EXERCISES

NU Pathology LearnR! Fall 2020
Sessions 3-4

Contents

DATA STRUCTURES AND TYPES

PRACTICE: Creating vectors, lists, and dataframes

(a) Using the code examples and information in the lesson,

1. Create a vector of at least 3 words.
2. Have R return the length of your vector.

(b) Describe how lists, vectors, and dataframes are similar and how they are different.
Solution: Vectors are 1-dimensional objects containing a SINGLE data type. Lists are 1-dimensional objects containing more than one data type. Data frames are 2-dimensional objects containing more than one data type. Columns in a data frame contain a single data type (so are vectors), but rows may contain more than one data type. We use different functions to create vectors (c()), lists (list()), and data frames (data.frame()).

(c) Create a list, mk8_list, with the following components:

- racer: Mario, Princess Peach, Wario, Baby Luigi
- wt: middle, middle, heavy, light
- speed: 3.75, 3.25, 4.75, 2.25

(d) Print the list and examine its structure.

(e) Create a dataframe, mk8_df, with the same components above.

(f) Print the dataframe and examine its structure. How can you find the dimensions of your dataframe?

(g) Before you run the code below, determine the output and then run the code - were you correct? What type of structure are you operating on in the chunk below?

```
racer <- c("Mario", "Princess Peach", "Wario", "Baby Luigi")
racer[2]
racer[3:4]

racer[[2]]
```

(h) Similar to above, think about the output for the code below, then run the code - were you correct? What is the data structure of each output for the code below?

```
one <- mk8_list[1]
one

mk8_list[2:3]

two <- mk8_list[[1]]
two

mk8_list$racer
```

(i) Finally, let's do the same for a dataframe. Inspect the code, predict the output, and check by running the code. What is the structure of the output for the code below?

```
mk8_df[1,1]
mk8_df[ ,2]
mk8_df[2, ]

mk8_df[[1,2]]
mk8_df[[2]]
```

(j) Using the [ ] notation, how would you return the data from the second row in the 2nd and 3rd columns of mk8_df?

(k) Using the [ ] notation, how would you return the data from rows 1 AND 3 for all columns in mk8_df?

MODIFYING COLUMNS

PRACTICE: Creating or modifying columns

Find the data file from Session 2: NHANES_FeMarkers_3to5y_w_Dates.csv
Read this into R (using `read.csv()`), saving as variable, nhanes_data.
Modify the columns to
(a) change the Subject, Gender, and Race_ethn columns to factor class, using factor()
(b) change the Age_months column to integer class, using as.integer()
(c) change the Resulted_t to datetime class, using as_datetime()

Create a new column, Hgb_dfct_gdL, for the Hemoglobin deficit:
(d) Hgb deficit (g/dL) = 14.0 - patient hgb conc.

Check that your code worked to change the classes of those variables.

EXTEND: Using helper functions

Now that you know about `mutate()`, we'll introduce its 'scoped' variants. These variant `mutate()` functions, suffixed with _if, _at or _all, apply an expression (sometimes several) to all variables within a specified

subset. This is super helpful, as it means we can apply the same transformation to multiple columns versus selecting individual columns by name.
- `mutate_all()` affects every variable
- `mutate_at()` affects variables selected with a character vector
- `mutate_if()` affects variables selected with a predicate function
What is a predicate function? A function that returns TRUE or FALSE. In this case, the mutate applies to variables where the predicate function is TRUE.

(a) Let's use `mutate_if()` to create a series of columns that transform each of the numeric columns *into a new column* with data that is centered and scaled using the `scale` function. The new column(s) should be named as: orig_colname_scaled (i.e., append "scaled" to original column name). You should get a tibble with 295 obs of 21 variables.
Hint: the syntax for `mutate_if()`: mutate_if(.tbl, .predicate, .funs, ...). The .tbl is your data (tibble), the .predicate is your predicate function (e.g., is.integer), the .funs is the function you want to apply to the selected variables (e.g., as.numeric, mean)

SELECTING, FILTERING, ARRANGING

PRACTICE: Creating subsets of data and arranging them

Use the `%>%` to chain together the following code to create a new subset of the NHANES data set:
(a) Select the first 5 columns of your NHANES data set - try using a numeric representation of columns (i.e., 1:2).
(b) Filter it for female subjects (i.e., Gender = 2).
(c) Arrange it by Race_ethn.

EXTEND: Using helper functions and operators

`select()`, `filter()`, and `arrange()` also have scoped versions that work like the `mutate()` variants described above.

As briefly mentioned in class, we can simplify our conditions for filtering ranges of values by using `between()`. Instead of using conditions of x >= left & x <= right, we can use between(x, left, right).

(a) Use `select_at()` with one of its helpers on your extended NHANES data set from above to select only columns that **DON'T** contain "_scaled".

(b) Filter your new data set for rows with a hemoglobin deficit between 0 and 2 g/dL – use `between()`.

(c) Arrange your new data set based on both Hgb_gdL and Ft_ngdL.

GROUPING & SUMMARIZING

PRACTICE: Grouping data by category and summarizing it

(a) Create a table that shows the median value and standard deviation of hemoglobin for each age.

EXTEND: Using scoped variants to summarize

Let's use a scoped variant of summarize to extend the summarization you performed above to all numeric columns with little additional code.

(a) Use `summarize_if()` to create a table that shows the median value and standard deviation of *all numeric* columns for each age.

(b) How is this different that what you did above using `mutate_if()`?

(c) What scoped variant would you use if you wanted to summarize columns based on matching to a character string?

WORKING WITH DATES

PRACTICE: Parsing date formats and extracting components

Which `lubridate` function would you use to parse a character column of dates with each the following formats:
(a) "02/28/17 12:01:34"

(b) "June 6, 1997"

(c) "2017-22-12 10:00"

Another great thing we learned about `lubridate` is its ability to help us extract different components of dates (e.g., weekday, month, etc.).

(d) Create a new column, named, weekday, in your NHANES data set that indicates the day of the week the sample was collected. Format the column so the full name of the day is displayed.

EXTEND: Creating date times from components

In our previous examples we were given character strings or numerics representing date formats that we could parse. The other way to create a date or datetime object is from combining individual components. The functions `make_date()` and `make_datetime()` help you do this.

An example of the syntax:

```
# to create a datetime of 01-05-2019 09:12:43 EST; all arguments are numeric with exception of tz (time
make_datetime(year = 2019,
              month = 01,
              day = 05,
              hour = 09,
              min = 12,
              sec = 43,
              tz = "EST")
```

```
## [1] "2019-01-05 09:12:43 EST"
```

Try it yourself using the `flights` dataset. This data is in the `nycflights13` package. You will need to install the package and load it.

(a) Create a new object, flights_data, that is the `flights` dataset.
Use the `head()` or `View()` functions to examine your new object.

(b) Create a new column for flights_data, departure_t, from the individual numeric component columns: year, month, day, hour, min.

(c) Check that the departure_t column is a datetime class.
