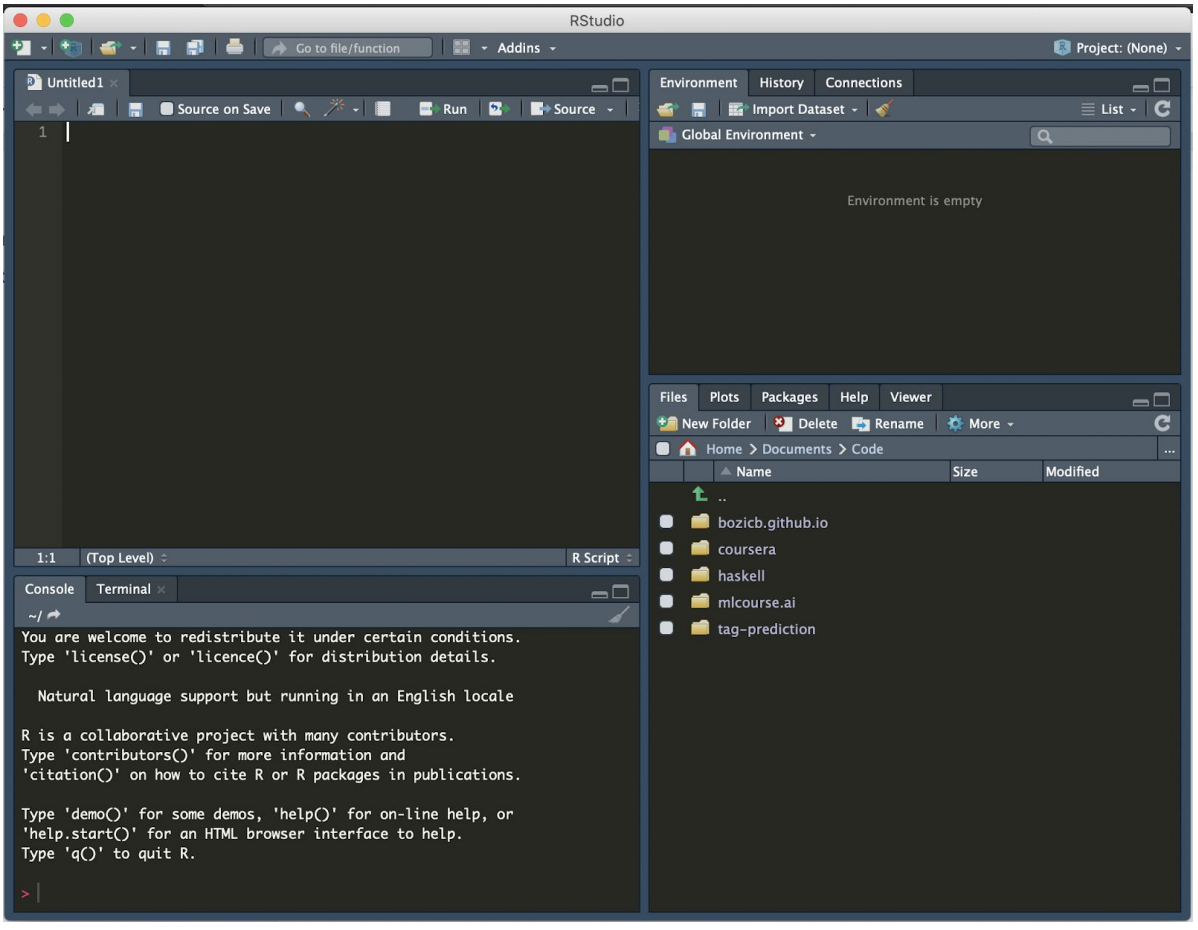
Working with Data - Lab Sheet 1

Source: https://moderndive.netlify.app/

Install R Studio

1. Download and install R by going to https://cloud.r-project.org/.
    - If you are a Windows user: Click on “Download R for Windows”, then click on “base”, then click on the Download link.
    - If you are macOS user: Click on “Download R for (Mac) OS X”, then under “Latest release:” click on R-X.X.X.pkg, where R-X.X.X is the version number. For example, the latest version of R as of November 25, 2019 was R-3.6.1.
    - If you are a Linux user: Click on “Download R for Linux” and choose your distribution for more information on installing R for your setup.
2. Download and install RStudio at https://www.rstudio.com/products/rstudio/download/.
    - Scroll down to “Installers for Supported Platforms” near the bottom of the page.
    - Click on the download link corresponding to your computer’s operating system.
3. Start up R Studio and you should see something like this:



