R cheat sheet

By @MaryJoWebster
March 2019

Data frame – This is from Base R; this is used for storing data tables. Tidyverse makes a "tibble" which is supposed to be a slightly better version of a data frame.

<- This is an assignment operator that is used to assign data to a data frame (Option+- on a Mac)

%>% This is a pipe operator used in Tidyverse to connect actions. (Cmd+Shift+M on a Mac)

See the Tools menu in RStudio and look at Keyboard Shortcuts (the key strokes are different for Windows and Mac)

df$columnname --- You might see this syntax occasionally. This is how to be explicit about exactly which dataframe a column is coming from. It's the name of the data frame with a dollar sign and then followed by the name of the column.

Stringr::sub_str() -- you might see syntax like this. This is how to be explicit about which package a function is coming from. In this case it's the "stringr" package and sub_str is the name of a function from that package.

getwd() --how to find out what your working directory is. You can type this in any file (script, markdown, etc) or in the console.

Dplyr basic syntax:

Select(columnname, columname)

Filter(columname== 'value')

Group_by(columnname)

Summarize(newcolumnname = n()) --- this will count the records

Summarize(newcolumnname = sum(columnname)) -- this will sum the values in a column

Summarize(newcolumnname = mean(columnname)) -- this will average values in a column

Arrange(columnname) -- this will sort your results by column specified, ascending

Arrange(desc(columnname) – this will sort descending

Mutate(newcolumnname = …….)) -- this is how you add a new column

How to calculate a percent when grouping data:

```
Df %>%
group_by(statename) %>%
summarize(count = n() ) %>%
mutate(pct = count/sum(count)

#note- the mutate line is grabbing the "count" field created in the summarize line and dividing it by the sum of all the values in "count"
```

How to add a new column to a data frame using mutate. For this example, let's assume we have two columns we want to add together to create a "total" column.

```
df <-  df  %>% mutate(total = column1 + column2)
```

Importing using Tidyverse:

Readr package in Tidyverse comes with 5 "parsers" for importing different text file types. https://readr.tidyverse.org/articles/readr.html

(Note that BaseR has a function called "read.csv" that functions very differently)

read_csv() – for comma-delimited files
read_tsv() – for tab-delimited files
read_fwf() – for fixed-width files
read_log() for web log files

If you need to import an Excel file, you need the **readxl** package from Tidyverse (see below)

Syntax for read_csv. Make sure your data file is in your working directory.

```
Df <- read_csv("mydatafile.csv")
```

If your data file is in a sub-directory of your working directory (in this example my sub-directory is called "data"):

```
Df <- read_csv("./data/mydatafile.csv")     #note the period at the beginning
```

Read_csv will guess the column formats (character, numeric, double, date, etc) based on the first 1,000 rows in the file. You can tell it took at more rows, or you can specify column types as part of the import code.

```
Df <- read_csv(“./data/mydatafile.csv”) ,
        col_types= cols(.default = col_character(), col1 = col_double(),
                col2 = col_date(format= “%m/%d/Y”) ) )
```

The example above tells it to default to character for all fields, except col1 and col2 which are set to specific types.

At the link above you can see all the column types and syntax, but here are the common ones:
col_integer
col_double
col_character
col_date
col_datetime

For date and datetime, you need to tell R what the format is of the dates coming in. In the example above, I told it that my data has dates that appear like “5/10/2018” (m/d/yyyy)

More info on this here: https://readr.tidyverse.org/reference/parse_datetime.html

Using **readxl** to import from Excel
https://readxl.tidyverse.org/

The nice thing about this package is that you can specify the sheet and the “range” of rows and columns where your data is sitting in the Excel workbook. So if your Excel files contains a bunch of notes or other junk at the top of the file, you can have R just ignore that part.

```
df <-  read_excel("mydatafile.xlsx", sheet="sheetname", range="B14:E312")
```

Joining data frames:

Joining is how you put two data frames together side-by-side, matching on one or more idnumbers or other columns that match in the two data frames. Your new data frame will have more columns, not necessarily more rows, depending on the join type.
More useful info here: https://medium.com/@HollyEmblem/joining-data-with-dplyr-in-r-874698eb8898

Join types:

Inner_join – this will return only the records that match between the two tables

Left_join – this will return all the records from the first table listed, but only the ones that match from the second table listed

Right_join – the opposite of a left_join. All records from second table.

Anti_join – this will return all the records from the first table that are NOT IN the second table

Full_join – this will return all the records from both tables, regardless if they match

Basic join syntax:

df_new <- inner_join(df1, df2, by=c("idnumber"= "idnumber"))

This basic syntax will put the two data frames together and will put all the columns (from both tables) into a new data frame. It will not repeat the "joining" column if they have the same name. If there are other columns with the same names, your new columns will be named with a ".x" and ".y" attached to the end of the name to identify which table they came from. X refers to the first table listed in the join. Y refers to the second.

