

Problem Set #2 Cheat Sheet

Victoria, Menissah, Sarayu

Key Terms:

Function: Pre-written pieces of code that accomplish some task; they generally follow three steps: Input, Process, Return

Data Frame: An object where the type is a list; datasets are data frames

Length (in context of data frame): The number of variables in a data frame

Element Name (in context of data frame): Variable name

NA: Missing value, different from character value “NA”

Filter: Selecting observations based on values of variables

Tips and Tricks:

- **If your code isn't running, check punctuation and formatting**
 - Backtick (`) vs. apostrophe (')
 - Spaces between characters and values
 - Indentation
 - Etc.
- **Make sure your code aligns with the variable values in your data frame**
 - Decimals vs. Whole Numbers, etc.
- **Use shortcuts, when applicable**
 - <https://support.rstudio.com/hc/en-us/articles/200711853-Keybaord-Shortcuts>
 - 'ALT' + '-' gets you the assignment operator '<-' only works in r code not in text
- **Explore alternate r packages if something isn't working**
 - Tinytex, smaller and simpler to install than MikeTex

```
install.packages('tinytex')
```

```
tinytex::install_tinytex
```

Cannot have another latex program installed (Miktex for examples)

- **Remember that there is often more than one way to accomplish a function**

Example with filters (PS2, Section 3, Question 3):

VERSION 1:

```
uni_alabama <- filter(df_school_all, visits_by_100751 >= 1, school_type == "public", state_code == "AL")
```

```
count(filter(uni_alabama, pct_black >= 50.0000 | pct_hispanic >= 50.0000))
```

VERSION 2:

```
count(filter(df_school_all, state_code == "AL", school_type == "public", visits_by_100751 >= 1, pct_black >= 50 | pct_hispanic >= 50))
```

Problem Set #3 Cheat Sheet

Anne, Esthela, Laura, Lauren

Key Terms:

Arrange: Sort the observations/rows in a data frame in ascending or descending (using `desc()`) cording to one or more variables.

Pipe (%>%): The pipe operator helps you perform multiple steps sequentially (left to right) in one line of code. It is only available in the tidyverse suite.

Input variable: The (possibly raw) data that is used to create new variables for analysis (rather than replacing the original data). It is a good idea to investigate an input variable before you create new variables. Try using the `summary()` function to find out mean, median, min/max, using `range()` to find min/mad, and/or looking into missing data.

Analysis variable: The variables used in your analyses. Many of these will be created using `mutate()`

Assign: By using the assignment operator (<-), you can save an object to a data frame (new or existing). Assignment occurs at the beginning of a line of code, even when using pipes.

Tips and Tricks

- **To limit the number of rows displayed (i.e., first 15)**

- head () function
- Examples:
 - Without pipes

```
head(select(arrange(df_school_all, desc(total_visits)), name, state_code,
city, school_type, total_visits, med_inc, pct_white, pct_black,
pct_hispanic, pct_asian, pct_amerindian), n=15)
```
 - With pipes

```
df_school_all %>%
  arrange(desc(total_visits)) %>%
  select(name, state_code, city, school_type, total_visits, med_inc,
pct_white, pct_black, pct_hispanic, pct_asian, pct_amerindian) %>%
  head(n=15)
```

- **Assignment problems**

- If you encounter problems with assignment, try creating a new data frame with each test of your code, so that you don't have to keep reloading the data to work with it. You can keep track of what happens with each change, taking notes along the way. Example:

```
df_school_all2 <- df_school_all %>% select(num_took_math) %>%
mutate(miss_took_math = ifelse(is.na(num_took_math),1,0)) %>%
count(miss_took_math)
```

#This creates a data frame with 2 obs and 2 variables

```
df_school_all3 <- df_school_all %>% select(num_took_math) %>%
mutate(miss_took_math = ifelse(is.na(num_took_math),1,0))
```

#This creates a data frame with 21301 obs and 2 variables

- **Issues with knitting (specific to tidyverse)**

- When loading tidyverse, the output displays check marks that indicate which specific tidyverse packages were installed (e.g., purrr, dplyr, etc).
- These check marks may not be read by the LaTeX set up because they are special symbols that requires "xelatex".