Using R via R Studio

Basic Programming

You can use console pane to try the following commands:

1. Basics:
    a. *Console pane*: where you enter in commands.
    b. *Running code*: the act of telling R to perform an act by giving it commands in the console.
    c. *Objects*: where values are saved in R. We'll show you how to *assign* values to objects and how to display the contents of objects.
    d. *Data types*: integers, doubles/numerics, logicals, and characters. Integers are values like `-1, 0, 2, 4092`. Doubles or numerics are a larger set of values containing both the integers but also fractions and decimal values like `-24.932` and `0.8`. Logicals are either TRUE or FALSE while characters are text such as "cabbage", "Hamilton", "The Wire is the greatest TV show ever", and "This ramen is delicious." Note that characters are often denoted with the quotation marks around them.
2. *Vectors*: a series of values. These are created using the `c()` function, where `c()` stands for "combine" or "concatenate." For example, `c(6, 11, 13, 31, 90, 92)` creates a six element series of positive integer values .
3. *Factors*: *categorical data* are commonly represented in R as factors. Categorical data can also be represented as *strings*.
4. *Data frames*: rectangular spreadsheets. They are representations of datasets in R where the rows correspond to *observations* and the columns correspond to *variables* that describe the observations.
5. *Conditionals*:
    a. Testing for equality in R using `==` (and not =, which is typically used for assignment). For example, `2 + 1 == 3` compares 2 + 1 to 3 and is correct R code, while `2 + 1 = 3` will return an error.
    b. Boolean algebra: TRUE/FALSE statements and mathematical operators such as `<` (less than), `<=` (less than or equal), and `!=` (not equal to). For example, `4 + 2 >= 3` will return `TRUE`, but `3 + 5 <= 1` will return `FALSE`.
    c. Logical operators: & representing "and" as well as | representing "or." For example, `(2 + 1 == 3) & (2 + 1 == 4)` returns `FALSE` since both clauses are not `TRUE` (only the first clause is `TRUE`). On the other hand, `(2 + 1 == 3) | (2 + 1 == 4)` returns `TRUE` since at least one of the two clauses is `TRUE`.
6. *Functions*, also called *commands*: Functions perform tasks in R. They take in inputs called *arguments* and return outputs. You can either manually specify a function's arguments or use the function's *default values*.
    a. For example, the function `seq()` in R generates a sequence of numbers. If you just run `seq()` it will return the value 1. That doesn't seem very useful! This is because the default arguments are set as `seq(from = 1, to =`

`1)`. Thus, if you don't pass in different values for from and to to change this behavior, R just assumes all you want is the number 1. You can change the argument values by updating the values after the = sign. If we try out `seq(from = 2, to = 5)` we get the result 2 3 4 5 that we might expect.