Note: when you are referring to the column names you are joining on, the first one needs to come from the first table listed. Think of it like this:

df_new <- inner_join(df1, df2, by=c("df1$idnumber"= "df2$idnumber"))

If you need to join by more than one column, here's how:

df_new <- inner_join(df1, df2, by=c("idnumber"= "idnumber", "datayear"= "datayear"))

If you want to only bring a few columns from the second table, you can do that as part of the join. Note that we include a pipe and select command before the comma and you MUST include any columns that are being used in the join.

df_new <- inner_join(df1, df2 %>% select(idnumber, col1, col2, col3) , by=c("idnumber"= "idnumber"))

Appending data:

Typically you would use this when you have two data files – hopefully with all the same columns and column names – that you want in one data file. In other words, you're adding the rows from one to the bottom of the other data frame.

Tidyverse has a function called "bind_rows" that will do this, but thankfully it doesn't insist that the number of columns (and their names) be identical between the two data frames. Base R has a function called "rbind" that does require these to be exact. So bind_rows tends to be more flexible.

Here we're going to make a new data frame by binding two existing data frames together:

df_new <- bind_rows(df1, df2)

Bind_rows will line up the columns in df1 with those in df2 that have the same name. If you have a field in df1 called "state" and a column in df2 called "statename", bind_rows will put

those in separate columns in your new data frame. And if you have a column that exists in one table but not the other, that will still show up as a column in your new data frame.

Renaming columns:

There are a couple different ways to rename a column. This syntax is what you'd likely use when you are importing or building a new data frame.

For example, this would create a data frame with just 4 columns selected from the old one and it will rename col1 in the process:

```
Df_new <- df %>% select(col1, col2, col3, col4)%>% rename(newname = col1)
```

If you just want to keep your data frame with all the columns, but rename one or more fields you can do this:

```
Df_new <- df %>% rename(newname1 = col1, newname2 = col2)
```

You can also rename on the fly in your analysis. In this example, the field is currently called "gender" but I want to change it to "sex".

```
df %>% select(sex=gender, firstname, lastname)

df %>% group_by(sex=gender)%>% summarize(count=n() )
```

Using case_when from Tidyverse:

Case_when is like an IF statement in Excel. You can use it to have R do different things depending on some variable or value.
https://www.rdocumentation.org/packages/dplyr/versions/0.7.8/topics/case_when

The basic syntax for case_when is like this (and typically you'd use it in combination with mutate to make a new column)

```
Df <- df %>% mutate( newcolumn = case_when( col1 = x ~ "new value 1",
                                            col1 = y ~ "new value 2",
                                            TRUE ~  "new value 3" ))
```

Note: the tilde (~) is the operator here, assigning the new value.

The word "TRUE" at the end is a catch-all for any records that don't meet the other criteria set out in the first two arguments.

You can have as many arguments as you want (the above example has 2 arguments, plus the catch-all)

This is super handy for cleaning up dirty data or for making a new column that assigns some sort of label to each row. For example, let's say you have a data frame with each row being a zip code and a field that identifies the median household income in that zip code. You want to create a new field that indicates if the median household income in either "above", "below" or "about the same" as the state's overall median household income. In this example, let's say the statewide median is $55,000. I want "about the same" to be anything from $50,000 to $59,000.

```
Df  <- df %>%
        mutate( bucket = case_when(income < 50000 ~ 'below',
                                   income >=50000 & income < 60000 ~ 'about the same' ,
                                   income >=60000 ~ 'above' ) )
```

Other resources:

R for Data Science (online book): https://r4ds.had.co.nz/

RStudio has numerous tipsheets, although they are so succinct that sometimes they are hard to understand: https://www.rstudio.com/resources/cheatsheets/

Aggregating and analyzing data with dplyr: https://datacarpentry.org/dc_zurich/R-ecology/04-dplyr.html

Datacamp: Introduction to the Tidyverse: https://www.datacamp.com/courses/introduction-to-the-tidyverse

Andrew Ba Tran Introduction to R online course: http://learn.r-journalism.com/en/introduction/

Storybench R articles: http://www.storybench.org/tag/r/

How do I….? https://smach.github.io/R4JournalismBook/HowDoI.html

Packages:

Muckrakr: https://github.com/andrewbtran/muckrakr

Knitr: https://yihui.name/knitr/

Janitor package – tabyls function - https://cran.r-project.org/web/packages/janitor/vignettes/tabyls.html

Kable and KableExtra: https://haozhu233.github.io/kableExtra/awesome_table_in_html.html

Data tables (DT): https://blog.rstudio.com/2015/06/24/dt-an-r-interface-to-the-datatables-library/

Themes in ggplot: https://ggplot2.tidyverse.org/reference/ggtheme.html

Stringr package: http://www.storybench.org/getting-started-stringr-textual-analysis-r/