R Notebook

R Markdown Tips

When you execute code within the notebook, the results appear beneath the code.

Add a new chunk by clicking the *Insert Chunk* button on the toolbar or by pressing *Ctrl+Alt+I*.

When you save the notebook, an HTML file containing the code and output will be saved alongside it (click the *Preview* button or press *Ctrl+Shift+K* to preview the HTML file).

The preview shows you a rendered HTML copy of the contents of the editor. Consequently, unlike *Knit*, *Preview* does not run any R code chunks. Instead, the output of the chunk when it was last run in the editor is displayed.

Loading Packages with pacman in one line

Tips: 1. When installing a package with the base R function `install.packages`, use quotes around the package name. 2. the pacman library simplifies installing and loading packages to a single command, `p_load`. It will only install a package if you don't have it already 3. Run each command at a time and record in the console below, by pressing *Ctrl-Enter* anywhere in each line, or group of highlighted lines 4. If you see warnings in red in the console that is often totally ok. Errors are different than warnings and need to be debugged. 5. When you see a right-pointing blue caret in the console, then R is done and ready for another command

1st we install and load the readr library

```r
# this installs pacman if you don't already have it
if (!require("pacman")) install.packages("pacman")
```

```
## Loading required package: pacman
```

```r
#the pacman::p_load function will install readr if it isn't installed
pacman::p_load("readr", "knitr")
#then, pacman::p_load will load it
```

Loading data with a relative path, using the `..` symbol

Run commands in this cell line by line with *CNTRL+Enter*

When working in a .Rmd file, the working directory is the location of the .Rmd file. The simplest way to tell R where to find data files is relative to the .Rmd file you are working in. Since this file is in the `tutorials` folder, and this folder is next to the `data` folder, we can specify the location of our file with `"../data/la_jolla_pm25_wind_data.xlsx"`

If you ever want to see visually where your files and folders exist in Rstudio, use the bottom right Files pane to explore directories. Clicking `..` will take you up a directory level.

```
p_load(readxl)
# load the data into a dataframe called air_data
# i get a read warning msg but then the blue caret so we can ignore it
air_data <- read_excel("../data/la_jolla_pm25_wind_data.xlsx")
```

```
## New names:
## * Date -> Date...1
## * Date -> Date...6
```

Ways to set the working directory (optional)

In R scripts, `getwd()` shows you the location you are currently working in, which should be the location of the .R script. However, .Rmd files set the working directory differently `setwd()` needs to be run for every chunk. So `setwd` and `gewd` commands aren't very useful for .Rmd files. What follows is a description of these commands if you ever need to view or set working directories in an R script.

After running `getwd()` Windows users will see a path that looks more similar to this, starting with `C:/` for C drive if you run `getwd()`. There may be spaces in the folder names too:

`C:/Users/melinda/Desktop/ITEP_tutorials/tutorials`

If you need to change the working directory to a path that has spaces in it, use a \ to "escape" the space and treat it as a character like so: `C:/Users/melinda\ ronca-battista/Desktop/ITEP_tutorials/tutorials` In an R script, you can verify your working directory with `getwd()` to see the folder you are working from. `setwd()` needs to be run from each code chunk if you need to load or write data with a command in that chunk.

```
# pretend this code chunk is an r script
# tells you where you are currently located
getwd()
```

```
## [1] "/home/rave/ITEP_tutorials/tutorials"
```

```
# for me this command returns "/home/rave/ITEP_tutorials/tutorials"
# I want to change this to "/home/rave/ITEP_tutorials/ since this is where my data folder is located, i

setwd("/home/rave/ITEP_tutorials")
air_data <- read_excel("data/la_jolla_pm25_wind_data.xlsx") # since we changed the working directory, n
```

```
## New names:
## * Date -> Date...1
## * Date -> Date...6
```

```
print(c("The working directory for this chunk is: ", getwd()))
```

```
## [1] "The working directory for this chunk is: "
## [2] "/home/rave/ITEP_tutorials"
```

Fixing headers (names)

When we look at the column names of the table we read in, we see some problems. There are backslashes \ to before the quotes. This is because when the `read_excel` function interprets the column names, it decides to put a backslash in front of the double quotes to signify that the double quote is part of the name. There's also quite a few unnecessary characters such as `__`, `+`, `'`, and `...` in the column names.