Errors, warnings, and messages

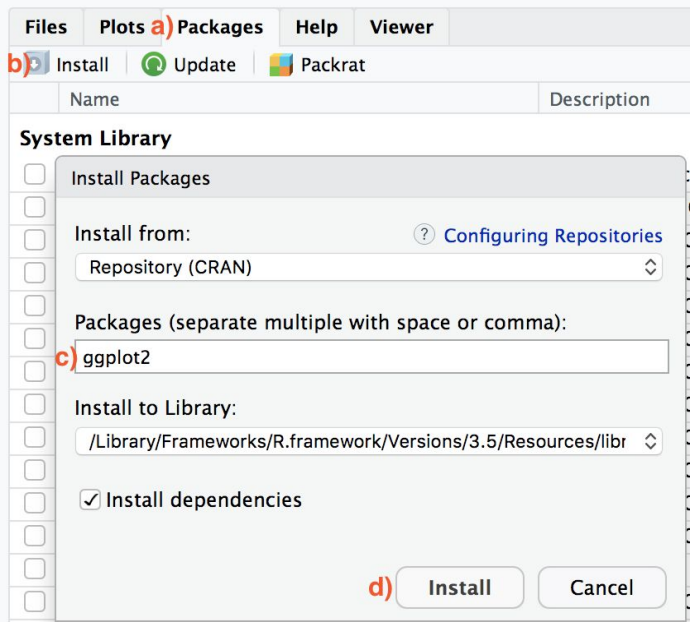
R will show red text in the console pane in three different situations:

- **Errors**: When the red text is a legitimate error, it will be prefaced with "Error in…" and will try to explain what went wrong. Generally when there's an error, the code will not run. For example, if you see Error in `ggplot(...)` : could not find function "ggplot", it means that the ggplot() function is not accessible because the package that contains the function (ggplot2) was not loaded with library(ggplot2). Thus you cannot use the `ggplot()` function without the ggplot2 package being loaded first.
- **Warnings**: When the red text is a warning, it will be prefaced with "Warning:" and R will try to explain why there's a warning. Generally your code will still work, but with some caveats. For example, if you create a scatterplot based on a dataset where two of the rows of data have missing entries that would be needed to create points in the scatterplot, you will see this warning: Warning: Removed 2 rows containing missing values (geom_point). R will still produce the scatterplot with all the remaining non-missing values, but it is warning you that two of the points aren't there.
- **Messages**: When the red text doesn't start with either "Error" or "Warning", it's *just a friendly message*. You'll see these messages when you load *R packages* or when you read data saved in spreadsheet files with the `read_csv()` function. These are helpful diagnostic messages and they don't stop your code from working. Additionally, you'll see these messages when you install packages too using `install.packages()`.

Package Installation

There are two ways to install an R package: an easy way and a more advanced way. Let's install the ggplot2 package the easy way. In the Files pane of RStudio:

a. Click on the "Packages" tab.
b. Click on "Install" next to Update.
c. Type the name of the package under "Packages (separate multiple with space or comma):" In this case, type ggplot2.
d. Click "Install."



An alternative but slightly less convenient way to install a package is by typing install.packages("ggplot2") in the console pane of RStudio and pressing Return/Enter on your keyboard. Note you must include the quotation marks around the name of the package.

TASK: Repeat the earlier installation steps, but for the dplyr, nycflights13, and knitr packages. This will install the earlier mentioned dplyr package for data wrangling, the nycflights13 package containing data on all domestic flights leaving a NYC airport in 2013, and the knitr package for generating easy-to-read tables in R. We'll use these packages in the next section.

Package loading

Recall that after you've installed a package, you need to "load it." In other words, you need to "open it." We do this by using the library() command.

For example, to load the ggplot2 package, run the following code in the console pane. What do we mean by "run the following code"? Either type or copy-and-paste the following code into the console pane and then hit the Enter key.

```
library(ggplot2)
```

If after running the earlier code, a blinking cursor returns next to the > "prompt" sign, it means you were successful and the ggplot2 package is now loaded and ready to use. If, however, you get a red "error message" that reads ...

Error in library(ggplot2) : there is no package called 'ggplot2'

... it means that you didn't successfully install it. This is an example of an "error message". If you get this error message, make sure to install the ggplot2 package before proceeding.

TASK: "Load" the dplyr, nycflights13, and knitr packages as well by repeating the earlier steps.

