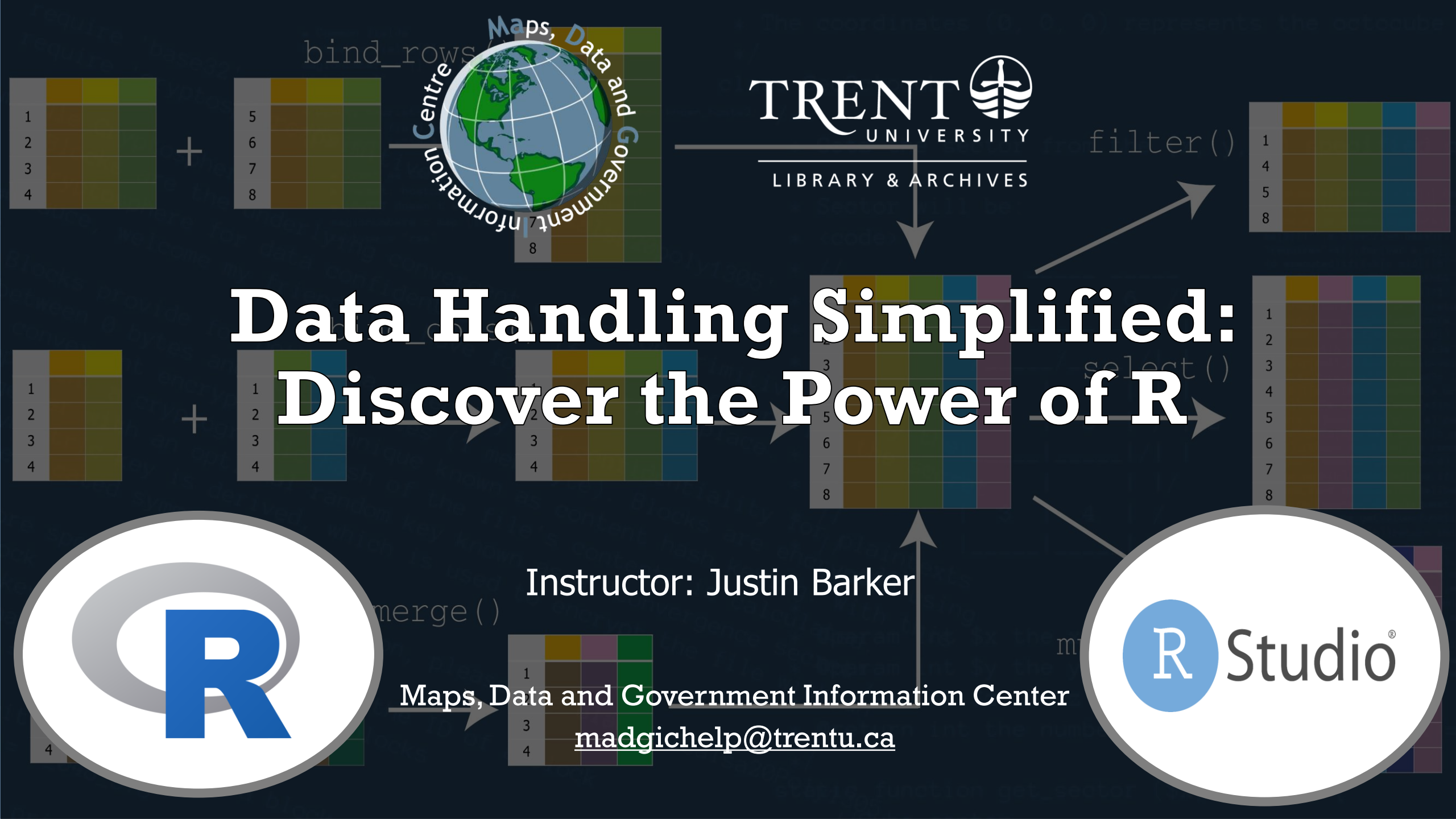


Data Handling Simplified: Discover the Power of R

Instructor: Justin Barker

Maps, Data and Government Information Center
madgichelp@trentu.ca

Workshop content

All content for this workshop can be found:
E:/DataHandlingR/

- Slides
- Cheat sheet
- Script
- Data for workshop

Feel free to make personal copies

Maps, Data and Government Information Centre

Collections Support

Data Visualization Lab

Advanced computers, spatial, statistical and visualization software

Bata Library 402

GIS, Statistical, and Data Visualization Support

ESRI / ArcGIS

esri

Research Data Management and Archiving

Research Collaborations

Regional Environmental History Atlas Project

Workshops and teaching support

Bata Library 415
8:30 am to 4:30 pm, Monday to Friday

madgichelp@trentu.ca
https://www.trentu.ca/library/madgic

R workflow

Import Data

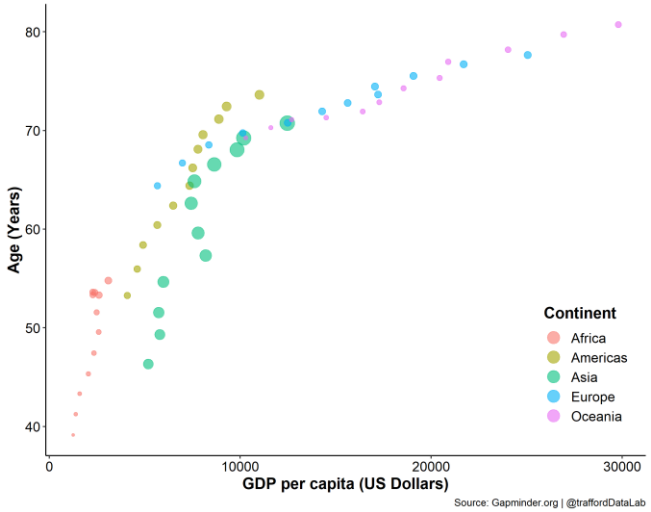
Data management

Explore

Transform

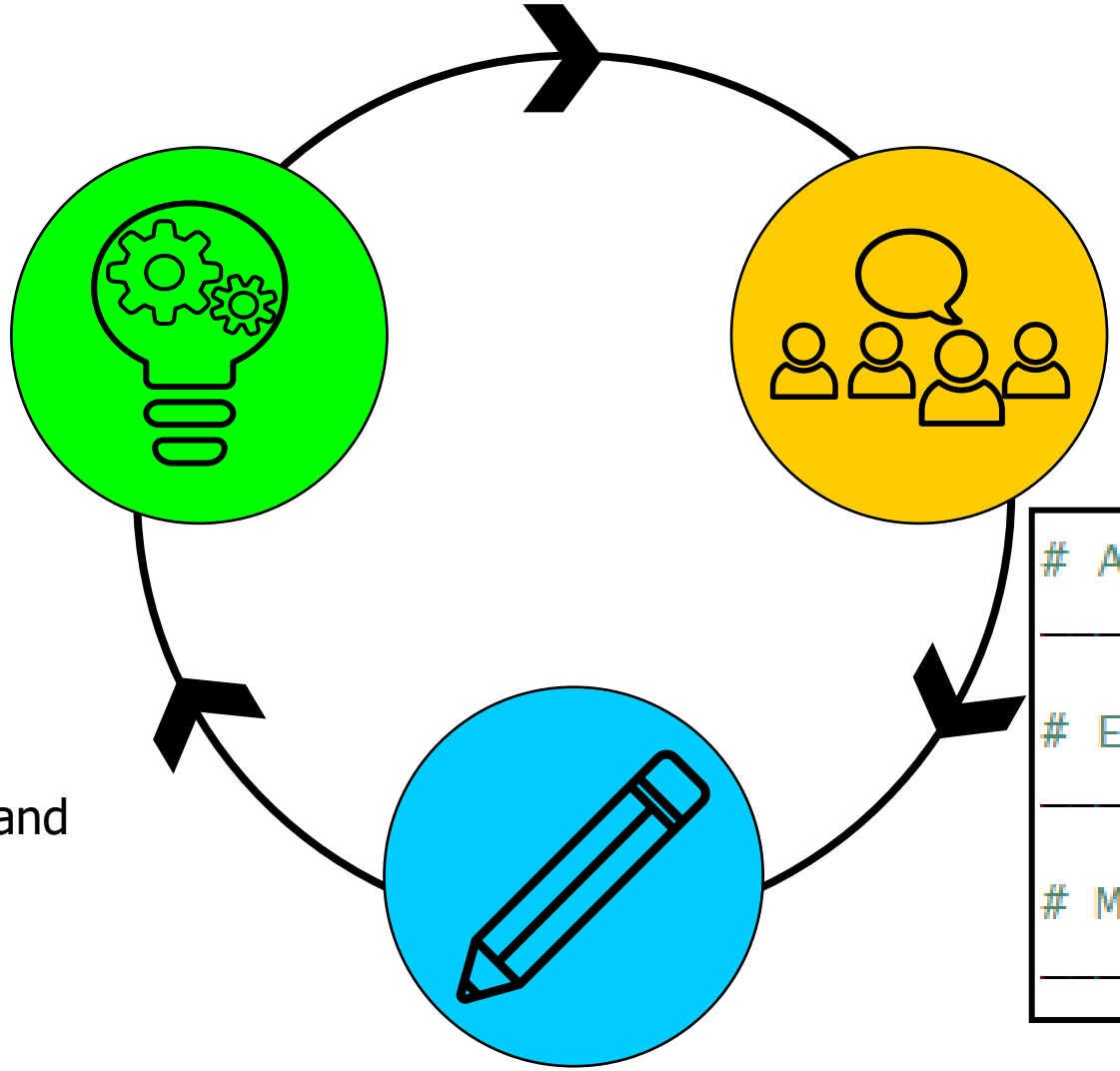
Visualize

Analyze



Communicate

Workshop structure



Learn
Short presentation of concepts

Practice
Guided walkthrough of R code

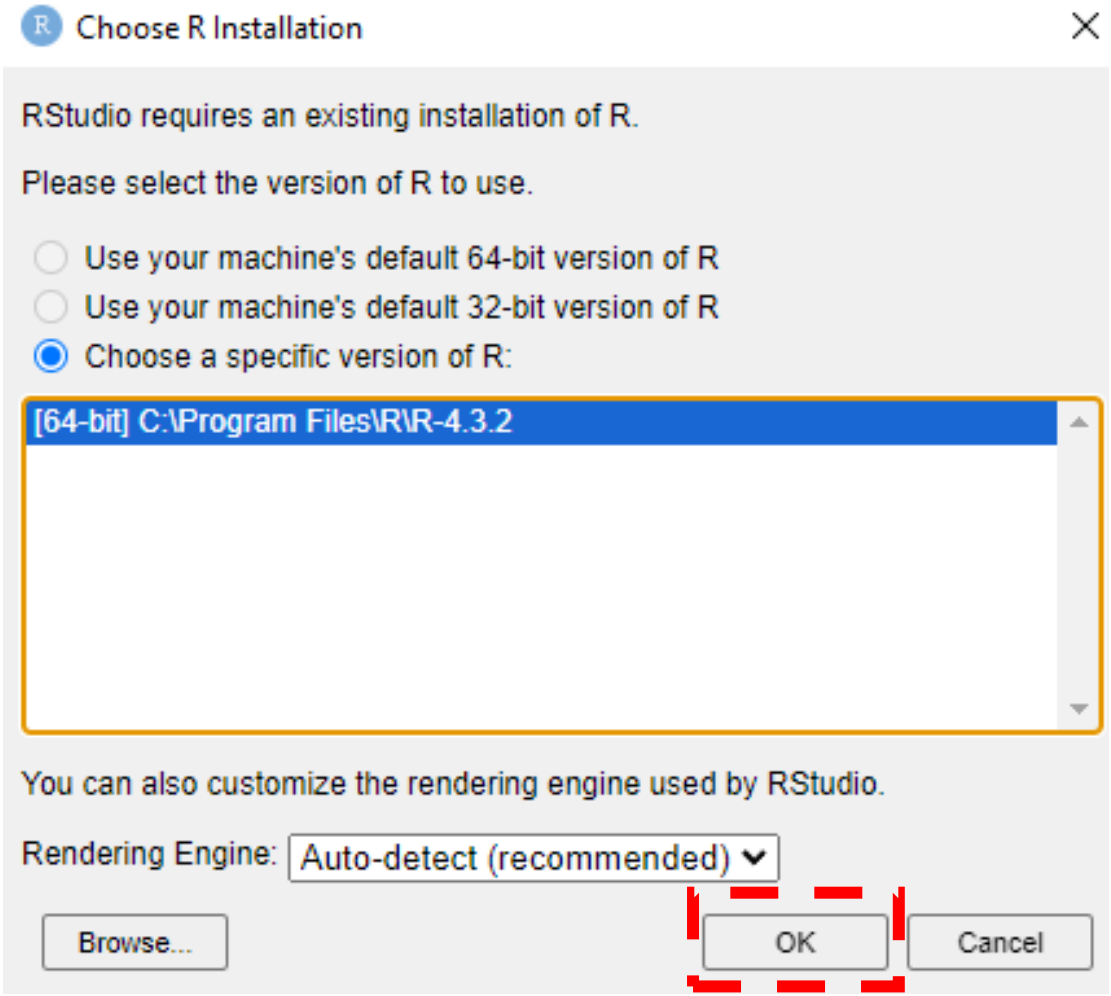
```
# A multiplication
_____

# Exponentiation
____

# More complex calculations: BEDMAS
_____ / _____
```