We can clean these column names with R and reuse these scripts in the future, rather than spending time cleaning multiple files like this one in excel.

```
names(air_data)
```

```
## [1] "Date...1"                   "\"Date+Time\""
## [3] "PM2.5 Conc__(ug/m3)"      "Wind'Speed (miles/h)"
## [5] "Wind Direction (Degrees)" "Date...6"
## [7] "% RH"                       "error"
## [9] "technician"
```

```
old_names <- names(air_data)  # make object of the old names

old_names # this new object shows up in the environment panel in upper right
```

```
## [1] "Date...1"                   "\"Date+Time\""
## [3] "PM2.5 Conc__(ug/m3)"      "Wind'Speed (miles/h)"
## [5] "Wind Direction (Degrees)" "Date...6"
## [7] "% RH"                       "error"
## [9] "technician"
```

We will clean the headers (names) using base R functions like `gsub` and `make.names`, to make sure each column name is unique (no duplicate columns), doesn't have unnecessary characters, and that none of the column names conflict with reserved words in R.

We need to load dplyr to use a pipe, which is like a function, and can be thought of as: the input on the left side of the pipe operator `%>%` goes to the next function. Or `x %>% function(x)`

```
p_load("dplyr")

new_names <- old_names %>%    # make object of the new names to clean up

  gsub("'", "", .)     %>%    # remove single quotation marks

  gsub("\"", "", .) %>%       # remove double quotation marks

  gsub("%", "percent", .) %>% # change the symbol % to the word percent

  gsub("^[ ]+", "", .) %>%    # this finds any plus signs and removes them
                              # the ^ means at the beginning of the string
                              # [] means extract, then the + tells R
                              # to extract a +, then replace w "nothing""

  make.names(.) %>%

  gsub("[.]+", "_", .) %>% # convert 1 or more periods to a single _

  gsub("[_]+", "_", .) %>% # fix multiple consecutive underscores

  tolower(.)               # make all letter lower case

# see what that did anytime by looking at new_names

new_names
```

```
## [1] "date_1"                  "date_time"
## [3] "pm2_5_conc_ug_m3_"       "windspeed_miles_h_"
## [5] "wind_direction_degrees_" "date_6"
## [7] "percent_rh"              "error"
## [9] "technician"
```

There's still a bit left to clean up, like the trailing underscores and multiple duplicate date columns, but we are closer to having a cleaner table. Even this isn't so bad and we could still use this table for analysis, but maybe not for generating nice looking reports.

```
new_names <- new_names %>%

gsub("_$", "", .) # remove string-final underscores
names_mask <- new_names %in% c("date_1","date_6") # this makes a new vector
names_mask # with TRUE where we want to remove the names and FALSE where we don't
```

```
## [1]  TRUE FALSE FALSE FALSE FALSE  TRUE FALSE FALSE FALSE
```

```
new_names <- new_names[! names_mask] # ! says make FALSE values TRUE and TRUE values FALSE.
new_names
```

```
## [1] "date_time"              "pm2_5_conc_ug_m3"
## [3] "windspeed_miles_h"      "wind_direction_degrees"
## [5] "percent_rh"             "error"
## [7] "technician"
```

Assigning the clean header/column names back to the table

Up to this point we've been modifying a new variable with the column names, not the original table that we read in with `read_excel`. Now we can set the new names.

We need to take out unused columns from air_data with names_mask so that new_names and air_data are the same shape.

```
# we need to use our mask from the previous cell to get the 7 correct columns. We apply the mask in the
air_data_with_correct_columns <- air_data[,! names_mask]
# it's good to make a new variable for the data with clean headers in case we need to refer back to the
air_data_with_clean_header <- setNames(air_data_with_correct_columns, new_names)

head(air_data_with_clean_header)          # to see the first 6 rows
```

```
## # A tibble: 6 x 7
##   date_time           pm2_5_conc_ug_m3 windspeed_miles~ wind_direction_~
##   <dttm>              <chr>                       <dbl> <chr>
## 1 2018-05-01 00:00:00 filter error                    3 68
## 2 2018-05-01 01:00:00 7                               4 68
## 3 2018-05-01 02:00:00 5                               5 68
## 4 2018-05-01 03:00:00 5                               5 68
## 5 2018-05-01 04:00:00 9                               4 68
## 6 2018-05-01 05:00:00 6                               3 135
## # ... with 3 more variables: percent_rh <lgl>, error <chr>,
## #   technician <chr>
```

```
# and to see the whole new file in a new tab in this editor pane

# click on air_data_1 in the upper right environment pane

# Save data as a CSV file to your working directory
```