Package Use

One very common mistake new R users make when wanting to use particular packages is they forget to "load" them first by using the `library()` command we just saw. Remember: *you have to load each package you want to use every time you start RStudio.* If you don't first "load" a package, but attempt to use one of its features, you'll see an error message similar to:

Error: could not find function

This is a different error message than the one you just saw on a package not having been installed yet. R is telling you that you are trying to use a function in a package that has not yet been "loaded." R doesn't know where to find the function you are using. Almost all new users forget to do this when starting out, and it is a little annoying to get used to doing it.

Explore a Dataset

Let's first load all the packages needed, assuming you've already installed them.

```
library(nycflights13)
library(dplyr)
library(knitr)
```

nycflights13 package

Are there ways that we can understand the reasons that cause flight delays?
We'd all like to arrive at our destinations on time whenever possible. We're going to analyze data related to all domestic flights departing from one of New York City's three main airports in 2013: Newark Liberty International (EWR), John F. Kennedy International (JFK), and LaGuardia Airport (LGA). We'll access this data using the nycflights13 R package, which contains five datasets saved in five data frames:

- `flights`: Information on all 336,776 flights.
- `airlines`: A table matching airline names and their two-letter International Air Transport Association (IATA) airline codes (also known as carrier codes) for 16 airline companies. For example, "DL" is the two-letter code for Delta.
- `planes`: Information about each of the 3,322 physical aircraft used.

- `weather`: Hourly meteorological data for each of the three NYC airports. This data frame has 26,115 rows, roughly corresponding to the 365×24×3=26,280 possible hourly measurements one can observe at three locations over the course of a year.
- `airports`: Names, codes, and locations of the 1,458 domestic destinations.

flights data frame

We'll begin by exploring the flights data frame and get an idea of its structure. Run the following code in your console, either by typing it or by cutting-and-pasting it. It displays the contents of the flights data frame in your console. Note that depending on the size of your monitor, the output may vary slightly.

`flights`

```
# A tibble: 336,776 x 19
    year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time
   <int> <int> <int>    <int>          <int>     <dbl>    <int>          <int>
 1  2013     1     1      517            515         2      830            819
 2  2013     1     1      533            529         4      850            830
 3  2013     1     1      542            540         2      923            850
 4  2013     1     1      544            545        -1     1004           1022
 5  2013     1     1      554            600        -6      812            837
 6  2013     1     1      554            558        -4      740            728
 7  2013     1     1      555            600        -5      913            854
 8  2013     1     1      557            600        -3      709            723
 9  2013     1     1      557            600        -3      838            846
10  2013     1     1      558            600        -2      753            745
# … with 336,766 more rows, and 11 more variables: arr_delay <dbl>,
#   carrier <chr>, flight <int>, tailnum <chr>, origin <chr>, dest <chr>,
#   air_time <dbl>, distance <dbl>, hour <dbl>, minute <dbl>, time_hour <dttm>
```

Let's unpack this output:

- `A tibble: 336,776 x 19`: A tibble is a specific kind of data frame in R. This particular data frame has
  - `336,776` rows corresponding to different *observations*. Here, each observation is a flight.
  - `19` columns corresponding to 19 *variables* describing each observation.
- `year`, `month`, `day`, `dep_time`, `sched_dep_time`, `dep_delay`, and `arr_time` are the different columns, in other words, the different variables of this dataset.
- We then have a preview of the first 10 rows of observations corresponding to the first 10 flights. R is only showing the first 10 rows, because if it showed all `336,776` rows, it would overwhelm your screen.

- ... with `336,766` more rows, and `11` more variables: indicating to us that 336,766 more rows of data and 11 more variables could not fit in this screen.

Unfortunately, this output does not allow us to explore the data very well, but it does give a nice preview. Let's look at some different ways to explore data frames.

Exploring Data Frames

There are many ways to get a feel for the data contained in a data frame such as `flights`. We present three functions that take as their "argument" (their input) the data frame in question. We also include a fourth method for exploring one particular column of a data frame:

1. Using the `View()` function, which brings up RStudio's built-in data viewer.
2. Using the `glimpse()` function, which is included in the `dplyr` package.
3. Using the `kable()` function, which is included in the `knitr` package.
4. Using the `$` "extraction operator," which is used to view a single variable/column in a data frame.