Follow along to fill in the blanks in the script

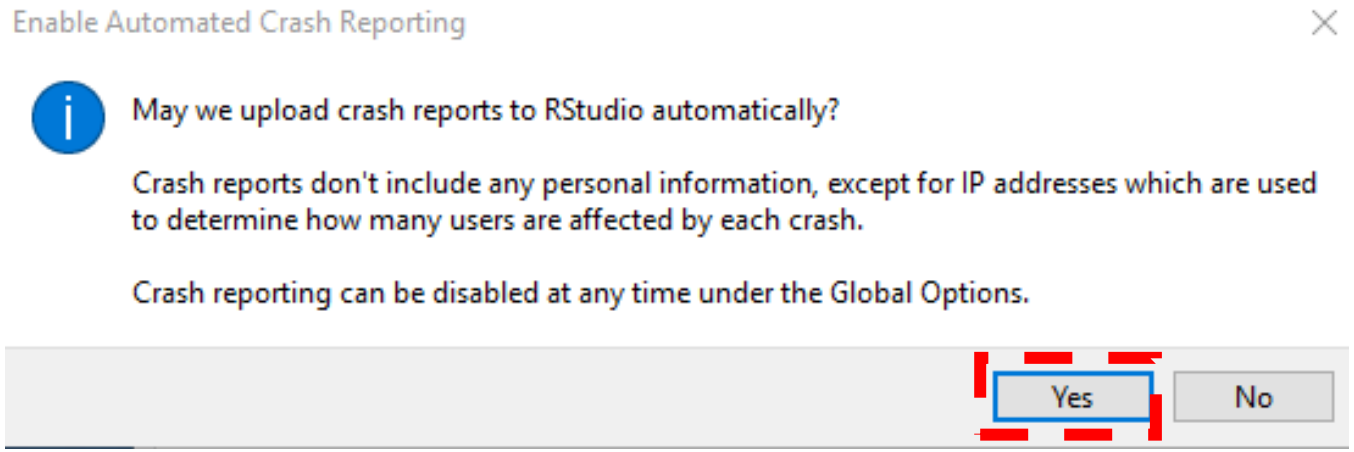
The main concepts and commands for this workshop are summarized in the provided handout.

Apply
Test your skills with a short challenge

Upon opening RStudio for the first time there will be two popups.



Accept the default R Installation



Allow R to upload crash reports

Please open the provided R script in RStudio

E:/DataHandlingR/ DataHandlingR.R

Within this file, please **fill in the blanks** to complete the R script.

```
# A multiplication
______

# Exponentiation
____

# More complex calculations: BEDMAS
_____ / _____
```

Follow along to fill in the blanks in the script

E:/DataHandlingR/ DataHandlingR_Filled.R

Within this file, the blanks are filled for you

```
# A multiplication
 3*5

# Exponentiation
 2^9

# More complex calculations: BEDMAS
 (100*5)/(4^16)
```

Review of R basics

- Run code in the source pane (text editor) with `Ctrl + Enter` or the 

button

- Any text to the left of at least one `#` is a comment. Comments are non-executable text. Use them to annotate the code
  - Four or five `#` provide headers or sub-headers in the script, respectively.
  - Use `Ctrl` + `Shift` + `C` to quickly (un)comment lines

- Create objects in R with the assignment operator, `<-`

```
> Spp_Ptbo_2021 <- 81
> Spp_Ptbo_2021
[1] 81
```

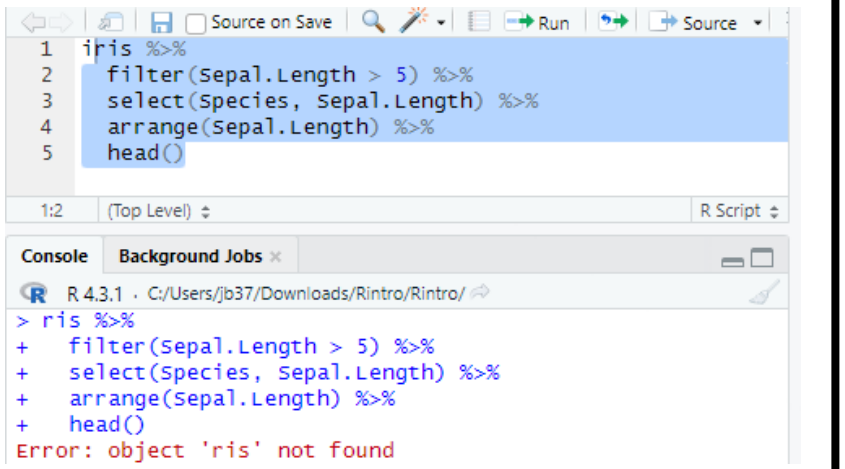
- Text data should always be input between `'` or `"`. Make sure text values are green!
  - This includes file paths

```
> read.csv("file.csv")
```

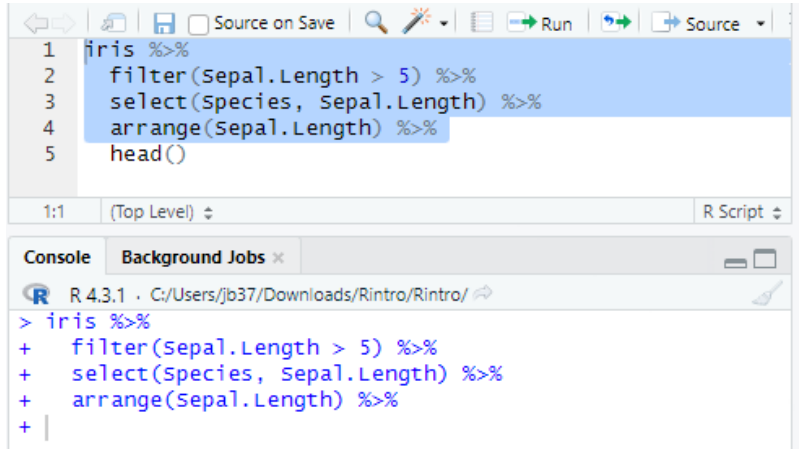
Caution: Highlighting to run

You can highlight part of a line, one line, or multiple lines.

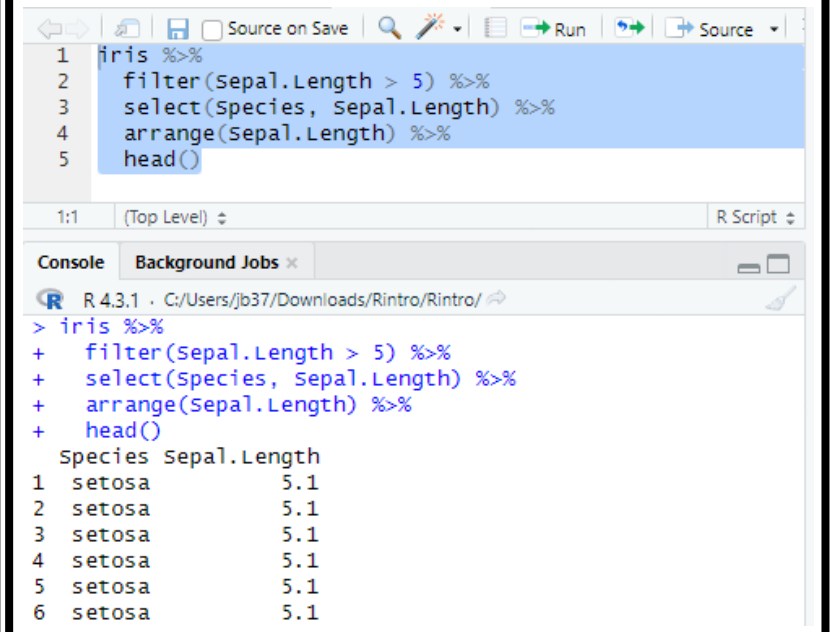
Be careful about what you are highlighting. Highlighting incomplete code may return an error or cause further issues.



Missing part of the beginning returns an error saying the data frame is not found



The last line was missed, R is waiting for more code. Run the last line or press esc



All lines highlighted were run in their entirety


Importing and
inspecting data

Audubon CHRISTMAS BIRD COUNT

The Christmas Bird Count is a yearly census of birds performed by volunteers, indicating species counts per region.

`Location`	Region where the count was conducted
`Year`	Year when the count was conducted
`Common_Name`	Common name of the species being reported
`Count`	Total number of observed individuals for the region and species.
`Flags`	Regional Editor flags (High count, low count, unusual count)

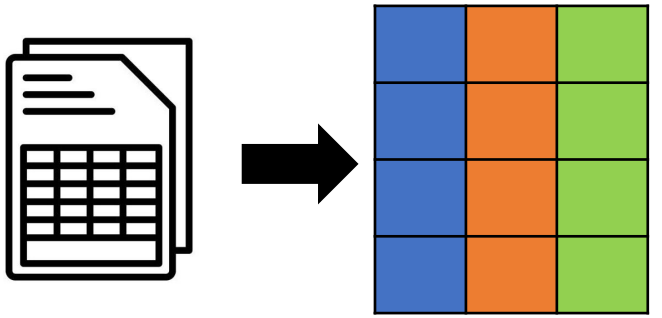
Importing Data

Most R work starts with bringing your data in.

R can read data from all common file types:

- Text (.txt)
- Comma separated values (.csv)

} Base R

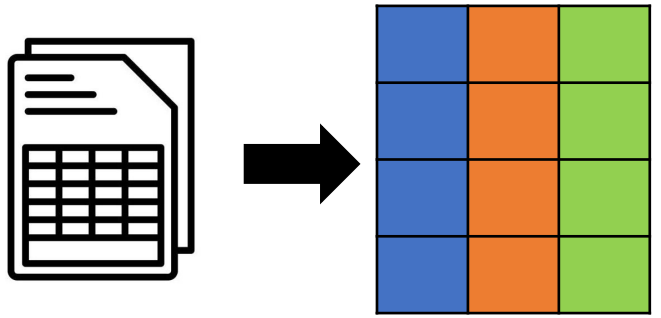


- Excel worksheet (.xlsx)
- SPSS dataset (.por)
- SAS data (.xpt)
- Stata (.dta)
- Raster (.tiff, .grd, .bil, .asc, .sdat, .rst, .nc, .envi, .img)
- Shapefiles (.shp)
- JSON (.json)
- SQL database queries

} Requires additional packages

Importing Data

Most R work starts with bringing your data in.