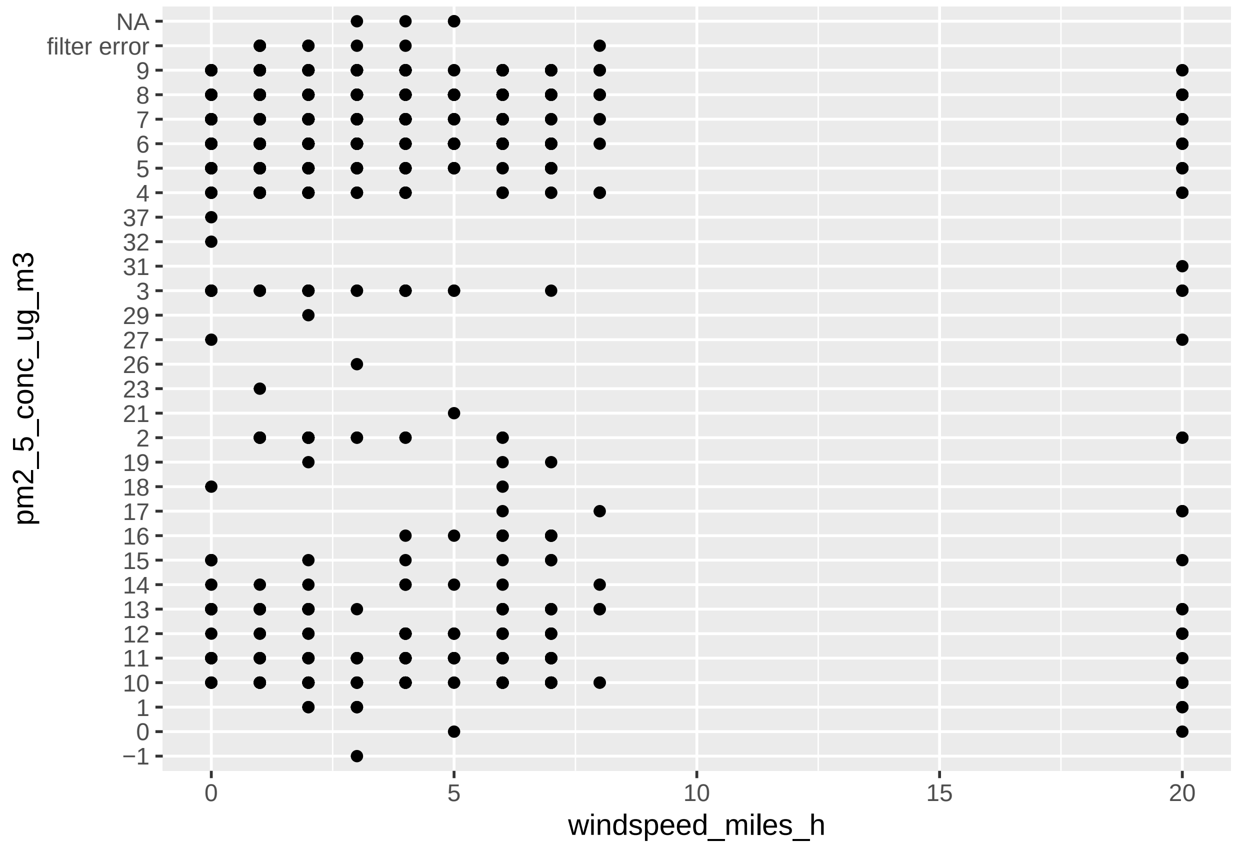
```r
write_csv(air_data_with_clean_header, "../data/air_data_with_clean_header.csv")
```