- In order to knit to PDF, you may have to suppress the output in the chunk by wrapping the library(tidyverse) with another function called suppressMessages (note that there are quotes around tidyverse):

```
suppressMessages(library("tidyverse"))
```
- Here is a link where Hadley Wickam comments that the message has to be suppressed in order to knit to PDF:
<https://community.rstudio.com/t/tidyverse-1-2-1-knitting-to-pdf-issue/2880/7>

- **When using case_when(), the order of your variables matters**

- The following code works (but creates a value with string name “NA” instead of a missing value):

```
wwlist <- wwlist %>%
  select(for_country, state) %>%
  mutate(residency = case_when(for_country == "No Response" ~ "NA", state ==
    "WA" ~ "in_state", state != "WA" ~ "out_state_us", !is.na(for_country) ~
    "not_in_us"))
```

- The following code does not create any NA values, because `for_country` is not *missing* for the "No Response" students (it is a string: “No Response”), so the students with “No Response” get coded as "not_in_us" during the third case_when statement:

```
wwlist <- wwlist %>%
  select(for_country, state) %>%
  mutate(residency = case_when(state == "WA" ~ "in_state", state != "WA" ~
    "out_state_us", !is.na(for_country) ~ "not_in_us", for_country == "No Response"
    ~ "NA"))
```

- Here is a better option, which creates three value labels and leaves the 17 students with “No Response” as true missing (<NA>)

```
wwlist <- wwlist %>%
  select(for_country, state) %>%
  mutate(residency = case_when(
    state == "WA" ~ "in_state",
    state != "WA" ~ "out_state_us",
    is.na(state) & for_country != "No Response" ~ "not_in_us"))
```

Problem Set #4 Cheat Sheet

Lupe, Ramon, Zhaopeng

Functions

Group_by(): Converts a data frame object into groups. After grouping, functions performed on data frame are performed “by group”

- Basic Syntax: ◦ `group_by(object, vars to group by separated by commas)`

summarise(): does calculations across rows; then collapses into single row

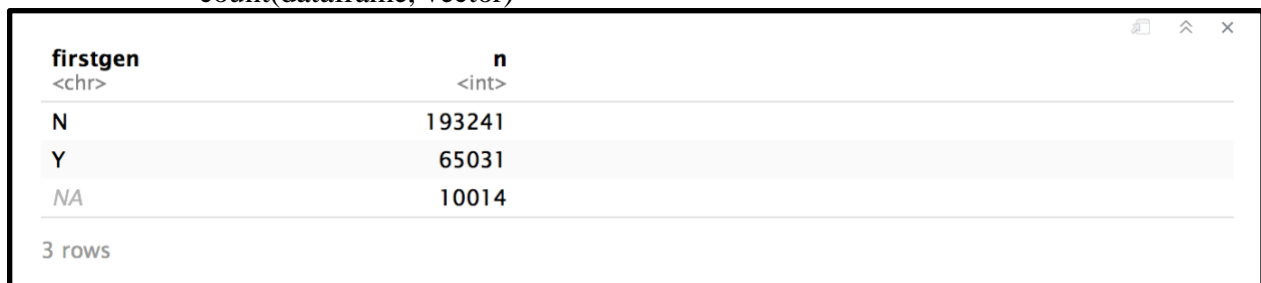
- Usage (i.e., syntax): `summarise(.data, ...)`

summarise() function combined with:

- **group_by :** `summarise()` on ungrouped vs. grouped data:
 - By itself, `summarise()` performs calculations across all rows of data frame then collapses the data frame to a single row
 - When data frame is grouped, `summarise()` performs calculations across rows within a group and then collapses to a single row for each group
- **counts:** The count function `n()` takes no arguments and returns the size of the current group
- **Mean:** The `mean()` function within `summarise()` calculates means, separately for each group
- **Mean and na.rm argument:** Many functions have argument `na.rm` (means “remove NAs ”)
 - `na.rm = FALSE` [the default for `mean()`]

Tips and Tricks:

- Different ways to yield count of missing values (NAs) for vectors in a dataframe
 - `count(dataframe, vector)`



firstgen <chr>	n <int>
N	193241
Y	65031
NA	10014

3 rows

- `count(dataframe, is.na(vector))`

is.na(firstgen)	n
<small><lgl></small>	<small><int></small>
FALSE	258272
TRUE	10014

