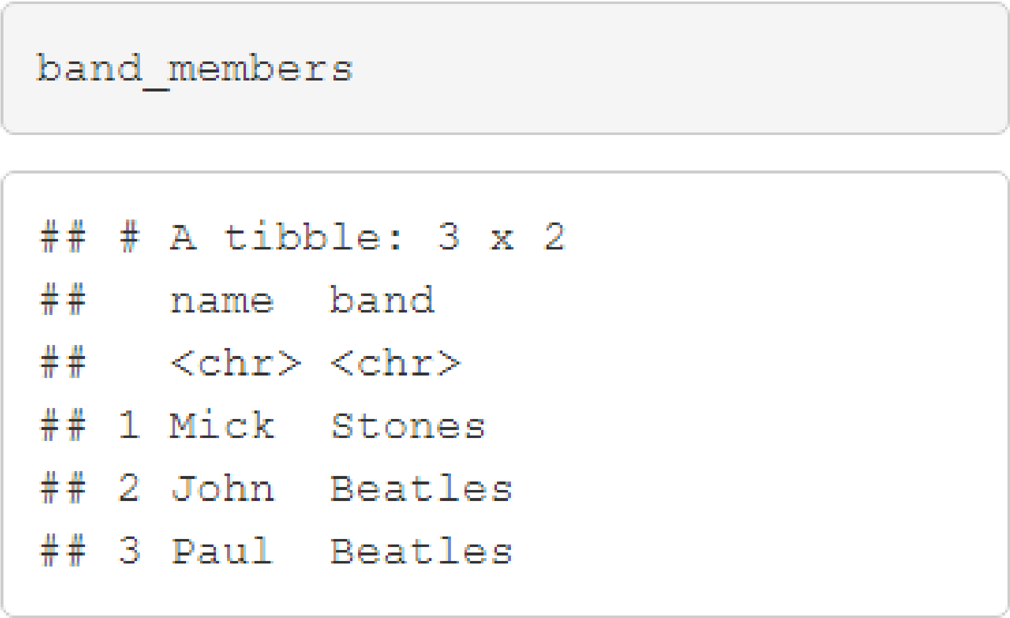
```
penguin_bill_lenghts_df %>%
  inner_join(latin_names_df)
```



```
## # A tibble: 344 x 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
```