R can read data from all common file types:

- Text (.txt)
- Comma separated values (.csv)
- Excel worksheet (.xlsx)
- SPSS dataset (.por)
- SAS data (.xpt)
- Stata (.dta)
- Raster (.tiff, .grd, .bil, .asc, .sdat, .rst, .nc, .envi, .img)
- Shapefiles (.shp)
- JSON (.json)
- SQL database queries

See `readr` and `forgein` packages functions to import other file types

Formatting files before importing

Raw Spreadsheet:

	A	B	C	D	E	F	G	H
1	Name	100m	Long jump	Long jump	High.jump			
2	SEBRLE	NA	Disqualified	14.83	2.07		Mean	8.672938
3	CLAY	10.76	7.4	14.26	1.86		Median	9.2
4	KARPOV	11.02	7.3	14.77	2.04		Min	1.85

R interpretation:

	Name	x100m	Long.jump	Long.jump.1	High.jump	X	x.1	x.2
1	SEBRLE	NA	"Disqualified"	14.83	2.07	NA	Mean	8.672937
2	CLAY	10.76	"7.4"	14.26	1.86	NA	Median	9.200000
3	KARPOV	11.02	"7.3"	14.77	2.04	NA	Min	1.850000
4	BERNARD	11.02	"7.23"	14.25	1.92	NA	Max	16.360000

Start column names with a letter

Remove blank columns and rows

Do not duplicate names or use spaces

Remove notes or other unnecessary data

Use NA or leave cells blank for missing values

Importing functions

Text files (.txt):

```
> read.table(file = "file.txt", header = FALSE,
             sep = "\t")
```

`file`: file to be imported

`header`: file to be imported contains column names as the first line (TRUE) or not (FALSE)

`sep`: how columns are deliminated*

*Check your cheat sheet for different deliminator codes

Comma Separated Values (.csv):

```
> read.csv(file = "file.csv",
           header = TRUE)
```

Excel Worksheets (.xlsx):

```
> library(openxlsx)
> read.xlsx(xlsxFile = "file.xlsx", sheet = 1)
> read.xlsx(xlsxFile = "file.xlsx", sheet = "Sheet1")
```

`sheet`: Sheet number (first sheet is 1) or name

R environment data (.rda):

```
>load("file.rda")
```

R environment data loads all objects saved in a previous R session

Importing methods

- **`file.choose()`:** Allows user to select file from file explorer

```
> read.csv(file.choose())
```

- **`File path`:** Open a file by providing the full file path with forward slashes (/)

```
> read.csv("E:/Data/Data.csv")
```

- **`Working directory`**: Open files within a folder and sub-folder by setting the working directory

```
> setwd("E:/Data/")
> read.csv("Data.csv")
```

Inspect the data

Data should be inspected after importing to ensure it was imported correctly. This can be done through various functions:

Function	Returns
`head()`	Displays the first 6 rows or values
`tail()`	Displays the last 6 rows or values
`str()`	Displays the data frame structure (column names, data types, and first few values)
`View()`	Look at the whole data frame in a new window or tab
`names()`	Column names
`rownames()`	Row names
`ncol()`	Number of columns
`nrow()`	Number of rows
`summary()`	Basic statistics of each column

Note: These functions can be used with any data structure (vector, matrix, list, array)

Inspect the data: str()

For a quick inspection, `str()` is recommended.

```
> str(Ptbo_2020)
'data.frame':    104 obs. of  5 variables:
 $ Common_name  : chr  "Snow Goose" "Cackling Goose" "Canada Goose" "Wood Duck" ...
 $ Count        : int  NA NA 1505 NA 12 1251 NA NA NA NA ...
 $ Flag         : chr  NA NA NA NA ...
 $ Location_name: chr  "Peterborough" "Peterborough" "Peterborough" "Peterborough" ...
 $ Year         : int  2020 2020 2020 2020 2020 2020 2020 2020 2020 2020 ...
```

- Column names
- Column data types
- Data structure
- Number of rows (observations)
- Number of columns (variables)
- Shows first few values per column

`head()` or `tail()` are good simple alternatives to show the first or last six rows.

`glimpse()` is synonymous with `str()`

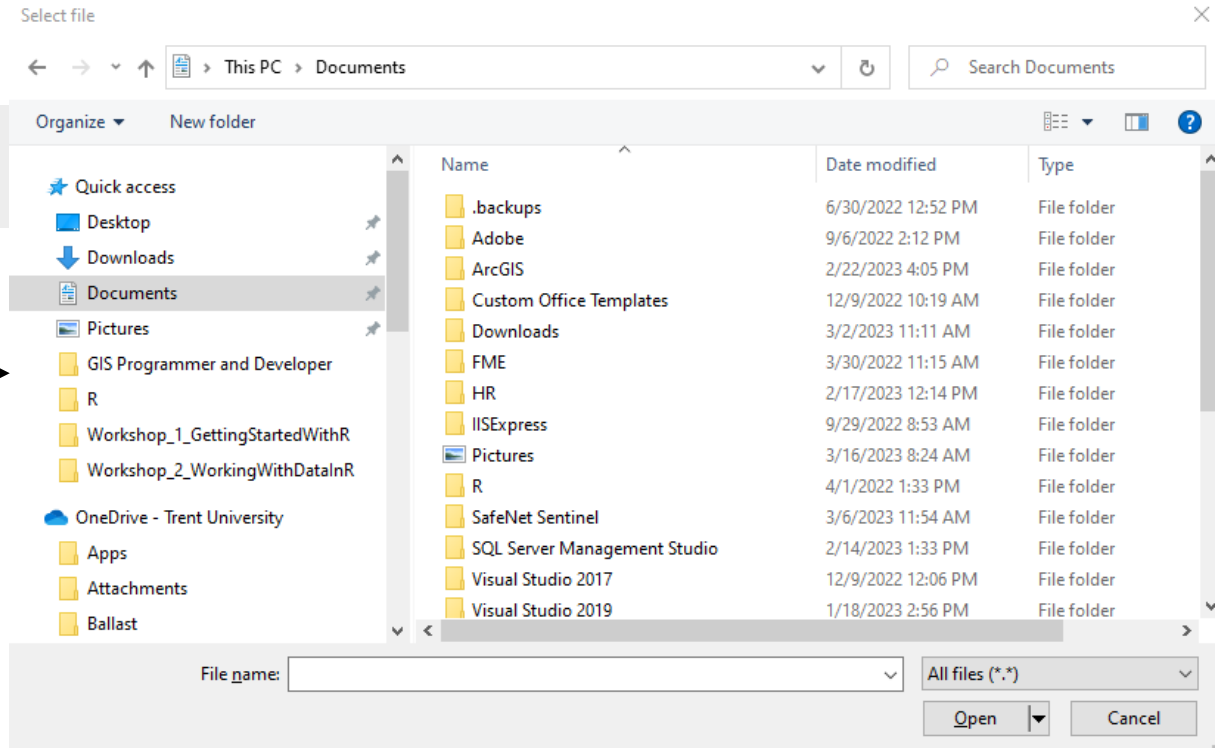
Let's Practice

Importing and inspecting data

Importing data: `file.choose()`

- **`file.choose()`:** Allows user to select file from file explorer

```
> read.csv(file.choose())
```



After running `file.choose()`, R will open file explorer. **You must select a file or cancel before running more code!**

Blanks:
Lines 30-32

```
FileChoose_CBC_2019 <- read.table(file = file.choose(),
                                  header = TRUE,
                                  sep = "\t")
```

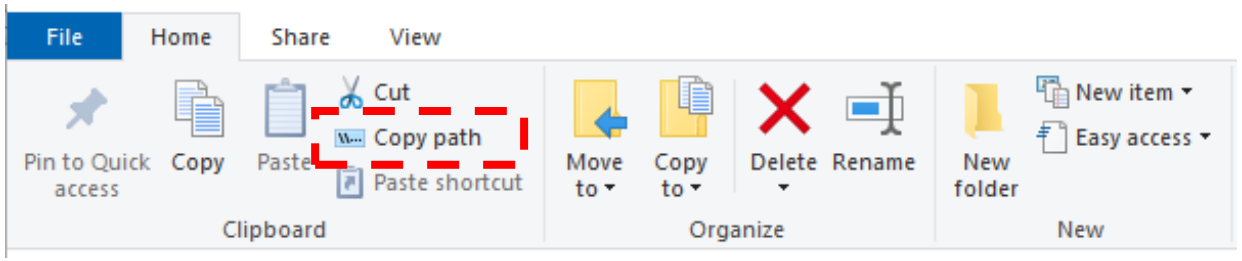
Importing data: File path

- **`File path:`** Open a file by providing the full file path with forward slashes (/)

```
> read.csv("E:/Data/Data.csv")
```

Windows:

1. Find and select the file in File Explorer.
2. Click the Copy path button



3. Paste into the appropriate line in R
4. Use Find and Replace to change all "\" with "/"

Mac:

1. Find and select the file in File Explorer.
2. Press `CMD + Option + C`
3. Paste into the appropriate line in R

Blanks

Lines

30 to 32:

```
FileChoose_CBC_2019 <- read.table(file = file.choose(),
                                  header = TRUE,
                                  sep = "\t")
```

63 to 66*:

```
FilePath_CBC_2019 <- read.table(file =
"E:/DataHandlingR/Data/ChristmasBirdCount_2019_SelectCities.txt",
                                sep = "\t",
                                header = TRUE)
```

* Only applicable if using a lab computer

Importing data: Working directory

- **`Working directory`**: Open files within a folder and sub-folder by setting the working directory

```
> setwd("E:/Data/")
> read.csv("Data.csv")
```

A **working directory** is the default location of any files that are imported from or exported to

Import data blanks

Lines

30-32:

```
FileChoose_CBC_2019 <- read.table(file = file.choose(),
                                  header = TRUE,
                                  sep = "\t")
```

63-66*:

* Only applicable if using a lab computer

```
FilePath_CBC_2019 <- read.table(file =
"E:/DataHandlingR/Data/ChristmasBirdCount_2019_SelectCities.txt",
                                sep = "\t",
                                header = TRUE)
```

94-96:

```
CBC_2019 <- read.table(file = "ChristmasBirdCount_2019_SelectCities.txt",
                       sep = "\t",
                       header = TRUE)
```

136-137:

```
CBC_2020 <- read.csv(file = "ChristmasBirdCount_2020_SelectCities.csv",
                     header = FALSE)
```

177-179:

```
CBC_2021 <- read.xlsx(xlsxFile = "ChristmasBirdCount_2019to2022_SelectCities.xlsx",
                      sheet = 3)
```

182-184:

```
CBC_2021 <- read.xlsx(xlsxFile = "ChristmasBirdCount_2019to2022_SelectCities.xlsx",
                      sheet = "Y2021")
```

