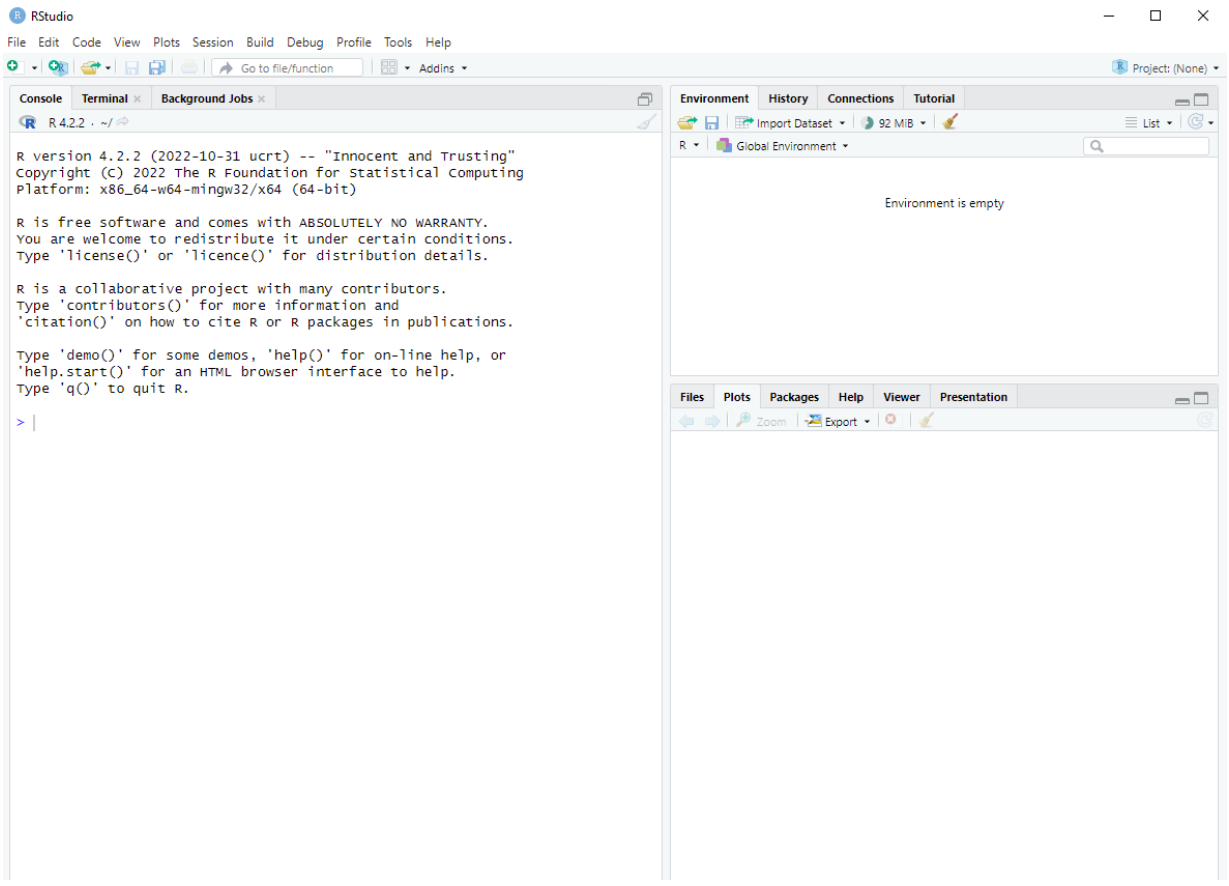
Handout 3 - R

R is a comprehensive statistical and graphical programming language that runs on a variety of platforms including Windows, Unix, and MacOS. Some advantages to R are that it provides a convenient platform for programming new statistical methods, and it has state-of-the-art graphing capabilities. Best of all, it's free!

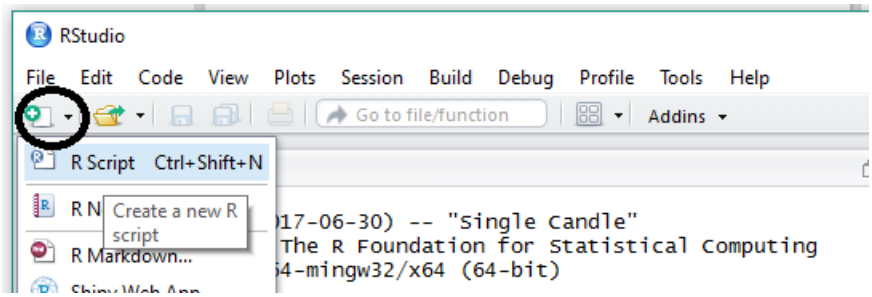
In this course, we will use RStudio which provides a richer user interface. To begin, you'll want to download R and RStudio from this website (https://posit.co/download/rstudio-desktop/). R is different from many other statistical packages in that it contains a very primitive interface (though this is continually improving) and as a result has a more hands-on programming feel than other statistical software packages.