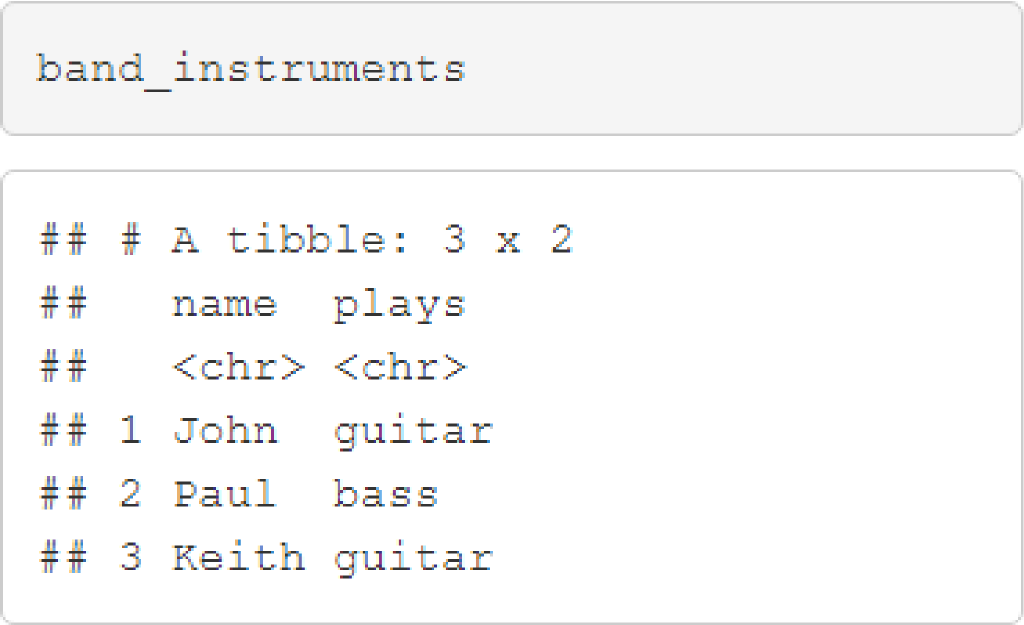
Types of join functions

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



```
band_members
```

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



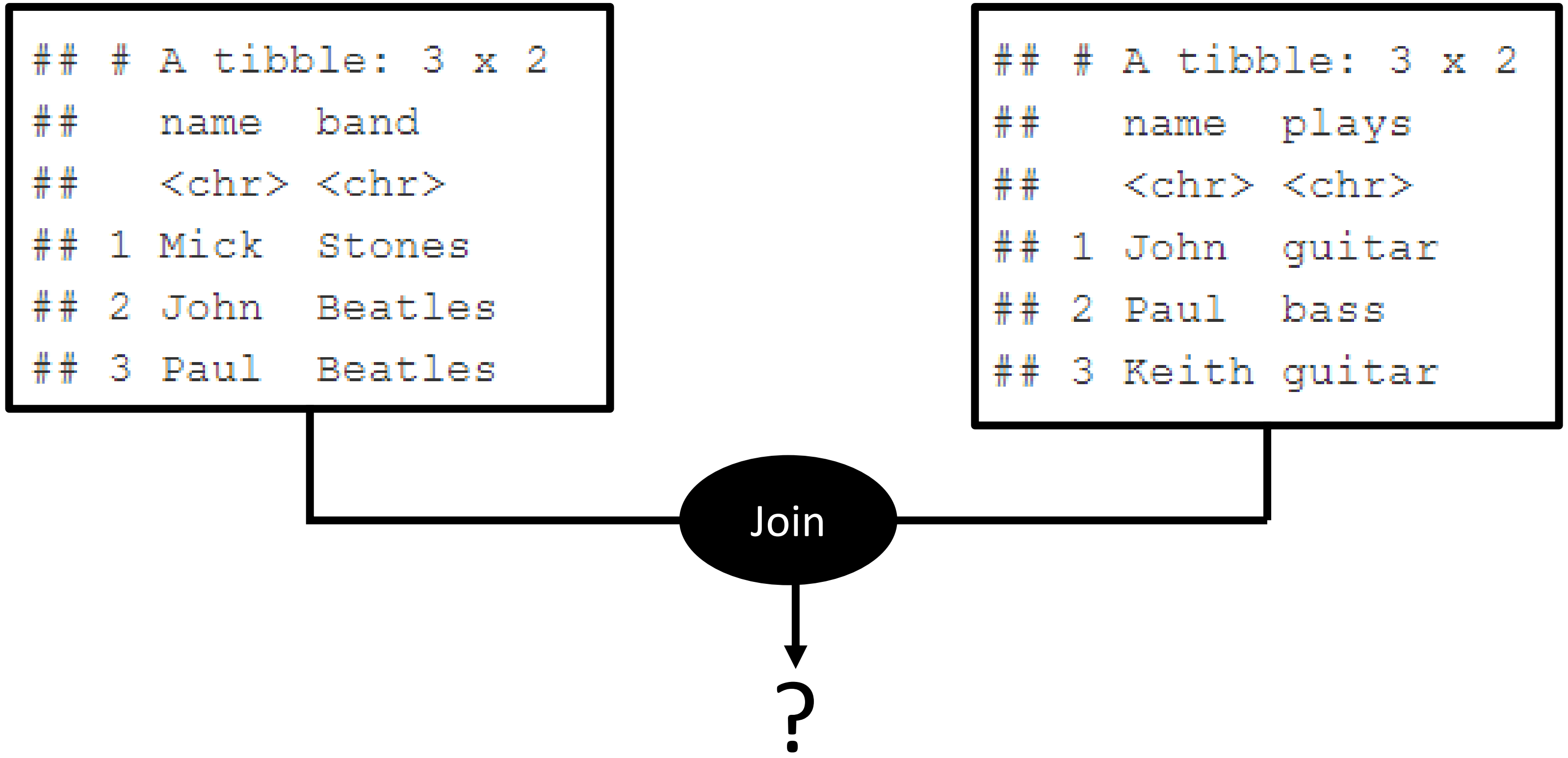
```
band_instruments
```

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

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

Types of join functions