Challenge 1

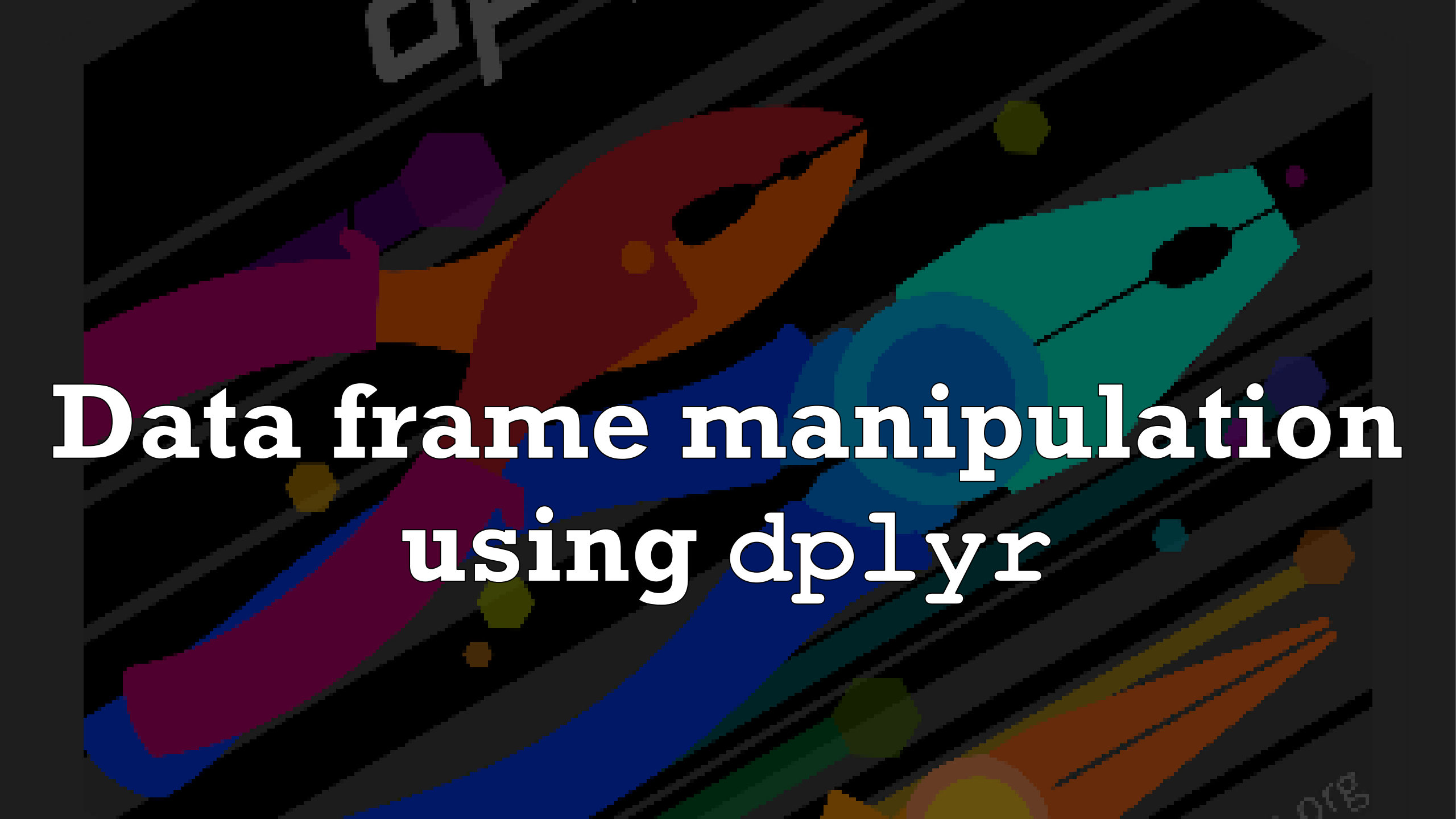
Open and examine *ChristmasBirdCount_2022_SelectCities.txt*,
Which of the following will successfully import it into R?

a)
```
Ptbo_2019 <- read.table("ChristmasBirdCount_2022_SelectCities.txt",
                        header = FALSE, sep="-")
```

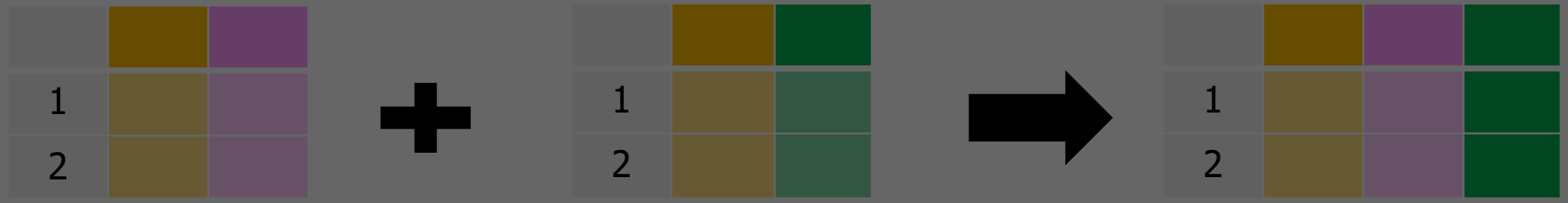
b)
```
Ptbo_2019 <- read.csv("ChristmasBirdCount_2022_SelectCities.txt",
                      header = FALSE)
```

c)
```
Ptbo_2019 <- read.xlsx("E:\ChristmasBirdCount_2022_SelectCities.txt",
                       sheetName = "Count_2019")
```

d)
```
Ptbo_2019 <- read.table("ChristmasBirdCount_2022_SelectCities.txt",
                        header = TRUE, sep="|")
```


Data frame manipulation
using dplyr

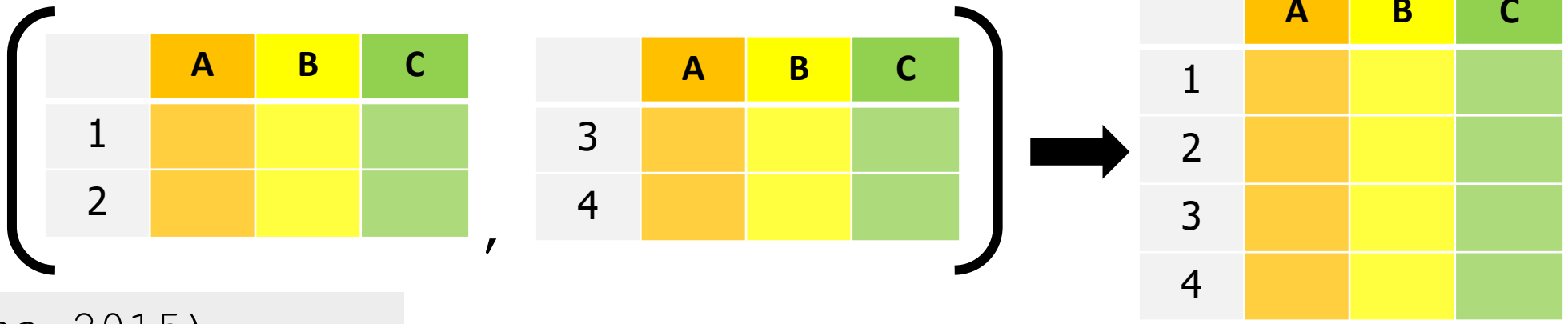
Combining data frames



Combine data frames by rows

Combine rows with the **same number of columns and names** `bind_rows()` :

bind_rows

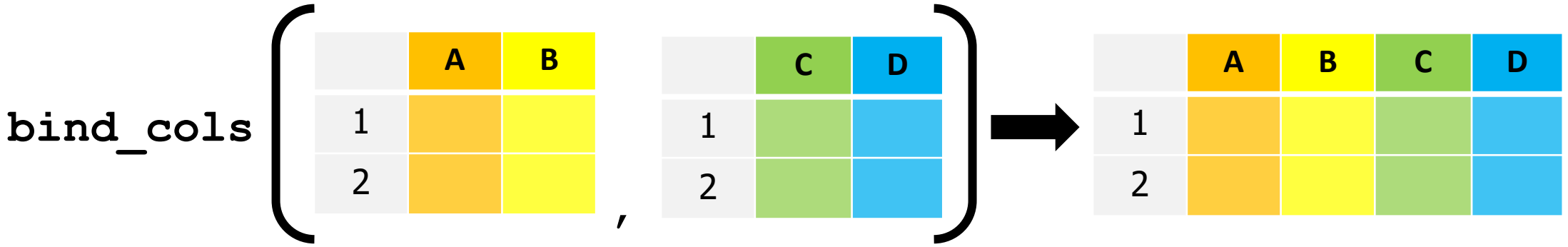



```
> head(Kawartha_2015)
          Common_Name Count Year
1       Canada Goose   293 2015
2 American Black Duck     1 2015
3            Mallard   382 2015
4      Greater Scaup     7 2015
> head(Kawartha_2016)
          Common_Name Count Year
1       Canada Goose    40 2016
2     Trumpeter Swan    19 2016
3 American Black Duck     2 2016
4            Mallard    26 2016
```

```
> Kawartha_2015_16 <- bind_rows(Kawartha
+ _2015, Kawartha_2016)
> head(Kawartha_2015_16)
          Common_Name Count Year
1       Canada Goose    40 2016
2     Trumpeter Swan    19 2016
3 American Black Duck     2 2016
4            Mallard    26 2016
5   Northern Pintail     1 2016
9          Bufflehead     5 2016
```

Combine data frames by column

Combine columns with the **same number of rows** with `bind_cols()` :

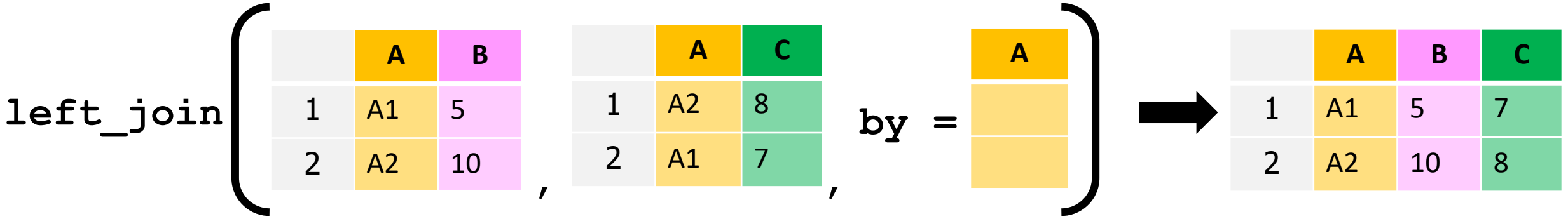



```
> head(Counts)
                  Name Y2015
1         Canada Goose   293
2 American Black Duck     1
3              Mallard   382
4        Greater Scaup     7
> head(More_Counts)
  Y2016  Y2017  Y2018
1    40      4      9
2     2      6      4
3    26     56     35
4     0      0      1
```

```
> Kawartha_counts <- bind_cols(Counts,
+ More_counts)
> head(Kawartha_2015_16)
                  Name Y2015 Y2016 Y2017 Y2018
1         Canada Goose   293    40     4     9
2 American Black Duck     1     2     6     4
3              Mallard   382    26    56    35
4        Greater Scaup     7     0     0     1
```

Combine data frames by value

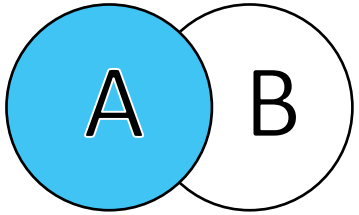
Combine columns with the **same values** with `left_join()` :




```
> head(Kawartha_2015)
                 Name Y2015
1        Canada Goose   293
2 American Black Duck     1
3             Mallard   382
4       Greater Scaup     7
> head(Kawartha_2016)
                 Name Y2016
1        Canada Goose    40
2 American Black Duck     2
3             Mallard    26
4       Greater Scaup     0
```

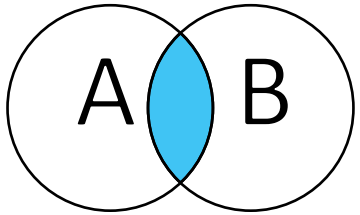
```
> Kawartha_15_16 <- left_join(
+ Kawartha_2015, Kawartha_2016,
+ by = "Name")
> head(Kawartha_2015_16)
                 Name Y2015 Y2016
1        Canada Goose   293    40
2 American Black Duck     1     2
3             Mallard   382    26
4       Greater Scaup     7     0
```