Note: If you already have R and RStudio installed on your machine, you may want to make sure both are the most recent versions of the software.

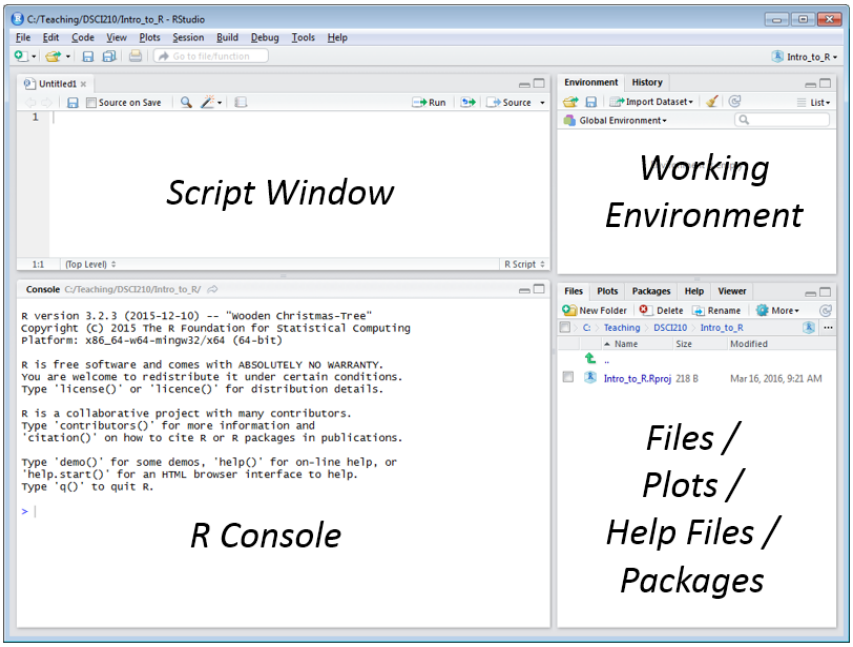
When you start RStudio, the following window should appear:



The frame in the upper-right contains your workspace/working environment. The frame on the left (called the Console Window) is where you can enter commands at the prompt. Instead of entering commands in the console, we can also use the R Script window, which can be accessed as shown below.



If you choose to use a script window (which I recommend), then your RStudio window should appear as follows.



R's base package contains many basic functions used in statistics and is always available during an R session. In addition to the base package, many individuals have created other packages that can be downloaded that will aid in various analyses. Note that R is an open-source package, which has both advantages and disadvantages. Much of the advanced functionality of R exists because of user contributed packages; however, the documentation can be difficult to sift through. Regardless, there are literally *thousands* of documents on the web that can help you use R efficiently (some are much better than others). The following links give some of the most popular webpages for R support.

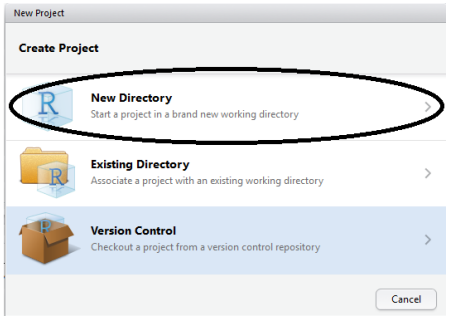
- https://cran.r-project.org/other-docs.html
- https://cran.r-project.org/manuals.html
- https://cran.r-project.org/doc/manuals/R-intro.html

It may take some time to get comfortable with R's command line interface, but after some time, R will become intuitive in its functionality.

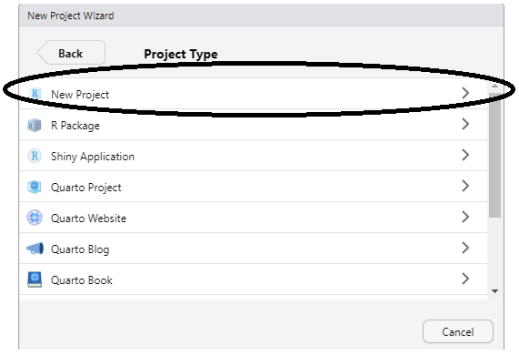
Creating a New Project

In RStudio you can divide your work into projects that each have their own working directory, workspace, history, and source documents. You can create a project in either a new directory or in an existing directory you already have saved R code and data.