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



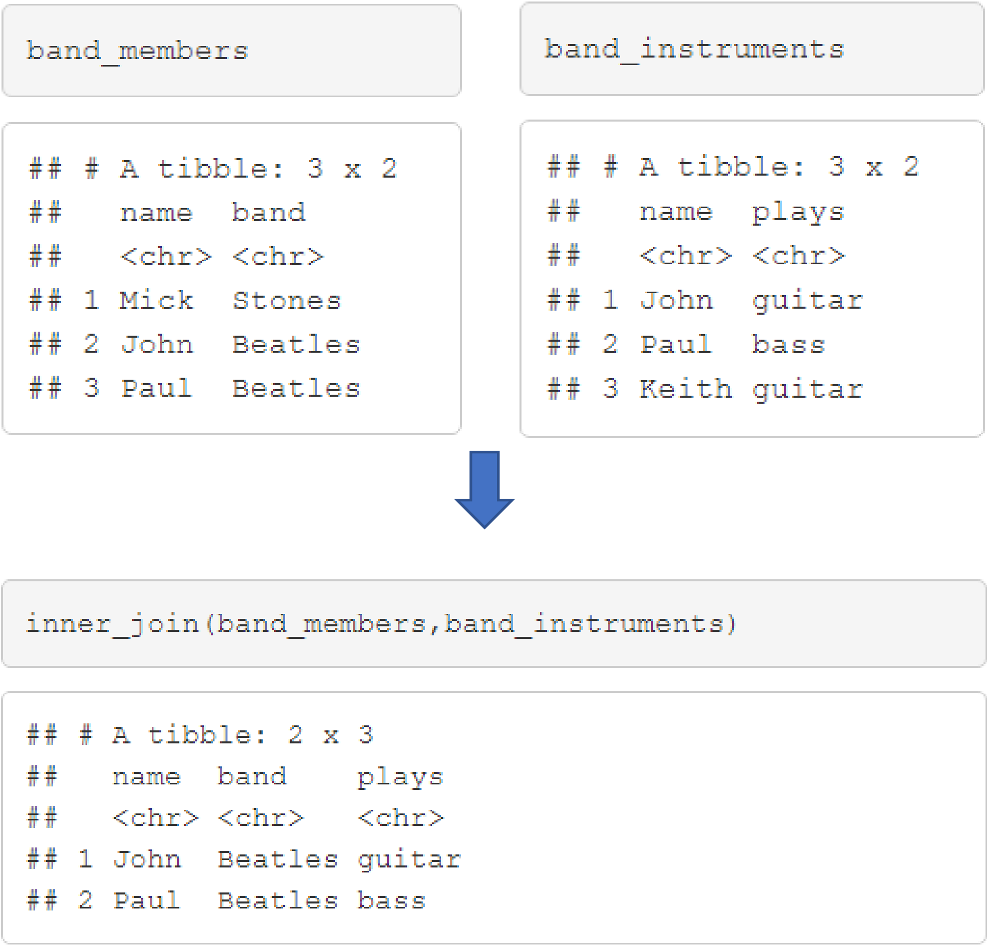
```
## # A tibble: 3 x 2
##   name  band
##   <chr> <chr>
## 1 Mick  Stones
## 2 John  Beatles
## 3 Paul  Beatles
```

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

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

Types of join functions

The inner join extracts the rows with a common entry in both tables.



```
band_members
```

