FDS::**CHEATSHEETS**[“project 3”]

EPFL EXTENSION SCHOOL

Hints

Part 1

- Import and clean the datasets.
- Store them in variables

Part 2

- Create a dataset containing the number of recommendations by each member.
- Rely on the `recommendation_from` column
- Attach it it to the dataset using the a joining {dplyr} verb and the by= argument

Part 3

- Rely on the joined table created in point 3
- Use dplyr::**filter()** or dplyr::**slice_max()** and dplyr::**pull()**

Part 4

- Rely on the joined table created in point 3
- Use dplyr::**filter()** or dplyr::**slice_max()** and dplyr::**pull()**

Part 5

- Rely on the joined table created in point 3
- Use dplyr::**filter()** or dplyr::**slice_max()** and dplyr::**pull()**

Part 6

- Use dplyr::**mutate()**

Part 7

- You could use a **set operator** or a **filtering join**

Part 8

- If you don’t remember it is tidy::**pivot_longer()**
- Use dplyr::rename() to change the weight into wk_000
- You can use a **selection helper**

Part 9

- Use dplyr::**filter()** or dplyr::**slice_max()** and dplyr::**pull()**

Part 10

- Use dplyr::**filter()**
- dplyr::**group_by()** and dplyr::**lag() can work in conjunction**

Data Import

readr::**read_csv(“file.csv”)** Allows you to read a comma delimited file

Data Cleaning

janitor**::clean_names()** Creates columns names in `snake_case`

janitor**::tabyl()** Allows you to create frequency tables

Pivoting Data

table4a

country	1999	2000
A	0.7K	2K
B	37K	80K
C	212K	213K

→

country	year	cases
A	1999	0.7K
B	1999	37K
C	1999	212K
A	2000	2K
B	2000	80K
C	2000	213K

pivot_longer(data, cols, names_to = "name", values_to = "value", values_drop_na = FALSE**)**

"Lengthen" data by collapsing several columns into two. Column names move to a new names_to column and values to a new values_to column.

```
pivot_longer(table4a, cols = 2:3,
    names_to ="year",
    values_to = "cases")
```

Data Wrangling

COUNTING

summarise() applies summary functions to columns to create a new table. Summary functions take vectors as input and return single values as output.

dplyr::**n()** - number of values/rows
dplyr::**n_distinct()** - # of uniques
sum(!is.na()) - # of non-NA’s

summarise(.data, ...**)**
Compute table of summaries.

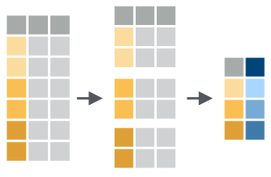
```
summarise(mtcars, avg = mean(mpg))
```

count(.data, ..., wt = NULL, sort = FALSE, name = NULL**)** Count number of rows in each group defined by the variables in ... Also **tally()**.

```
count(mtcars, cyl)
```

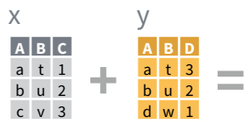
COUNTING GROUP SIZES

Use **group_by(**.data, ..., .add = FALSE, .drop = TRUE**)** to create a "grouped" copy of a table grouped by columns in ... dplyr functions will manipulate each "group" separately and combine the results.



```
mtcars  %>%
   group_by(cyl) %>%
   summarise(n =n ())
```

JOINS AND FILTERING JOINS



left_join(x, y, by = NULL**)** Join matching values from y to x.

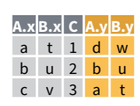
semi_join(x, y, by = NULL, copy = FALSE, ..., na_matches = "na"**)** Return rows of x that have a match in y. Use to see what will be included in a join.

anti_join(x, y, by = NULL, copy = FALSE, ..., na_matches = "na"**)** Return rows of x that do not have a match in y. Use to see what will not be included in a join.

COLUMN MATCHING FOR JOINS

Use **by = c("col1", "col2", ...)** to specify one or more common columns to match on.

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



Use a named vector, **by = c("col1" = "col2")**, to match on columns that have different names in each table.

```
left_join(x, y, by = c("C" = "D"))
```

SET OPERATIONS

intersect(x, y, ...)
Rows that appear in both x and y.

setdiff(x, y, ...)
Rows that appear in x but not y.