2 rows

- `table(dataframe$vector, useNA = "always")`

N	Y	<NA>
193241	65031	10014

- `dataframe %>% count(vector)`

firstgen	n
<small><chr></small>	<small><int></small>
N	193241
Y	65031
NA	10014

3 rows

- `dataframe %>% count(is.na(vector))`

is.na(firstgen)	n
<small><lgl></small>	<small><int></small>
FALSE	258272
TRUE	10014

2 rows

- `dataframe %>% filter(is.na(vector)) %>% count(vector)`

firstgen	n
<small><chr></small>	<small><int></small>
NA	10014

1 row

- Important to know the differences and implications of ``select`` and ``filter``
 - ``select`` retains or drops particular vectors, whereas ``filter`` retains or drops particular observations that meet (or do not meet) the specified conditions of a vector

- for `select`, ranges of vectors can also be specified; e.g., `dataframe %>% select(vector1:vector10)`
- Key to remember the effect of using `select` and `filter` when assigning to a dataframe, especially the original dataframe since it will most likely change the presence and/or order of vectors in it
- `summarise` can be used with or without `group_by`
 - `summarise` with `group_by` - `dataframe %>% group_by(vector) %>% summarise(new vector = function(input vector))`
 - will present specified calculations by the elements of the vector(s) being grouped by



in_state <dbl>	tot_prosp <int>
0	172268
1	96018

2 rows

- `summarise` without `group_by` - `dataframe %>% summarise(new vector = function(input vector))`
 - will present overall calculations for all observations in the dataframe



tot_prosp <int>
268286

1 row

Problem Set #5 Cheat Sheet

Lindy & Michelle

Data Investigation:

FUNCTIONS WE FOUND HELPFUL:

- **group_by():** converts a data frame object into groups.
 - Works well with: `count()` and `summarise()`
- **names():** list names of variables (or columns) in the data frame
- **var_label():** lists full variable names (e.g., “crsecrred” = “course credits possible”)

- **summary()**: provides min, max, standard deviation for each numeric variable (column); provides number of rows for “character” class variables.
- **filter()**: Shows rows for specific values
- **select()**: shows specified variables/columns

OTHER HELPFUL FUNCTIONS FROM CLASS DISCUSSION

- **min()** and **max()** for particular variables; must use with **summarise_at** or **mutatate()**
- **structure()** gives overview of the each variable (data type, a few rows, etc.) and gives a nice overview of the dataset as a whole

Notes from class:

- Strategies may change depending on the size of your dataset, e.g. can you simply **View()** your data or do you need to use functions to find min, max, etc.
- Investigating your data will help you understand the limitations of your data and thus the limitations of your results (like if particular schools/sites have missing or dirty data)
- Being able to combine functions is a must.
- In the viewer, you can see the range and there’s a slider. You can “see” the data at the unit level (for this data set, unit level is class level), which may be more intuitive.

Data Cleaning

HELPER FUNCTIONS FOR MUTATE

- **mutate()**: creates new column (variables) based on existing column(s).
 - **ifelse()**: Uses a logical condition (T/F);
 - **recode()**: Takes values within one variable and sets them to new values.
 - **case_when()**: Uses multiple variables and conditions within those variables. Left side is which values match the case, and right side is the new value.
- Often uses logical operators: `is.na`, `>`, `%in%`, `==`, etc.
- Can return the original values of the variable within a function:
 - **Example:** `Ifelse(oldvariable > cutoff, NA, oldvariable)`.
 - `Ifelse(crsecr>4.00, NA, crsecr)` which means if course credit is greater than 4.00 credits, make it null, and if it’s not greater than 4.00 credits, keep the same value of course credit that it originally had in `crsecr`.
- Critical for creating `numgrade`, `crscredv2&3`, and `GPA` variables.

Notes from class:

- Knowing conventions of the field is helpful, particularly for null values
 - NA values can be coded as 99, 999.00, 999.999, or even as a negative number
- If you don’t assign an output for a case in your **case_when()** function, then it will be assigned NA as its value.
- You cannot explicitly assign “NA” as part of the **case_when()** function. Must use “NA_Real”. Or you can leave that case unassigned, and the function will automatically assign NA to “leftover cases”