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

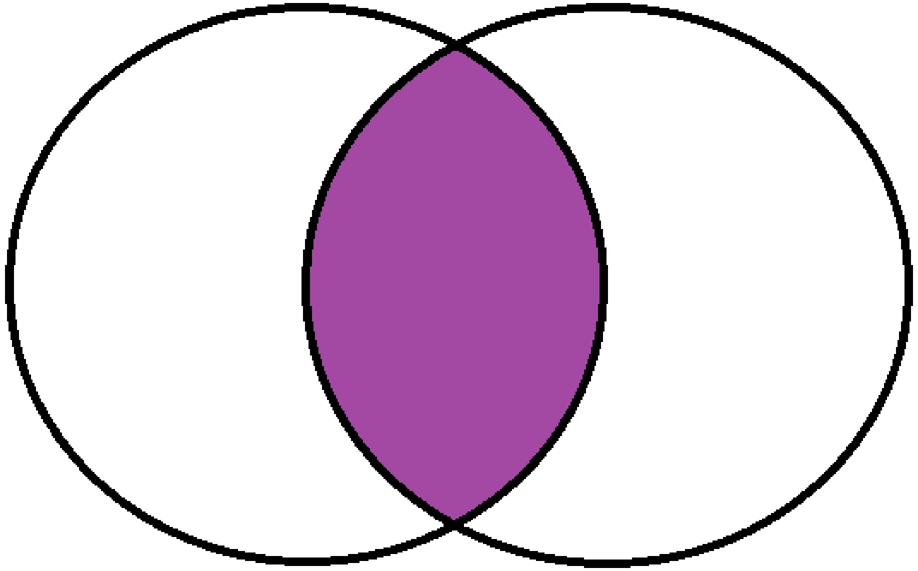
```
band_instruments
```

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

```
inner_join(band_members,band_instruments)
```

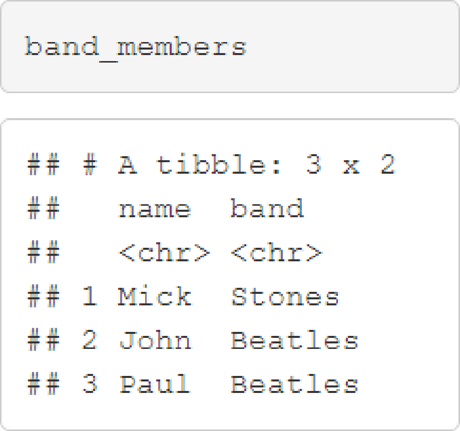
```
## # A tibble: 2 x 3
##   name  band    plays
##   <chr> <chr>   <chr>
## 1 John  Beatles guitar
## 2 Paul  Beatles bass
```

Inner join



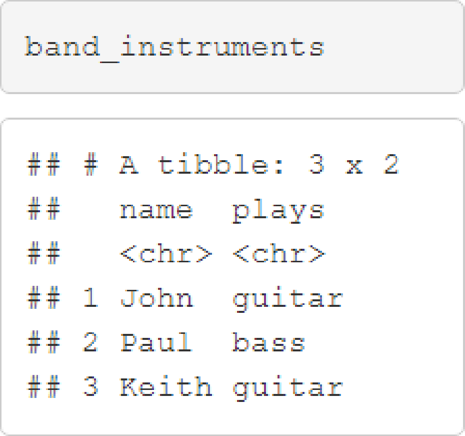
Types of join functions

The full join (also known as an outer join) extracts the rows with an entry in either tables.



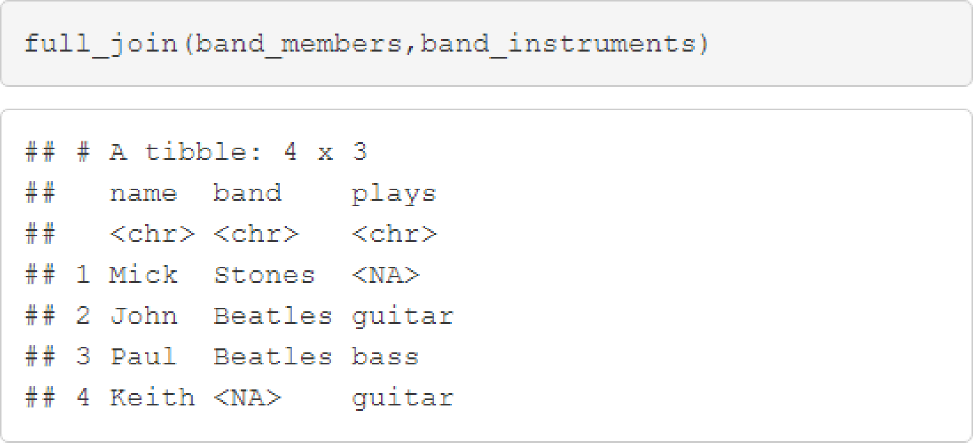
```
band_members
```

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



```
band_instruments
```