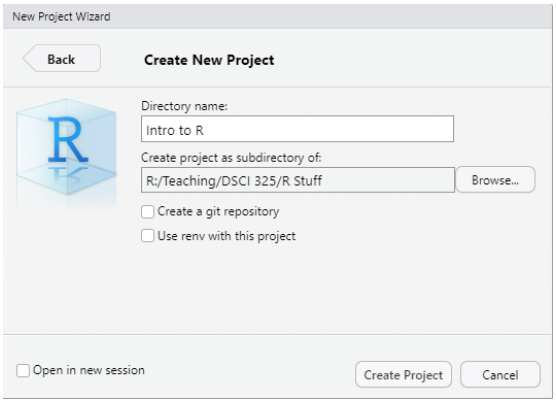
To create a new project, select **File → New Project…** In the dialog box that appears, select "New Directory."



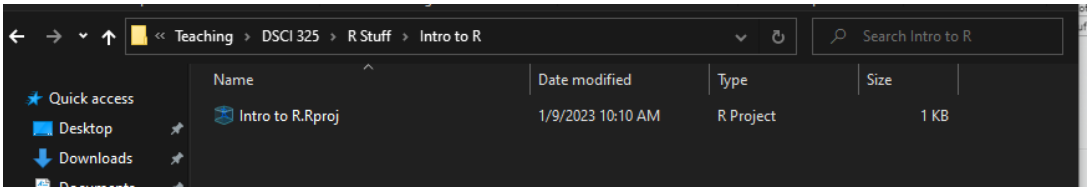
Then, select "New Project."



Next, specify the name and location of the desired directory.



Click "Create Project" and check your directory to verify that the new project has been created.



Basic Operators in R

R uses the following comparison operators.

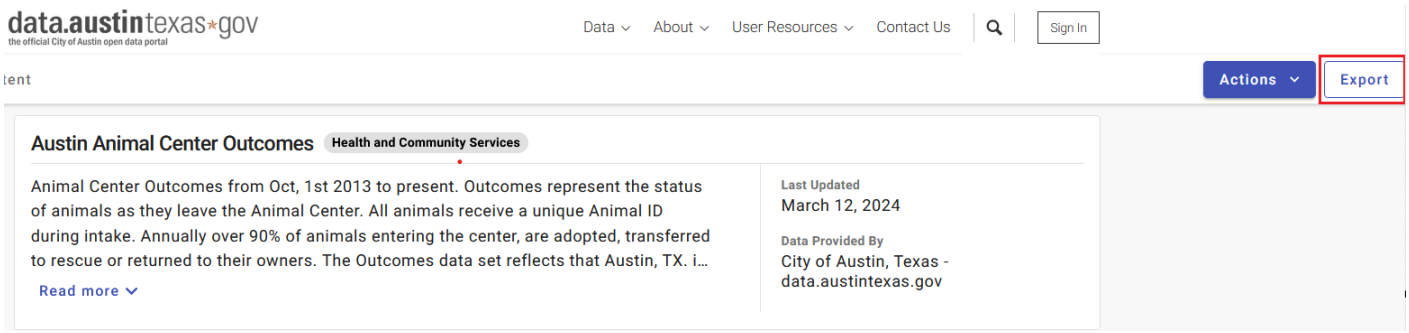
==	Equal
!=	Not equal
>	Greater than
>=	Greater than or equal to
<	Less than
<=	Less than or equal to

R uses the following logical operators.

&	AND returns TRUE if both comparisons return TRUE
|	OR returns TRUE if at least one comparison returns TRUE
!	NOT returns the negation of a logical vector

Example 1: Austin Animal Center

We are going to learn how to read in a datafile using data from the Austin Animal Center in the city of Austin, TX. An open data portal like this is used to provide data to the public. If you click on the link above, it will take you to the following website.