Cleaning Bad Data Values

We're not done with cleaning our data yet! Even though the column names look presentable, whenever we are dealing with physical measurements we usually have missing data values. In our case, these missing data values are reported differently depending on the column, and even change the type of the column from a number to a character. The type change can ruin plots if the plotting is expecting a numerical variable.

```r
p_load("ggplot2")

ggplot(data=air_data_with_clean_header) +
  geom_point(aes(x = windspeed_miles_h, y = pm2_5_conc_ug_m3))
```



If we look at the variable `air_data_with_clean_header`, we see that there are quite a few cells with "filter error" instead of numbers. This makes the `pm2_5_conc_ug_m3` column have a `chr` type for "character" instead of a `dbl` type for float number. We will need to change this to a numerical nodata value in order to plot it or do calculations with it. From our plot, we also see that there are multiple values representing no data in this column, which isn't best practice.

With the `dplyr mutate` function, we can set the column pm2_5_conc_ug_m3 to the output of a function. In this case we will set it to the output of `as.numeric(pm2_5_conc_ug_m3)`

```r
p_load("dplyr")
air_data_with_clean_header
```

```
## # A tibble: 733 x 7
##    date_time           pm2_5_conc_ug_m3 windspeed_miles~ wind_direction_~
##    <dttm>              <chr>                       <dbl> <chr>
##  1 2018-05-01 00:00:00 filter error                    3 68
##  2 2018-05-01 01:00:00 7                               4 68
##  3 2018-05-01 02:00:00 5                               5 68
##  4 2018-05-01 03:00:00 5                               5 68
##  5 2018-05-01 04:00:00 9                               4 68
##  6 2018-05-01 05:00:00 6                               3 135
##  7 2018-05-01 06:00:00 6                               3 68
##  8 2018-05-01 07:00:00 8                               3 113
##  9 2018-05-01 08:00:00 11                              2 90
## 10 2018-05-01 09:00:00 9                               2 68
## # ... with 723 more rows, and 3 more variables: percent_rh <lgl>,
## #   error <chr>, technician <chr>
```

```r
air_data_pm_dbl <- air_data_with_clean_header %>% mutate(pm2_5_conc_ug_m3 = as.numeric(pm2_5_conc_ug_m3
```

```
## Warning: NAs introduced by coercion
```

```r
air_data_pm_dbl
```