Joins



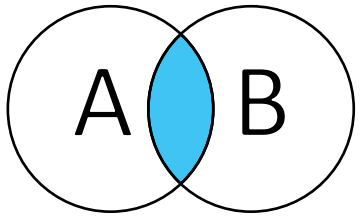

```
left_join(A, B, by = "x")
```

Join matching rows from B to A



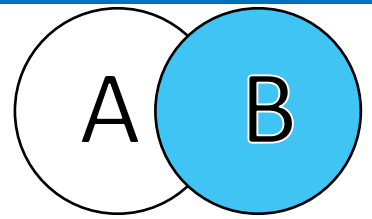

```
inner_join(A, B, by = "x")
```

Join all matching rows in A and B, including duplicates



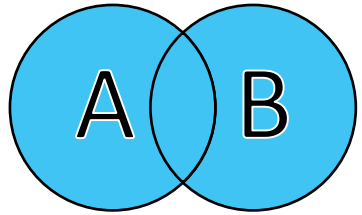

```
semi_join(A, B, by = "x")
```

All rows in A that have a match in B, without duplicates



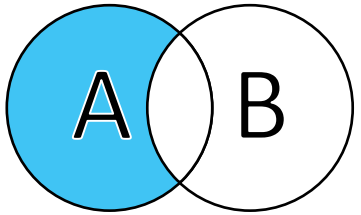

```
right_join(A, B, by = "x")
```

Join matching rows from A to B




```
full_join(A, B, by = "x")
```

Join data, retain all values, all rows




```
anti_join(A, B, by = "x")
```

All rows in A that do not have a match in B

Let's Practice

Combining data frames

Combining data frames blanks

Lines

294:
```
CBC_2019_2020 <- bind_rows(CBC_2020, CBC_2019)
```

347:
```
Birdspecific <- bind_cols(Birdspecific_1, Birdspecific_2)
```

376:
```
CBC_wScinames <- left_join(CBC_2019_2022, Scinames, by = "Common_Name")
```

Challenge 2

Given the AVONET data set, how would one add `AVONET` to `CBC_data_joined`?

```
> str(CBC_data_joined)
'data.frame':   3096 obs. of  7 variables:
 $ Location            : chr  "Guelph" "Guelph"
 $ Year                : num  2020 2020 2020 202
 $ Common_Name         : chr  "American Black Du
ald Eagle" ...
 $ Count               : num  48 499 239 5 10 ..
 $ Flags               : chr  NA NA NA NA ...
 $ Scientific_Name     : chr  "Anas rubripes" "C
us" "Haliaeetus leucocephalus" ...
 $ Num_Species_Reported: int  71 71 71 71 71 71
```

```
> str(AVONET)
'data.frame':   9993 obs. of  5 variables:
 $ Scientific_Name: chr  "Accipiter albogularis" "Accip
 $ Beak.Width     : num  10.6 8.8 9.2 8.9 8.7 6.6 8.3 8
 $ Beak.Depth     : num  14.7 11.6 13.5 11.9 11.1 12 10.
 $ Habitat        : chr  "Forest" "Shrubland" "Woodland"
 $ Trophic.Level  : chr  "Carnivore" "Carnivore" "Carniv
```

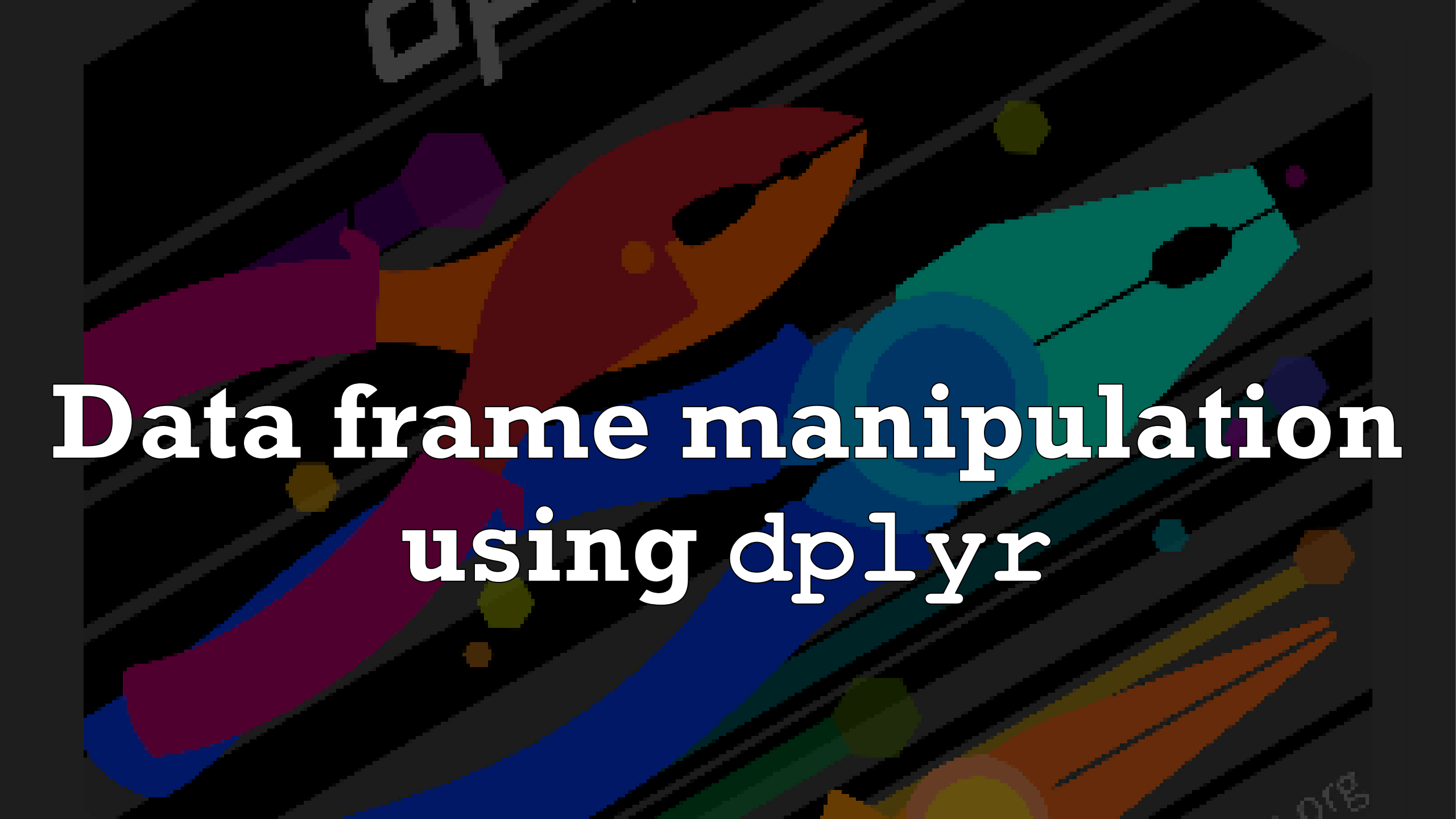
a) `bind_rows(CBC_Ptbo_wScinames, AVONET)`

b) `bind_cols(CBC_Ptbo_wScinames, AVONET)`

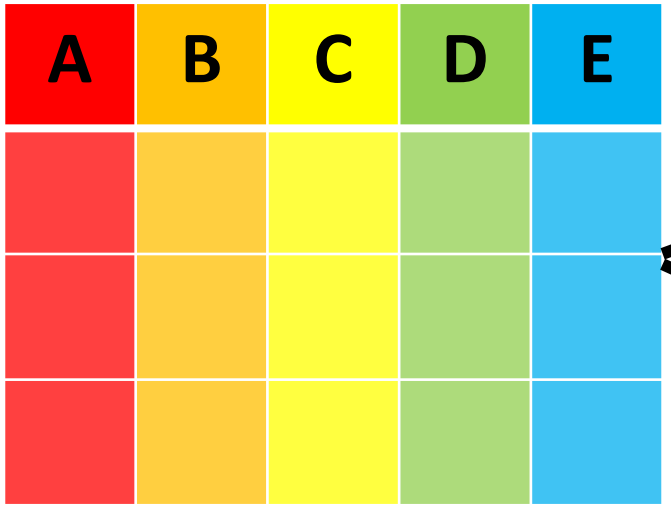
c) `left_join(CBC_data_joined, AVONET, by = "Scientific_name")`

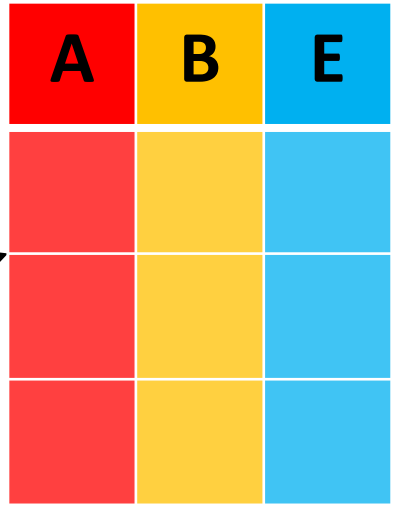
Break time

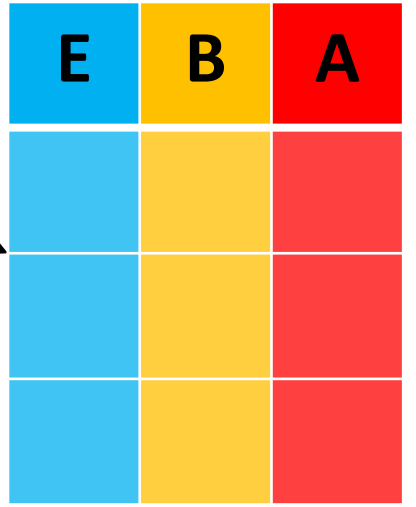
5 minutes


Data frame manipulation
using dplyr

Subset, re-arrange, and remove columns



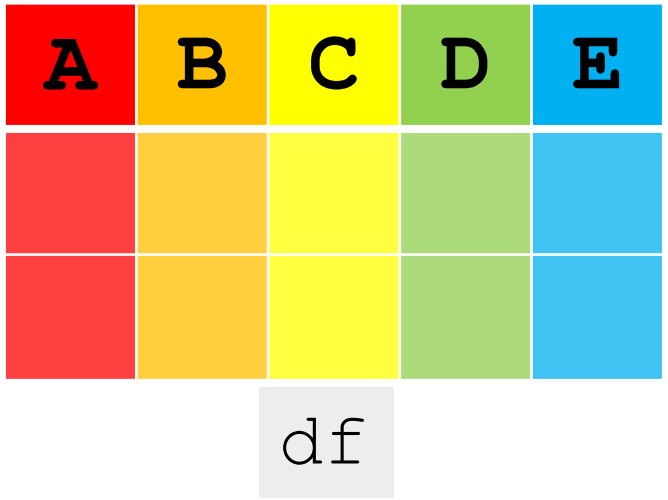




Subset columns

Columns can be subset by name or position using `select()`:

```
select(data_frame, <existing column(s)>)
```



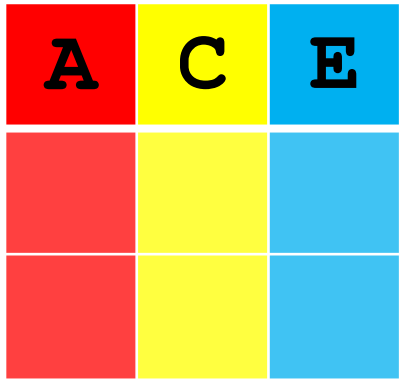
df

By name:

```
> df2 <- select(df, A, C, E)
```

By position:

```
> df2 <- select(df, 1, 3, 5)
```

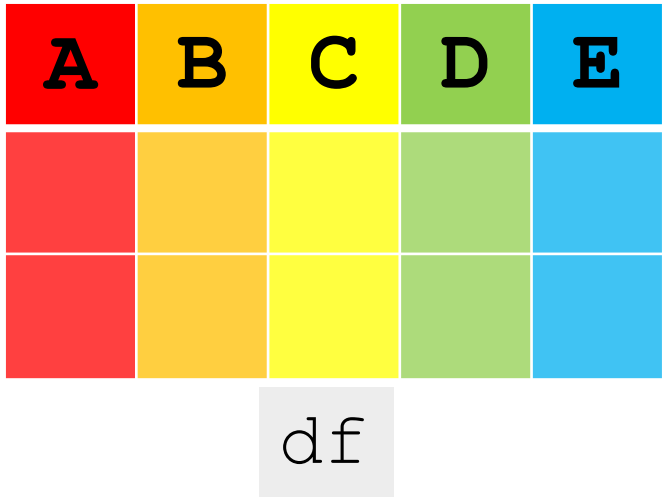


df2

Re-arrange columns

Changing column order in `select()` will return the order provided:

```
select(data_frame, <existing column(s)>)
```