1. `View()`:

Run `View(flights)` in your console in RStudio, either by typing it or cutting-and-pasting it into the console pane. Explore this data frame in the resulting pop up viewer. You should get into the habit of viewing any data frames you encounter. Note the uppercase V in `View()`. R is case-sensitive, so you'll get an error message if you run `view(flights)` instead of `View(flights)`.

TASK: What does any *ONE* row in this `flights` dataset refer to?

- **A. Data on an airline**
- **B. Data on a flight**
- **C. Data on an airport**
- **D. Data on multiple flights**

By running `View(flights)`, we can explore the different *variables* listed in the columns. Observe that there are many different types of variables. Some of the variables like `distance`, `day`, and `arr_delay` are what we will call *quantitative* variables. These variables are numerical in nature. Other variables here are *categorical*.

2. `glimpse()`:

The second way we'll cover to explore a data frame is using the `glimpse()` function included in the `dplyr` package. Thus, you can only use the `glimpse()` function after you've loaded the `dplyr` package by running `library(dplyr)`. This function provides us with an alternative perspective for exploring a data frame than the `View()` function:

```
glimpse(flights)
```

```
Rows: 336,776
Columns: 19
$ year           <int> 2013, 2013, 2013, 2013, 2013, 2013, 2013, 2013, 2013, …
$ month          <int> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, …
$ day            <int> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, …
$ dep_time       <int> 517, 533, 542, 544, 554, 554, 555, 557, 557, 558, 558,…
$ sched_dep_time <int> 515, 529, 540, 545, 600, 558, 600, 600, 600, 600, 600,…
$ dep_delay      <dbl> 2, 4, 2, -1, -6, -4, -5, -3, -3, -2, -2, -2, -2, -2, -…
$ arr_time       <int> 830, 850, 923, 1004, 812, 740, 913, 709, 838, 753, 849…
$ sched_arr_time <int> 819, 830, 850, 1022, 837, 728, 854, 723, 846, 745, 851…
$ arr_delay      <dbl> 11, 20, 33, -18, -25, 12, 19, -14, -8, 8, -2, -3, 7, -…
$ carrier        <chr> "UA", "UA", "AA", "B6", "DL", "UA", "B6", "EV", "B6", …
$ flight         <int> 1545, 1714, 1141, 725, 461, 1696, 507, 5708, 79, 301, …
$ tailnum        <chr> "N14228", "N24211", "N619AA", "N804JB", "N668DN", "N39…
$ origin         <chr> "EWR", "LGA", "JFK", "JFK", "LGA", "EWR", "EWR", "LGA"…
$ dest           <chr> "IAH", "IAH", "MIA", "BQN", "ATL", "ORD", "FLL", "IAD"…
$ air_time       <dbl> 227, 227, 160, 183, 116, 150, 158, 53, 140, 138, 149, …
$ distance       <dbl> 1400, 1416, 1089, 1576, 762, 719, 1065, 229, 944, 733,…
$ hour           <dbl> 5, 5, 5, 5, 6, 5, 6, 6, 6, 6, 6, 6, 6, 6, 5, 6, 6, …
$ minute         <dbl> 15, 29, 40, 45, 0, 58, 0, 0, 0, 0, 0, 0, 0, 0, 59, …
$ time_hour      <dttm> 2013-01-01 05:00:00, 2013-01-01 05:00:00, 2013-01-01 …
```

Observe that `glimpse()` will give you the first few entries of each variable in a row after the variable name. In addition, the *data type* of the variable is given immediately after each variable's name inside `< >`. Here, `int` and `dbl` refer to "integer" and "double", which are computer coding terminology for quantitative/numerical variables. "Doubles" take up twice the size to store on a computer compared to integers.

In contrast, `chr` refers to "character", which is computer terminology for text data. In most forms, text data, such as the `carrier` or `origin` of a flight, are categorical variables. The `time_hour` variable is another data type: `dttm`.