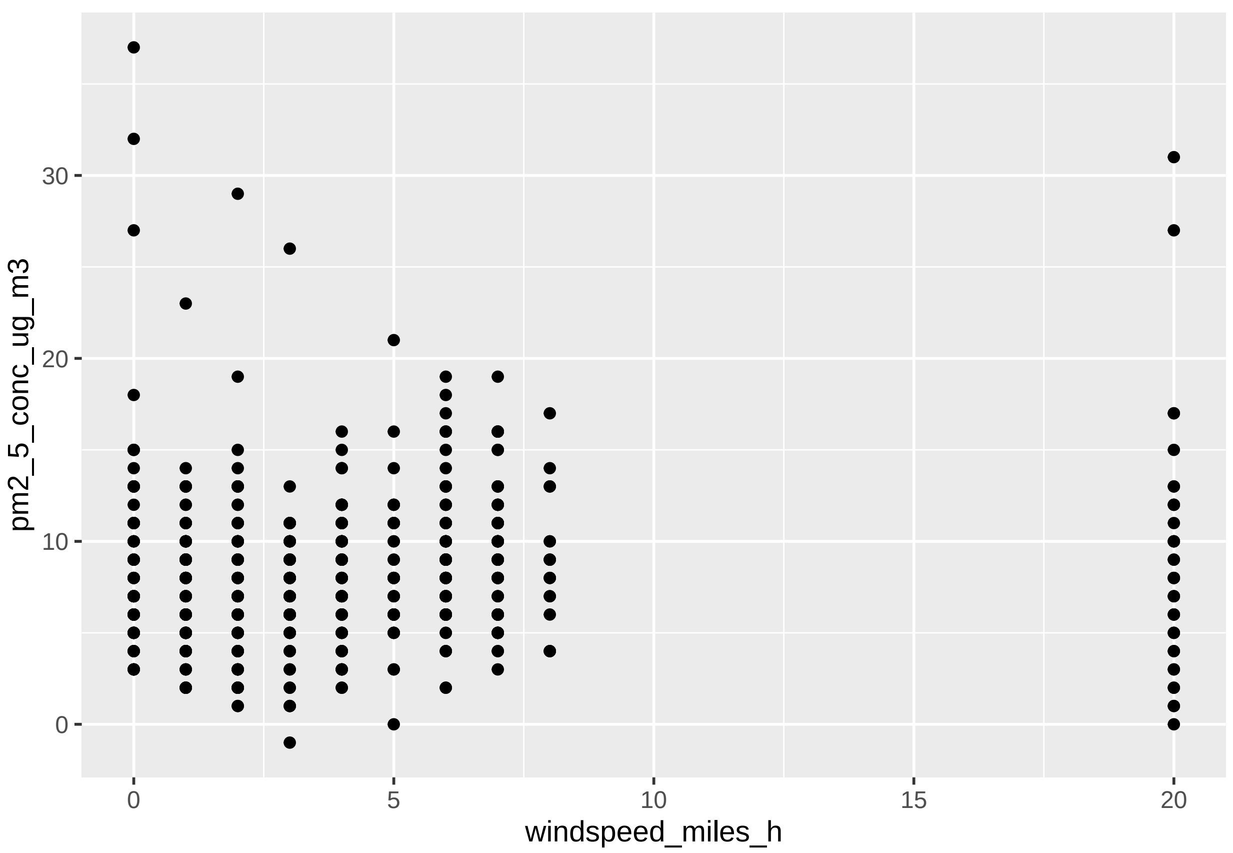
```
## # A tibble: 733 x 7
##    date_time           pm2_5_conc_ug_m3 windspeed_miles~ wind_direction_~
##    <dttm>                         <dbl>            <dbl> <chr>
##  1 2018-05-01 00:00:00               NA                3 68
##  2 2018-05-01 01:00:00                7                4 68
##  3 2018-05-01 02:00:00                5                5 68
##  4 2018-05-01 03:00:00                5                5 68
##  5 2018-05-01 04:00:00                9                4 68
##  6 2018-05-01 05:00:00                6                3 135
##  7 2018-05-01 06:00:00                6                3 68
##  8 2018-05-01 07:00:00                8                3 113
##  9 2018-05-01 08:00:00               11                2 90
## 10 2018-05-01 09:00:00                9                2 68
## # ... with 723 more rows, and 3 more variables: percent_rh <lgl>,
## #   error <chr>, technician <chr>
```

We get a warning in red saying "NAs introduced by coercion". "coercion" is when R follows some rules to convert values from one type to another. In this case, the `as.numeric` function will automatically change any values with non-numeric characters like "filter error" to NA values.

Let's try to make our plot again now that the data have the correct type.

```r
ggplot(air_data_pm_dbl, aes(x=windspeed_miles_h, y = pm2_5_conc_ug_m3)) +
  geom_point()
```

```
## Warning: Removed 13 rows containing missing values (geom_point).
```



```
air_data_pm_dbl
```

```
## # A tibble: 733 x 7
##    date_time           pm2_5_conc_ug_m3 windspeed_miles~ wind_direction_~
##    <dttm>                         <dbl>            <dbl> <chr>
##  1 2018-05-01 00:00:00               NA                3 68
##  2 2018-05-01 01:00:00                7                4 68
##  3 2018-05-01 02:00:00                5                5 68
##  4 2018-05-01 03:00:00                5                5 68
##  5 2018-05-01 04:00:00                9                4 68
##  6 2018-05-01 05:00:00                6                3 135
##  7 2018-05-01 06:00:00                6                3 68
##  8 2018-05-01 07:00:00                8                3 113
##  9 2018-05-01 08:00:00               11                2 90
## 10 2018-05-01 09:00:00                9                2 68
## # ... with 723 more rows, and 3 more variables: percent_rh <lgl>,
## #   error <chr>, technician <chr>
```

Challenge: plot wind direction on the x axis and pm2_5_conc_ug_m3 on the y axis. You may need to mutate the type of wind direction!