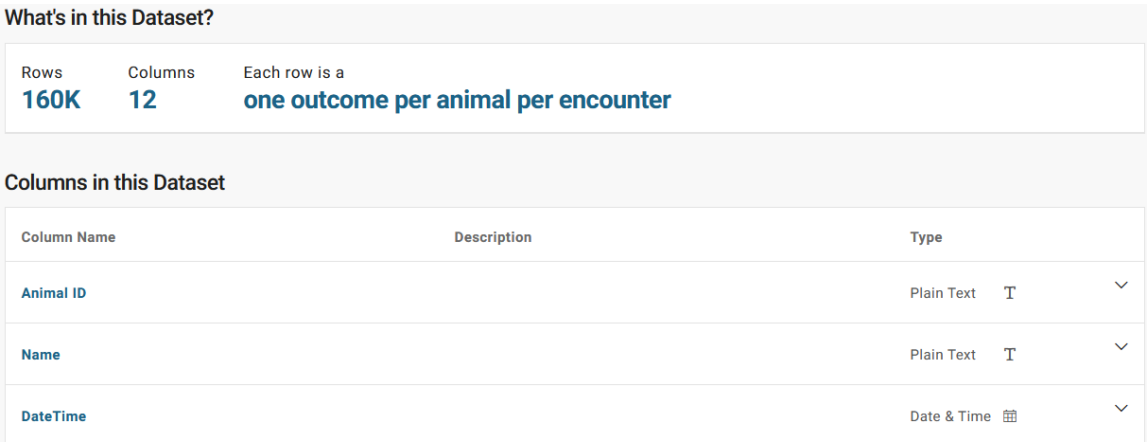


The dataset includes information regarding the status of animals as they leave the Animal Care Center. Click Export to obtain a local copy of this dataset. For this example, a .csv copy will be downloaded.

Details regarding Data via Open Data Portal

The Open Data Portal provides some basic information regarding this dataset. In particular, the number of rows and columns have been provided (sometimes an estimate is provided instead of the exact value). In addition, some basic information regarding the columns provided in this dataset are given.

Definition: A dataset that takes on the form a simple rows and columns is sometimes referred to as data in a **table format**. This data may be referred to as data that is **highly structured**.



What's in this Dataset?

Rows 160K | Columns 12 | Each row is a one outcome per animal per encounter

Columns in this Dataset

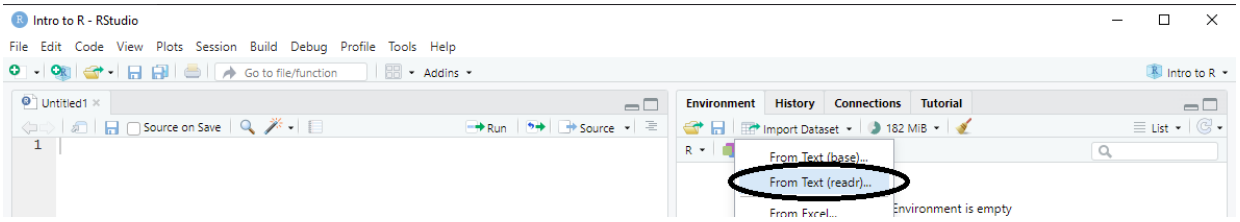
Column Name	Description	Type
Animal ID		Plain Text
Name		Plain Text
DateTime		Date & Time

Once you've saved the data to your machine, it should look something like this.

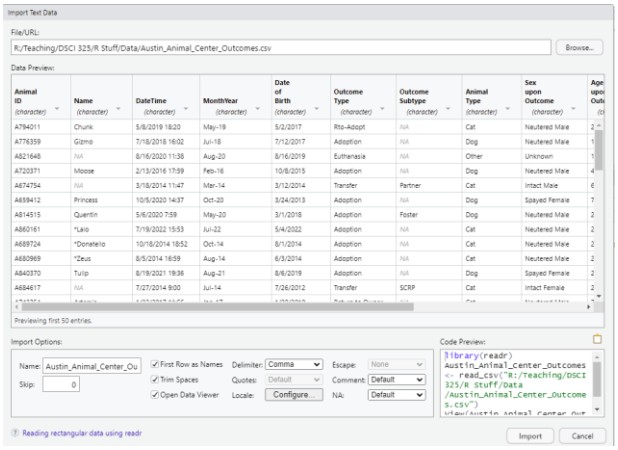
	A	B	C	D	E	F	G	H	I	J	K	L
1	Animal ID	Name	DateTime	MonthYear	Date of Birth	Outcome Type	Outcome Subtype	Animal Type	Sex upon Outcome	Age upon Outcome	Breed	Color
2	A794011	Chunk	5/8/2019 18:20	19-May	5/2/2017	Rto-Adopt		Cat	Neutered Male	2 years	Domestic Shorthair Mix	Brown Tabby/White
3	A776359	Gizmo	7/18/2018 16:02	18-Jul	7/12/2017	Adoption		Dog	Neutered Male	1 year	Chihuahua Shorthair Mix	White/Brown
4	A821648		8/16/2020 11:38	20-Aug	8/16/2019	Euthanasia		Other	Unknown	1 year	Raccoon	Gray
5	A720371	Moose	2/13/2016 17:59	16-Feb	10/8/2015	Adoption		Dog	Neutered Male	4 months	Anatol Shepherd/Labrador Retriever	Buff
6	A674754		3/18/2014 11:47	14-Mar	3/12/2014	Transfer	Partner	Cat	Intact Male	6 days	Domestic Shorthair Mix	Orange Tabby