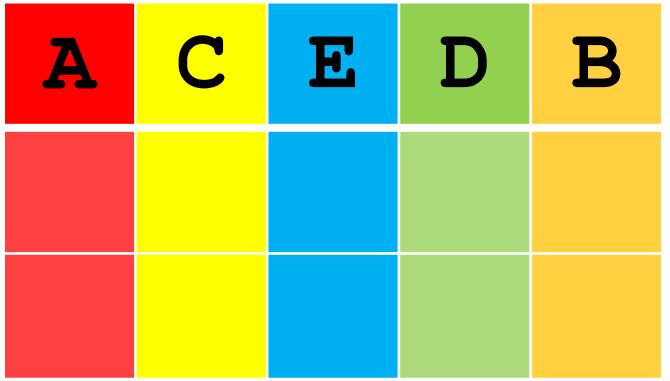
df

By name:

```
> df2 <- select(df, A,
+               C, E, D, B)
```

By position:

```
> df2 <- select(df, 1,
+                3, 5, 4, 2)
```

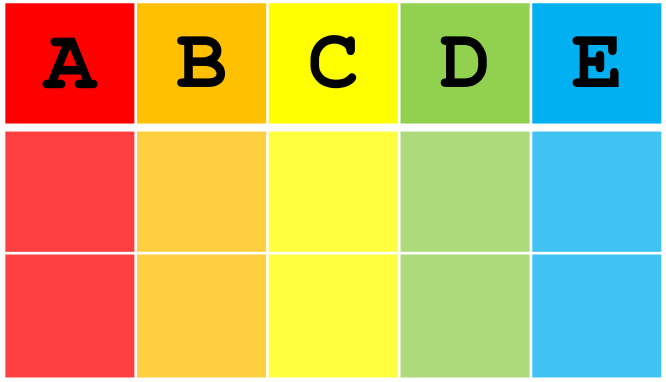


df2

Remove columns

Columns can be removed by placing a minus sign, `-`, before the column name or position in `select()`:

```
select(data_frame, -<existing column(s)>)
```



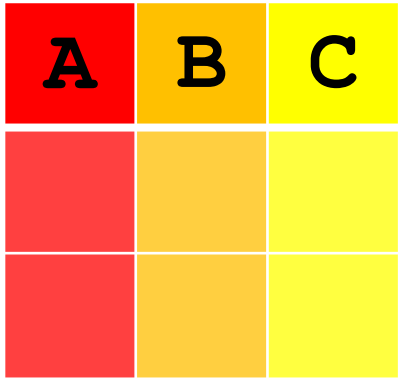
df

By name:

```
> df2 <- select(df, -D, -E)
```

By position:

```
> df2 <- select(df, -4, -5)
```



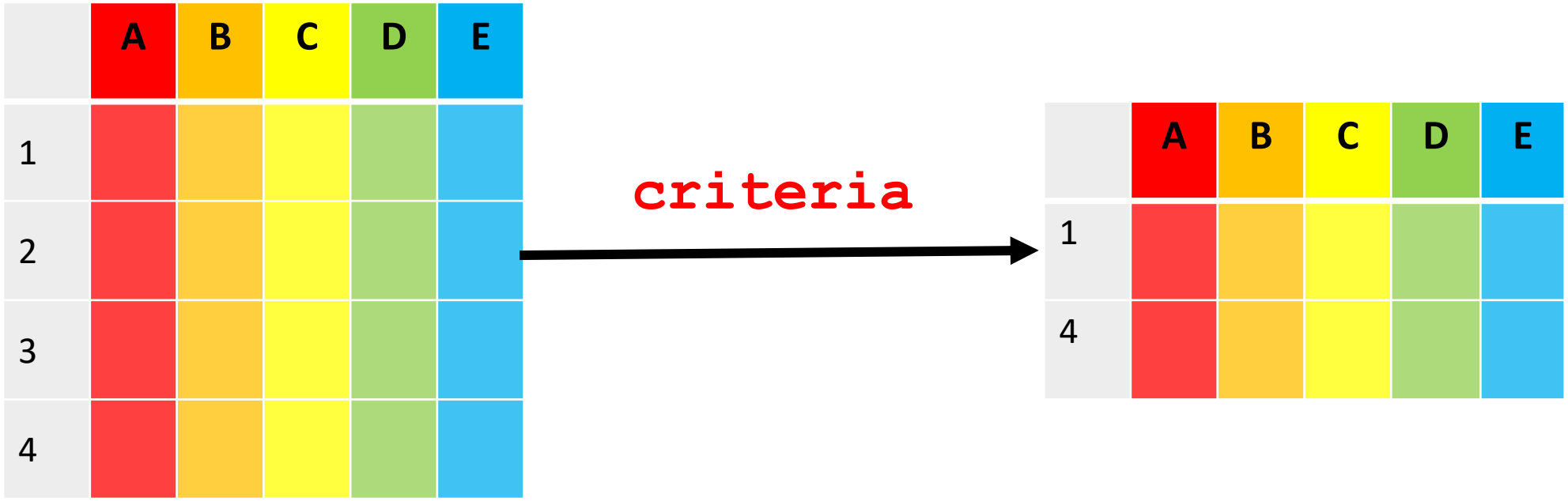
df2

Selection helpers

Columns can be selected based on a common pattern or criteria:

- `everything()` all columns, unless otherwise selected
- `last_col()` select the last column, possibly with an offset
- `starts_with("A")` columns that start with a specified pattern
- `ends_with("s")` columns that end with a specified pattern
- `contains("a")` columns that contain a specified pattern
- `:` select sequential columns (e.g. `Col_A:Col_J`)

Subset rows



Select rows by value: `filter()`

Rows may be isolated based on a criteria

```
filter(data_frame, col <criteria>)
```

Different data types require specific criteria.

All common data types can be filtered based on a few common operators:

Description	Operator
Equals	`==`
Not equal to	`!=`
Missing values	`is.na()`
Non-missing values	`!is.na()`
Any of	`X %in% c()`
Not any of	`!X %in% c()`

X represents a single column name

```
>head(TO_20, 3)
      Common_Name Count Flag
1   Canada Goose  1229 <NA>
2      Mute Swan   136 <NA>
3 Trumpeter Swan   102 <NA>
> filter(TO_20, Flag == "US")
         Common_Name Count Flag
1 American Woodcock     1   US
2      Palm Warbler     1   US
```

Select rows by numeric value

```
filter(data_frame, col <criteria>)
```

Numeric filters include:

Description	Operator
Greater than	`>`
Less than	`<`
Greater than or equal to	`>=`
Less than or equal to	`<=`
Between two numbers	`between(X, left, right)`

X represents a single column name

```
>head(TO_20, 3)
      Common_Name Count Flag
1   Canada Goose  1229 <NA>
2      Mute Swan   136 <NA>
3 Trumpeter Swan   102 <NA>
> filter(TO_20, Count <= 10)
         Common_Name Count Flag
1          Wood Duck     7 <NA>
2  Northern Pintail     1 <NA>
3         Canvasback     3 <NA>
4  Ring-necked Duck     1 <NA>
5     Merganser sp.     8 <NA>
6           Duck sp.     3 <NA>
7        Wild Turkey     3 <NA>
8        Common Loon     3 <NA>
```

Select rows by character value

Filtering character values is made easier through the `stringr` package:

Description	Operator
Contains	`str_detect(X,"s")`
Begins with	`str_detect(X, "^s")`
Ends with	`str_detect(X, "s$")`
Contains digits	`str_detect(X, "\\d")`

X represents a single column name
"s" represents a string

```
>head(TO_20, 2)
      Common_Name Count Flag
1   Canada Goose  1229 <NA>
2      Mute Swan   136 <NA>
> filter(TO_20, str_detect(Common_Name, "^B"))
      Common_Name Count Flag
1    Black Scoter    NA <NA>
2      Bufflehead   616 <NA>
3      Bald Eagle     1 <NA>
```

Filter by multiple criteria: AND

Multiple criteria can be applied using the AND (`&`) or OR (`|`) operators:

- **AND**: All criteria must be met
- OR: One of the criteria must be met

```
filter(data_frame, col <criteria>)
```

```
> filter(TO_20, str_detect(Common_Name, "d$") &
+            Count < 1000)
          Common_Name Count Flag
1             Redhead   825 <NA>
2           Bufflehead   616 <NA>
3        Gray Catbird     3 <NA>
4 Northern Mockingbird    21 <NA>
5 Red-winged Blackbird     1 <NA>
6 Brown-headed Cowbird     3 <NA>
```

Common names that end in "d" ***AND*** *have counts less than 1000*

Filter by multiple criteria: OR

Multiple criteria can be applied using the AND (`&`) or OR (`|`) operators

- AND: All criteria must be met
- **OR**: One of the criteria must be met

```
filter(data_frame, col <criteria>)
```

```
> filter(TO_20, str_detect(Common_Name, "d$") |
+                  Count < 1000)
                Common_Name Count Flag
1                 Mute Swan   136 <NA>
2           Trumpeter Swan   102 <NA>
3                  Mallard  2713 <NA>
4         Northern Pintail     1 <NA>
5                Canvasback     3 <NA>
6                  Redhead   825 <NA>
...
```

Add a **|** with Shift + \

*Common names that end in "d" **OR** have counts less than 1000*

Sort rows: arrange()

Sort rows by one or more columns in ascending order (A – Z, 0 – 100):

```
arrange(data_frame, <by>)
```

Sort by a single column

```
> arrange(TO_20, Count)
                 Common_Name Count Flag
1          Northern Pintail     1 <NA>
2          Ring-necked Duck     1 <NA>
3          Red-necked Grebe     1 <NA>
4                Bald Eagle     1 <NA>
5      Red-shouldered Hawk      1 <NA>
6              American Coot     1 <NA>
7          Purple Sandpiper     1 <NA>
8        American Woodcock      1   US
```

Sort by multiple columns

```
> arrange(TO_20, Count,
+             Common_Name)
                 Common_Name Count Flag
1              American Coot     1 <NA>
2        American Woodcock       1   US
3                Bald Eagle     1 <NA>
4    Downy/Hairy Woodpecker      1 <NA>
5                Fox Sparrow     1 <NA>
6              Glaucous Gull     1 <NA>
7          Northern Pintail      1 <NA>
8     Northern Saw-whet Owl      1 <NA>
```

Sorts by Count THEN Common_Name

Sort rows in descending order

Sort rows by one or more columns in descending order (Z – A, 100 – 0) by using `desc()` around each desired column:

```
arrange(data_frame, desc(<by>))
```

Sort by a single column

```
> arrange(TO_20, desc(Count))
                 Common_Name Count
1                    Mallard  2713
2           European Starling  2703
3            Long-tailed Duck  2370
4                House Sparrow  2012
5     Black-capped Chickadee  1475
6                Greater Scaup  1365
7            Ring-billed Gull  1353
8                 Canada Goose  1229
```

Sort by multiple columns

```
> arrange(TO_20, desc(Count),
+     desc(Common_Name))
                  Common_Name Count
1     White-winged Crossbill     1
2                  Snowy Owl     1
3          Slaty-backed Gull     1
4            Short-eared Owl     1
5           Ring-necked Duck     1
6        Red-winged Blackbird     1
7         Red-shouldered Hawk     1
8            Red-necked Grebe     1
```

Only Common_Name is arranged in descending order

Let's Practice

Subsets and sorting

Subsets and sorting blanks

Lines

453-457:

```
columm_select <- select(CBC_wAvonet,
                        Year, Locaiton, Common_Name, Scientific_Name, Habitat,
                        Trophic.Level, Beak.Width, Count, Flags)
```

488-489:

```
filter_numeric <- filter(CBC_data, # Data frame to subset rows from
                         Count >=  5)
```

529-530:

```
filter_and <- filter(CBC_data,
                           Count < 5 & !is.na(Flags))
```