TASK: What are some other examples in this dataset of *categorical* variables? What makes them different than *quantitative* variables?

3. `kable()`:

The final way to explore the entirety of a data frame is using the `kable()` function from the `knitr` package. Let's explore the different carrier codes for all the airlines in our dataset two ways. Run both of these lines of code in the console:

`airlines`

```
kable(airlines)
```

At first glance, it may not appear that there is much difference in the outputs. However, when using tools for producing reproducible reports such as R Markdown, the latter code produces output that is much more legible and reader-friendly.

4. `$` operator

Lastly, the `$` operator allows us to extract and then explore a single variable within a data frame. For example, run the following in your console

```
airlines$name
```

We used the `$` operator to extract only the `name` variable and return it as a vector of length 16. We'll only be occasionally exploring data frames using the `$` operator, instead favoring the `View()` and `glimpse()` functions.

Identification and measurement variables

There is a subtle difference between the kinds of variables that you will encounter in data frames. There are *identification variables* and *measurement variables*. For example, let's explore the `airports` data frame by showing the output of `glimpse(airports)`:

```
glimpse(airports)
```

```
Rows: 1,458
Columns: 8
$ faa   <chr> "04G", "06A", "06C", "06N", "09J", "0A9", "0G6", "0G7", "0P2", …
$ name  <chr> "Lansdowne Airport", "Moton Field Municipal Airport", "Schaumbu…
$ lat   <dbl> 41.1, 32.5, 42.0, 41.4, 31.1, 36.4, 41.5, 42.9, 39.8, 48.1, 39.…
$ lon   <dbl> -80.6, -85.7, -88.1, -74.4, -81.4, -82.2, -84.5, -76.8, -76.6, …
$ alt   <dbl> 1044, 264, 801, 523, 11, 1593, 730, 492, 1000, 108, 409, 875, 1…
$ tz    <dbl> -5, -6, -6, -5, -5, -5, -5, -5, -5, -8, -5, -6, -5, -5, -5, -5,…
$ dst   <chr> "A", "A", "A", "A", "A", "A", "A", "A", "U", "A", "A", "U", "A"…
$ tzone <chr> "America/New_York", "America/Chicago", "America/Chicago", "Amer…
```

The variables `faa` and `name` are what we will call *identification variables*, variables that uniquely identify each observational unit. In this case, the identification variables uniquely identify airports. Such variables are mainly used in practice to uniquely identify each row in a data frame. `faa` gives the unique code provided by the FAA for that airport, while the name variable gives the longer official name of the airport. The remaining variables (`lat`, `lon`, `alt`, `tz`, `dst`, `tzone`) are often called *measurement* or *characteristic* variables: variables that describe properties of each observational unit. For example, `lat` and `long` describe the latitude and longitude of each airport.

Furthermore, sometimes a single variable might not be enough to uniquely identify each observational unit: combinations of variables might be needed. While it is not an absolute rule, for organizational purposes it is considered good practice to have your identification variables in the leftmost columns of your data frame.

TASK: What properties of each airport do the variables `lat`, `lon`, `alt`, `tz`, `dst`, and `tzone` describe in the `airports` data frame? Take your best guess.

TASK: Provide the names of variables in a data frame with at least three variables where one of them is an identification variable and the other two are not. Further, create your own tidy data frame that matches these conditions.

Help File

Another nice feature of R are help files, which provide documentation for various functions and datasets. You can bring up help files by adding a `?` before the name of a function or data frame and then run this in the console. You will then be presented with a page showing the corresponding documentation if it exists. For example, let's look at the help file for the `flights` data frame.

```
?flights
```

The help file should pop up in the Help pane of RStudio. If you have questions about a function or data frame included in an R package, you should get in the habit of consulting the help file right away.

TASK: Look at the help file for the `airports` data frame. Revise your earlier guesses about what the variables `lat`, `lon`, `alt`, `tz`, `dst`, and `tzone` each describe.