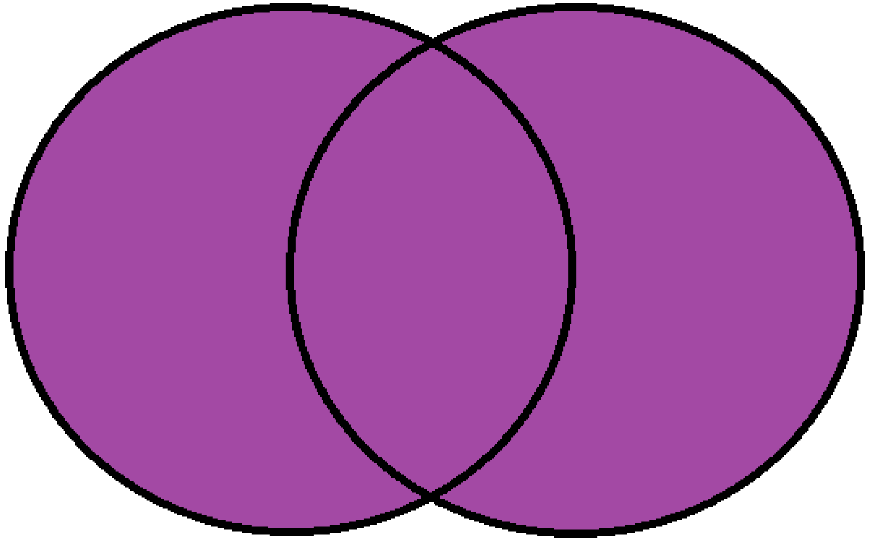
```
## # A tibble: 3 x 2
##   name  plays
##   <chr> <chr>
## 1 John  guitar
## 2 Paul  bass
## 3 Keith guitar
```



```
full_join(band_members,band_instruments)
```

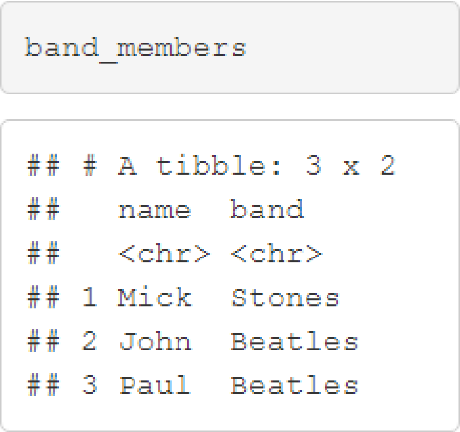
```
## # A tibble: 4 x 3
##   name  band    plays
##   <chr> <chr>   <chr>
## 1 Mick  Stones  <NA>
## 2 John  Beatles guitar
## 3 Paul  Beatles bass
## 4 Keith <NA>    guitar
```

Full join



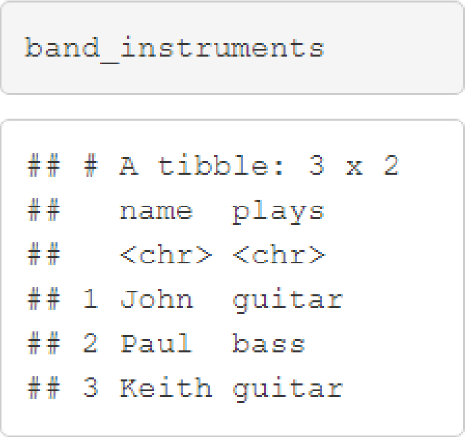
Types of join functions

The left join extracts the rows with an entry in the left table.



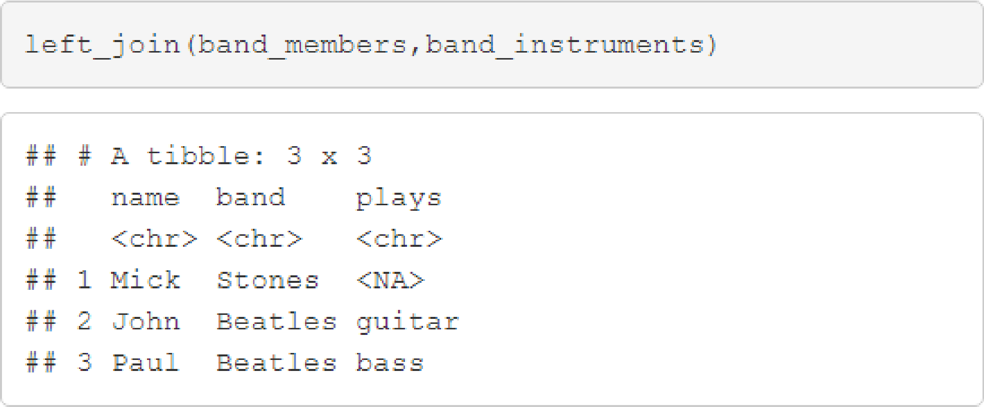
```
band_members
```

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



```
band_instruments
```