Method 1:

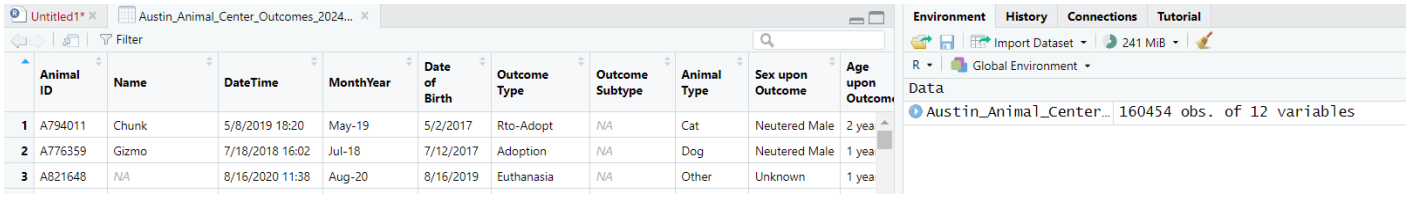
One method for opening this file in R is to select **Import Dataset** in the window in the upper-right-hand corner of the screen. Choose to import data **From Text (readr)…**. (Note: R may have to update a package or two if you've previously installed it before it can import the data set. It should do this automatically.)



Select the text file to be read in (using Browse). The Austin Animal Center data set is being read in here. Give a name to the data set in the Name box. Options may need to be specified when importing data, however the default settings are sufficient for this data.



Click **Import** and the data set will be added to your workspace.



	Animal ID	Name	DateTime	MonthYear	Date of Birth	Outcome Type	Outcome Subtype	Animal Type	Sex upon Outcome	Age upon Outcom
1	A794011	Chunk	5/8/2019 18:20	May-19	5/2/2017	Rto-Adopt	*NA*	Cat	Neutered Male	2 yea
2	A776359	Gizmo	7/18/2018 16:02	Jul-18	7/12/2017	Adoption	*NA*	Dog	Neutered Male	1 yea
3	A821648	*NA*	8/16/2020 11:38	Aug-20	8/16/2019	Euthanasia	*NA*	Other	Unknown	1 yea

Questions:

1. How many observations (i.e., rows) does Austin Animal Center contain?

2. How many variables (i.e., columns) does the Austin Animal Center contain?

Method 2:

Another method for opening this file in using Base R is to use the **read.csv()** command in the script window. This can be accomplished by using the following line of code:

```
AustinAnimalShelter_BaseR <- read.csv("R:/Teaching/325/Notes/R/Handout 3/Austin_Animal_Center_Outcomes_20240312.csv")
```

NOTE: You'll need to change the path to where you saved the file on your computer.

Utility Functions for a data.frame

3. When the dataset was imported into R the following code was shown in the console window. What does this chunk of code do?

```
View(Austin_Animal_Center_Outcomes)
```

4. What does the following code do?

```
head(AustinAnimalShelter_BaseR)
```

5. What does the following code do?

```
tail(AustinAnimalShelter_BaseR)
```

6. What does the following code do?

```
class(AustinAnimalShelter_BaseR)
```

7. What does the following code do?

```
str(AustinAnimalShelter_BaseR)
```

Factors in R

Let's again read in the dataset using the **read.csv()** function in Base R. However, this time we want to tell R to treat all the character variables as factors. This can be accomplished with the following code.

```
AustinAnimalShelter_BaseR_AsFactor <- read.csv("R:/Teaching/325/Notes/R/Handout 3/Austin_Animal_Center_Outcomes_20240312.csv", stringsAsFactors = TRUE)
```

We can again use the **str()** function to see if the above code worked as intended.

```
str(AustinAnimalShelter_BaseR_AsFactor)
```

We needed to change the variables to factors in order to create summaries. Let's try one.