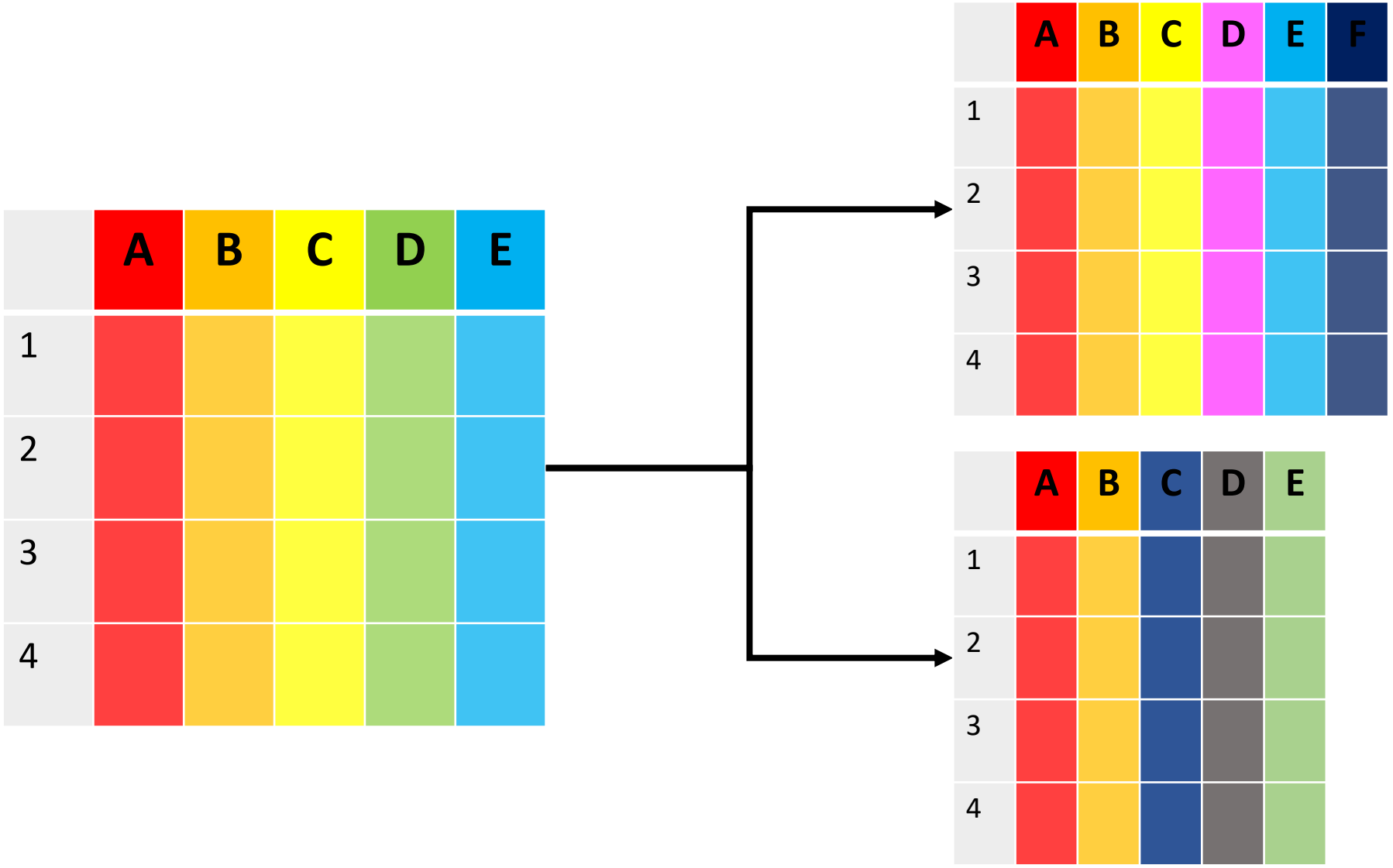
Challenge 3

1. Using the three-step method demonstrated in the script, determine how many omnivore (`Tropic.Level == "Omnivore"`) birds were found in Peterborough's 2020 count.

2. Present the results in a simplified data frame showing only scientific name, common name, and count.

3. Order the rows by decreasing counts.

Add or alter columns



Creating new columns: `mutate()`

Create new columns from existing data:

```
mutate(data_frame, new_col = function(existing_col))
```

```
> head(CBC_Barrie, 4)
         Common_Name  Y2019 Y2020
1       Accipiter sp.      1     1
2 American Black Duck     56    75
```

```
> mutate(CBC_Barrie, Count_Change = Y2020 - Y2019)
             Common_Name Y2019 Y2020 Count_Change
1          Accipiter sp.      1     1            0
2 American Black Duck        56    75           19
...
```

Alter column values: `mutate()`

If a name of an existing column is given in `mutate()`, that column is altered:

```
mutate(data_frame, existing_col = function(existing_col))
```

```
> head(CBC_Barrie, 4)
          Common_Name Y2019 Y2020
1       Accipiter sp.     1     1
2 American Black Duck    56    75
…
```

```
> mutate(CBC_Barrie, Y2019 = Y2019 + 10)
            Common_Name Y2019 Y2020
1       Accipiter sp.     11     1
2 American Black Duck     66    75
...
```

Replace values

Use `mutate()` with `ifelse()` to change specified values:

```
mutate(data_frame, existing_col = function(existing_col))
```

```
> head(CBC_Barrie, 4)
            Common_Name Y2019 Y2020
1         Accipiter sp.     1     1
2   American Black Duck    56    75
3         American Coot    NA    NA
4         American Crow   109    99
5       American Robin     NA    NA
```

```
> mutate(CBC_Barrie, Y2019 = ifelse(is.na(Y2019), 0, Y2019))
            Common_Name Y2019 Y2020
1         Accipiter sp.     1     1
2   American Black Duck    56    75
3         American Coot     0    NA
4         American Crow   109    99
5       American Robin      0    NA
```

If Y2019 is NA, then replace with 0, else return original Y2019 value

Specify where the new column goes

When creating a new column with `mutate()` the column will be placed on the end (right) of the data frame. Use the arguments `.before` or `.after` to specify a different location.

```
> mutate(CBC_Barrie, Total = Y2019 + Y2020, .before = Y2019)
             Common_Name Total Y2019 Y2020
1          Accipiter sp.     2     1     1
2   American Black Duck    131    56    75
3          American Coot     0     0     0
…
```

```
> mutate(CBC_Barrie, Total = Y2019 + Y2020,.after = Common_Name)
             Common_Name Total Y2019 Y2020
1          Accipiter sp.     2     1     1
2   American Black Duck    131    56    75
3          American Coot     0     0     0
…
```

Let's Practice

Add or alter columns

Add or alter columns blanks

Lines

638-640:

```
mutate_newcol0 <- mutate(CBC_data_arranged,
                         Common_Name_CAPS = toupper(Common_Name))
```

647-650:

```
mutate_newcol1 <- mutate(CBC_data_arranged,
                                Beak.Width_cm = Beak.Width/10,
                                .after = Beak.Width)
```

Challenge 4

Using `mutate()`, which code would create a new column, `Count_Sq`, representing the `Count` value squared (Hint: `Count^2`) to the right of `Count`?

a)
```
> mutate(CBC_data_cleaned, Count_Sq = Count^2)
```

b)
```
mutate(CBC_data_cleaned, Count_Sq = Count^2,
       .before = Count)
```

c)
```
mutate(CBC_data_cleaned, Count_Sq = Count^2,
       .after = Count)
```

Break time

5 minutes

Streamline data manipulation: Piping

Piping %>%

Conducting multiple functions on a data frame requires creating new data frames or overwriting the original.

```
> CBC_Barrie <- filter(CBC_data, Location =
+                        "Barrie")
> CBC_Barrie_Dif <- mutate(CBC_Barrie, Difference =
+                            Y2019 - Y2020)
```

Piping allows multiple functions to be computed in sequence from a single data set.

```
> CBC_Barrie_Dif <- CBC_data %>%
+      filter(Location = "Barrie") %>%
+      mutate(Difference = Y2019 - Y2020)
```

Piping framework

Previously applying functions followed the framework:

```
function(data, arguments)
```

```
> filter(CBC_data, Location = "Barrie")
```

Piping rewrites this framework as:

```
data %>% function(arguments)
```

```
> CBC_data %>% filter(Location = "Barrie")
```

Quickly add `%>%` with:

- `Ctrl` + `Shift` + `M` on Windows
- `Cmd` + `Shift` + `M` on Mac

Writing pipelines

The pipe operator, `%>%`, is put at the END of each dataset or function.
The output of each function is then applied to the next function.

```
> CBC_data %>%
+ filter(Location == "Guelph")%>%
+ select(Year, Count, Flag) %>%
+ arrange(Year, desc(Count))
```

Data frame created by `filter()` input into `select()`

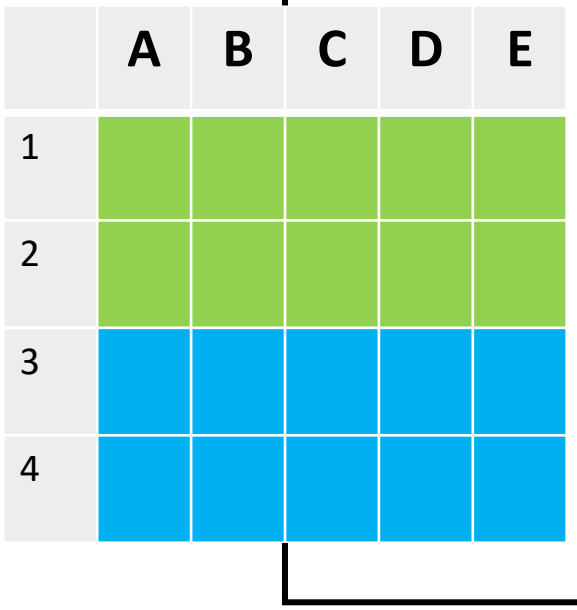
Data frame created by `select()` input into `arrange()`

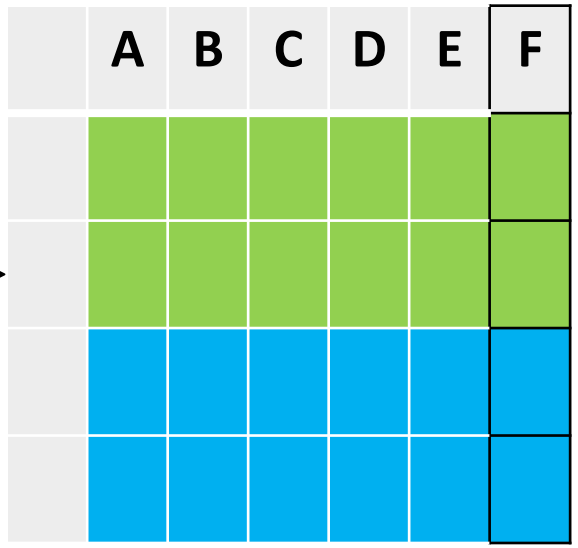
End the pipeline by **not** adding a `%>%`

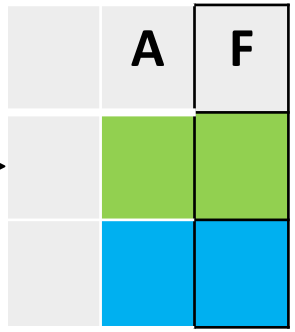
Piping improves code readability, limits the need to create or overwrite objects, and reduces user commands.