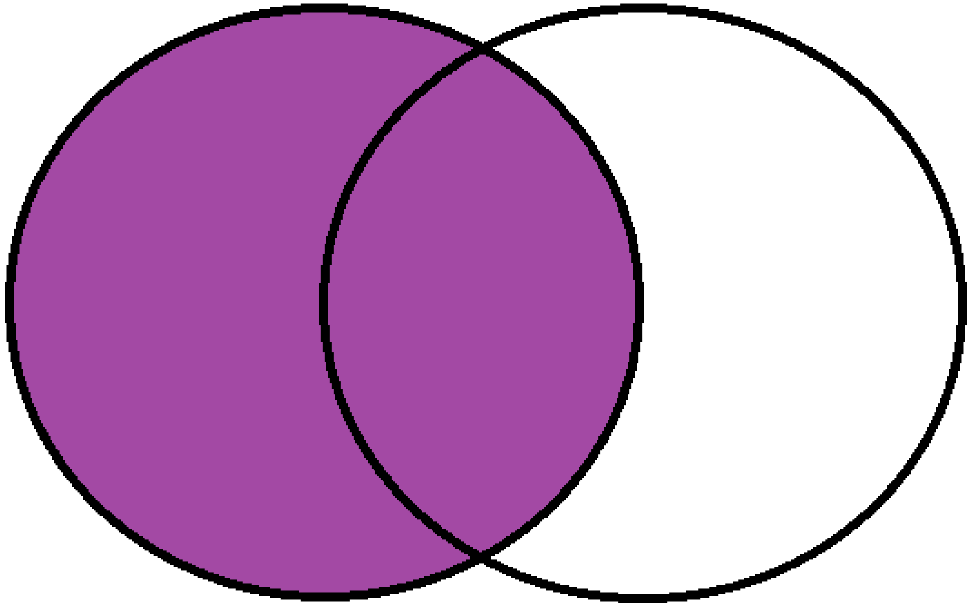
```
## # A tibble: 3 x 2
##   name  plays
##   <chr> <chr>
## 1 John  guitar
## 2 Paul  bass
## 3 Keith guitar
```



```
left_join(band_members,band_instruments)
```

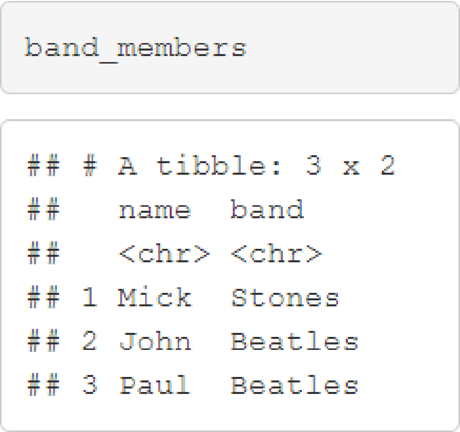
```
## # A tibble: 3 x 3
##   name  band    plays
##   <chr> <chr>   <chr>
## 1 Mick  Stones  <NA>
## 2 John  Beatles guitar
## 3 Paul  Beatles bass
```

Left join



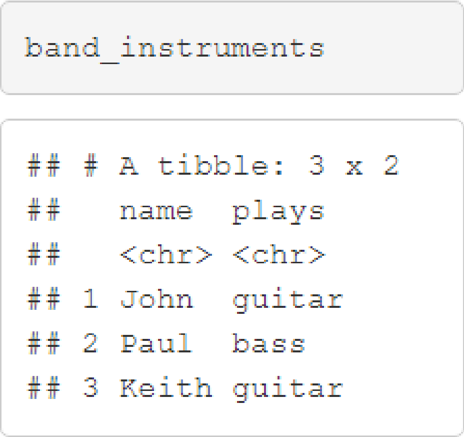
Types of join functions

The right join extracts the rows with an entry in the right table.



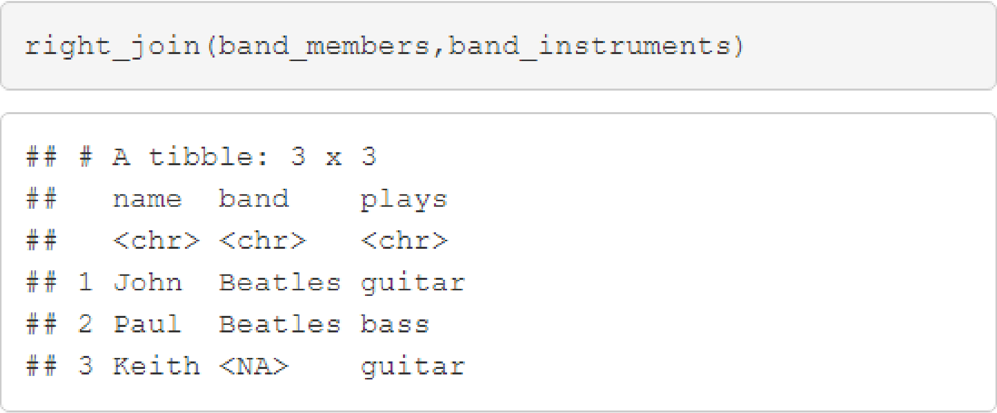
```
band_members
```

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



```
band_instruments
```

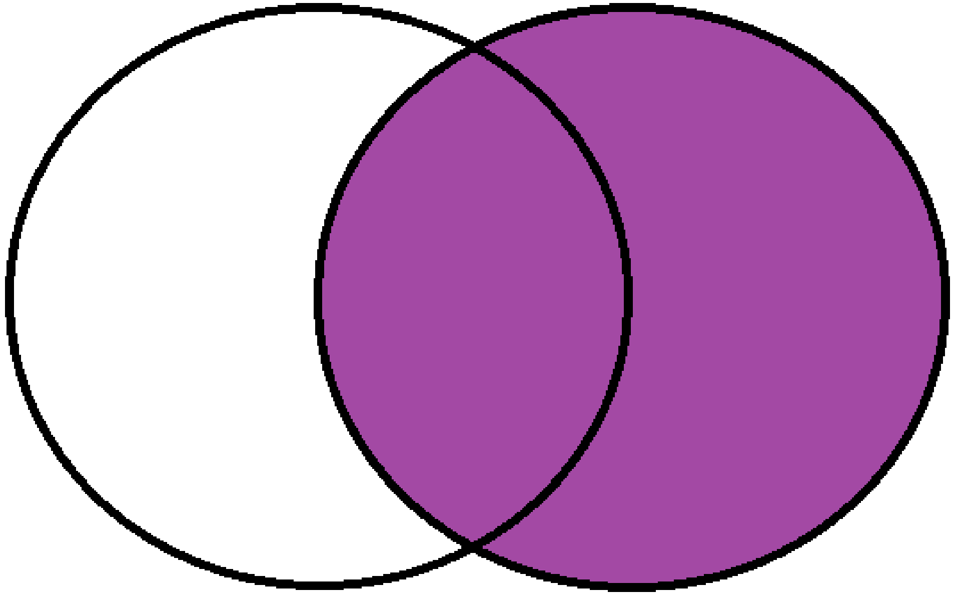
```
## # A tibble: 3 x 2
##   name  plays
##   <chr> <chr>
## 1 John  guitar
## 2 Paul  bass
## 3 Keith guitar
```



```
right_join(band_members,band_instruments)
```

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

Right join



What have we covered?

- We introduced the dplyr library for data wrangling.
- We saw how the select and filter functions allow us to extract sub-tables.
- We saw that the mutate function allows us to add new coloumns.
- We explored fast ways to get summary data frames with the summarize and groupby functions.
- We learnt how to fuse tables together with different types of join function.

Thanks for listening!

Henry W J Reeve

henry.reeve@bristol.ac.uk

Include EMATM0061 in the subject of your email.