Aggregating data

	A	B	C	D	E
1					
2					
3					
4					



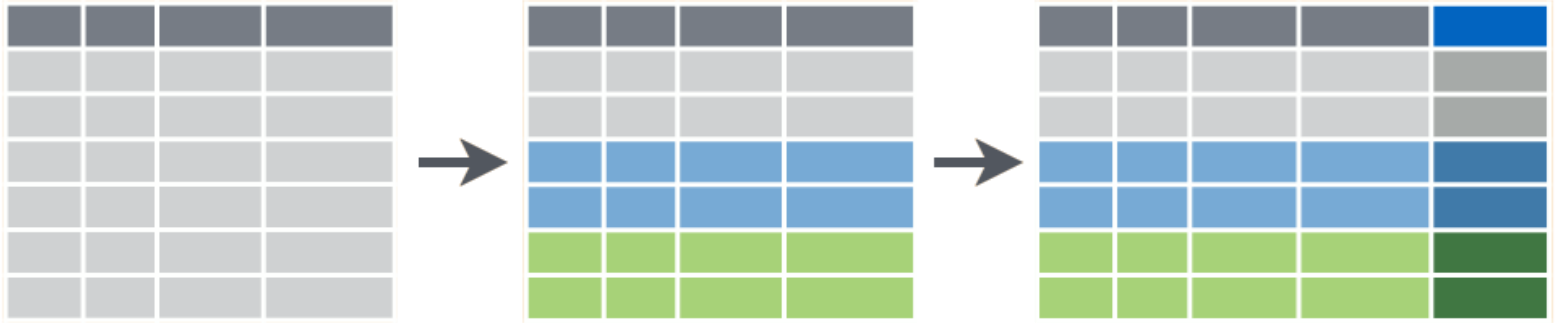




Aggregating data

Piping allows for quick calculations of values per group with `group_by()` followed by either `mutate()` or `summarise()`.

```
> data %>%
+     group_by(…) %>%
+     mutate(…)
```



```
> data %>%
+     group_by(…) %>%
+     summarise(…)
```

Aggregating data: group_by() %>% mutate()

```
> data %>%
+    group_by(...) %>%
+    mutate(…)
```

Computes new values by group for each row (observation) within the group.

```
> head(CBC_Data, 3)
  Location Year Common_Name Count   Flag
1 Peterborough 2020 Cooper's Hawk    9   <NA>
2 Peterborough 2017  Cooper's Hawk  12    HC
3 Peterborough 2019  Cooper's Hawk   7   <NA>
> CBC_Data %>%
+    group_by(Location, Year) %>%
+    mutate(N_Spp = n())
  Location Year Common_Name Count   Flag  N_Spp
1 Peterborough 2020 Cooper's Hawk    9   <NA>    54
2 Peterborough 2017  Cooper's Hawk  12    HC     51
3 Peterborough 2019  Cooper's Hawk   7   <NA>    47
…
```

Aggregating data: group_by() %>% summarise()

```
> data %>%
+     group_by(...) %>%
+     summarise(…)
```

Computes a summary for each group and removes all unconsidered data and individual rows (observations).

```
> head(CBC_Data, 3)
  Location Year Common_Name Count   Flag
1 Peterborough 2020 Cooper's Hawk    9   <NA>
2 Peterborough 2017  Cooper's Hawk  12    HC
3 Peterborough 2019  Cooper's Hawk   7   <NA>
> CBC_Data %>%
+     group_by(Location, Year) %>%
+     summarise(N_Spp = n())
   Location   Year  N_Spp
1  Peterborough   2017     51
2  Peterborough   2018     55
3  Peterborough   2019     47
4  Peterborough   2020     54
```

Let's Practice

Aggregating data

Piping blanks

Lines

710-717:

```
Step3_pipe <- CBC_data_cleaned %>%
filter(Trophic.Level ==  "Omnivore" & Year ==  2020 & Location ==
       "Peterborough" & Count > 0) %>%
select(Scientific_Name, Common_Name, Count) %>%
arrange(desc(Count))
```

744-748:

```
CBC_data_cleaned %>%
       filter(Count > 0) %>%
       group_by(Location, Year) %>%
       mutate(Total_Count = sum(Count, na.rm = TRUE)) %>%
       data.frame() %>% head()
```

752757:

```
CBC_data_cleaned %>%
       filter(Count > 0) %>%
       group_by(Location, Year) %>%
       summarise(Total_Count = sum(Count, na.rm = TRUE)) %>%
       data.frame() %>% head()
```

Challenge 5

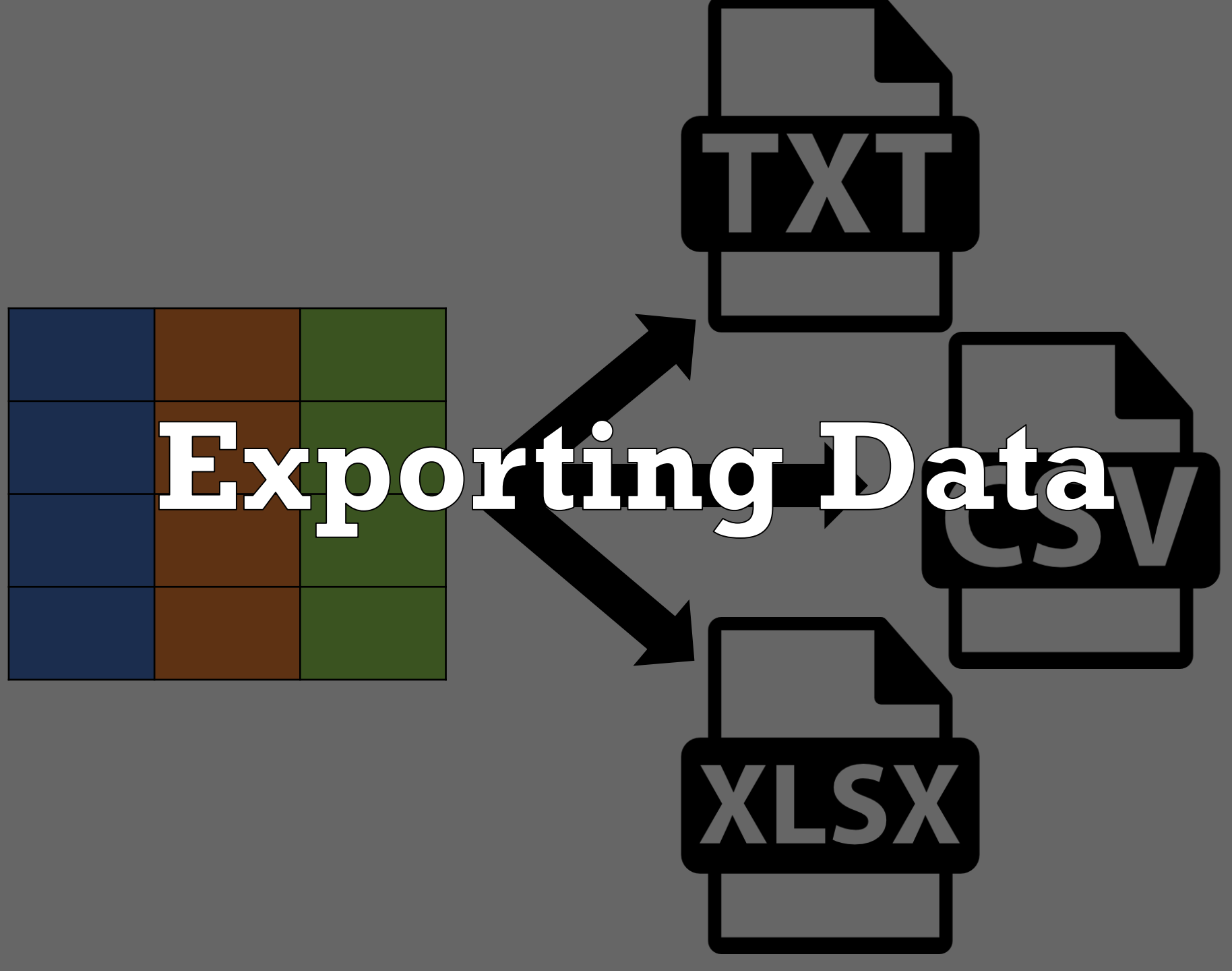
What is the correct code to produce a summary table of average break width per habitat? That table should only show Habitat andAvg_BeakWidth.

a)
```
> CBC_data_cleaned %>%
+    summarise(Avg_BeakWidth = mean(Beak.Width))
```

b)
```
> CBC_data_cleaned %>%
+    mutate(Avg_BeakWidth = mean(Beak.Width))
```

c)
```
> CBC_data_cleaned %>%
+    group_by(Habitat) %>%
+    summarise(Avg_BeakWidth = mean(Beak.Width))
```

d)
```
> CBC_data_cleaned %>%
+    group_by(Habitat) %>%
+    mutate(Avg_BeakWidth = mean(Beak.Width))
```

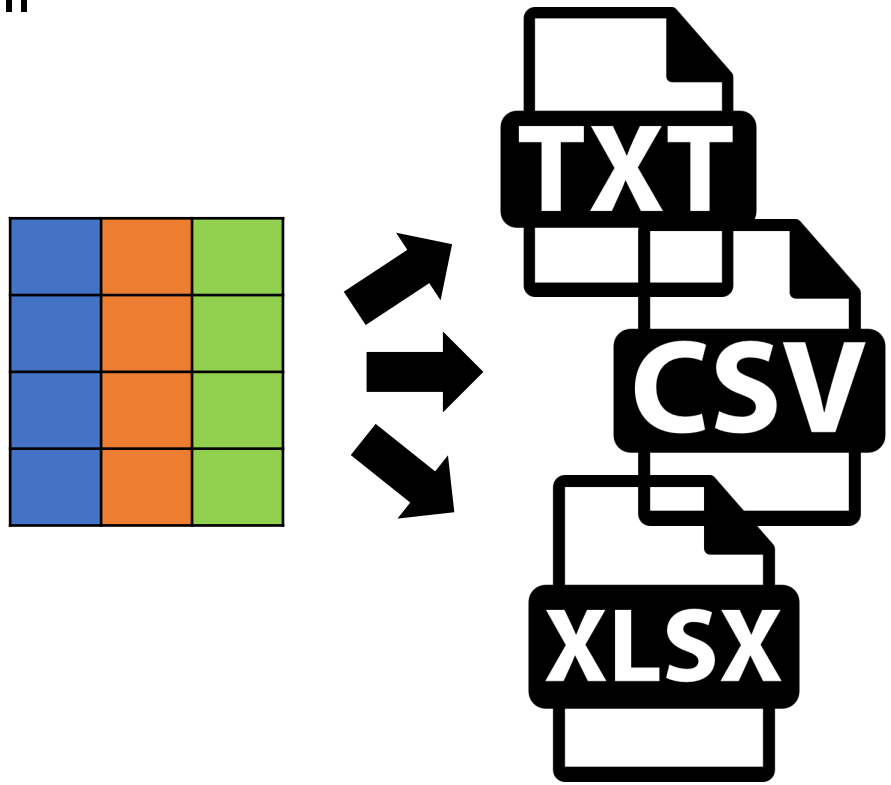

Exporting Data
TXT
csv
XLSX

Export data

Once data has been manipulated, it can be exported in all multiple formats (depending on the data).

R can read data from all common file types:

- Text (.txt)
- Comma separated values (.csv)
- Excel worksheet (.xlsx)
- SPSS dataset (.por)
- SAS Data (.xpt)
- Stata (.dta)
- Raster (.tiff, .grd, .bil, .asc, .sdat, .rst, .nc, .envi, .img)
- Shapefiles (.shp)
- JSON (.json)



See `readr` and `forgein` packages functions to export other file types

Export functions

Text files are exported with `write.table()`


```
> write.table(x = data, file = "file.txt", sep="\t",
+                 row.names = TRUE, col.names = TRUE)
```

Comma separated values are exported with `write.csv()`


```
> write.csv(x = data, file = "file.csv", row.names = TRUE)
```

Excel Worksheets (.xlsx) are exported with `write.xlsx()`


```
> library(openxlsx)
> write.xlsx(x = data, file = "file.xlsx")
```

R environment data (.rda):

```
> save(list = ls(), "file.rda")
```

Export the entire R environment

```
> save(data1, data2, "file.rda")
```

Export the only objects data1 and data2

Let's Practice

Exporting

Exporting blanks

Lines

831-833:

```
write.table(x = CBC_summary,
            file = "Export/Counts_per_Habitat_2017_2020.txt",
            col.names = TRUE, row.names = FALSE, sep = "\t")
```

844-846

```
write.csv(x = CBC_summary, file = "Export/Counts_per_Habitat_2017_2020.csv",
              row.names = FALSE)
```

Maps, Data and Government Information Centre

Bata Library 415
8:30 am to 4:30 pm, Monday to Friday

madgichelp@trentu.ca

https://www.trentu.ca/library/madgic