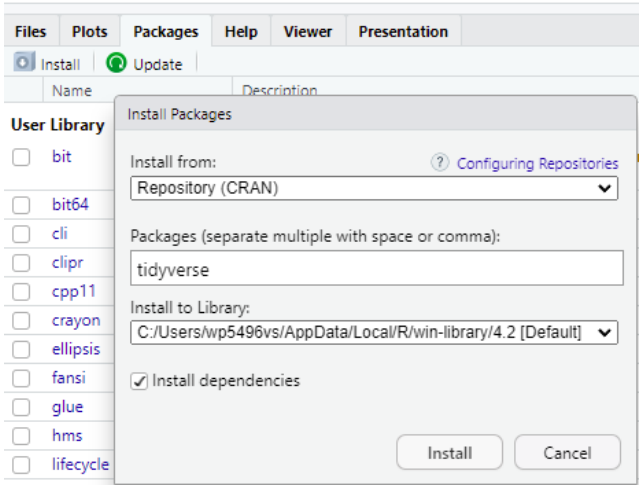
8. Run the following code.

   ```
   AnimalCounts <- table(AustinAnimalShelter_BaseR_AsFactor$Animal.Type)
   barplot(AnimalCounts, xlab="Type of Animal")
   ```

   Briefly describe what each line of this code is doing.

Method 3:

Tidyverse is a collection of R packages designed for data science. The R packages contained in this collection have been written to overcome some of the inherent limitations of base R. For example, Tidyverse uses a modified version of a data.frame. First, the Tidyverse package needs to be installed and then loaded.



1. Click on **Packages** in the bottom right window in RStudio.
2. Click **Install**.
3. Type **Tidyverse** in the packages box in the dialogue box that pops up.
4. Click Install at the bottom of the dialogue box.
5. Run the following code
   ```
   library(tidyverse)
   ```

You should see the following if the Tidyverse package was correctly loaded.

```
── Attaching core tidyverse packages ──────────────────────────────────── tidyverse
2.0.0 ──
✓ dplyr     1.1.4     ✓ purrr     1.0.2
✓ forcats   1.0.0     ✓ stringr   1.5.1
✓ ggplot2   3.5.0     ✓ tibble    3.2.1
✓ lubridate 1.9.3     ✓ tidyr     1.3.1
── Conflicts ────────────────────────────────────────────────────── tidyverse_confli
cts() ──
✖ dplyr::filter() masks stats::filter()
✖ dplyr::lag()    masks stats::lag()
```

Next, read in the Austin Animal Center dataset using the following code. Note that the **read_csv()** function is used.

```
AustinAnimalShelter_Tidyverse <- read_csv("R:/Teaching/325/Notes/R/Handout 3/Austin_Ani
mal_Center_Outcomes_20240312.csv")
```

9. What does the following chunk of code output?

```
ls()
```

10. Run the **class()** function on the dataset that was read into R using read_csv(). What is the variable type?

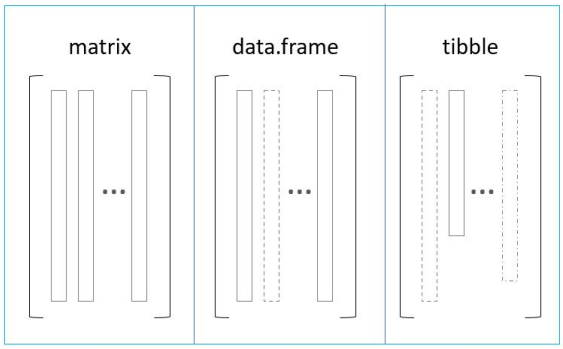
11. Run the **str()** function on the dataset that was read into R using read_csv(). What is the name of the first column? How does this name differ from the data.frame read in from read.csv()? Briefly discuss.

There are three common structures for tabled data in R.

Matrix: A collection of vectors which must all be the same length **and** same data type.

Data.frame: A collection of vectors that must all be the same length.

Tibble – tabled data.frame: A collection of vectors.



Getting Help with R

Within R, you can find help on any command (or find commands) using the following:

- If you know the name of the R function, e.g. summary, use help(summary) or ?summary

```
help(summary)
?summary
```

- If you don't know the function and want to do a keyword search for it, use help.search().

```
help.search("summary")
```

- The help.start() function will launch the full help system that includes all manuals, references, etc.

```
help.start()
```

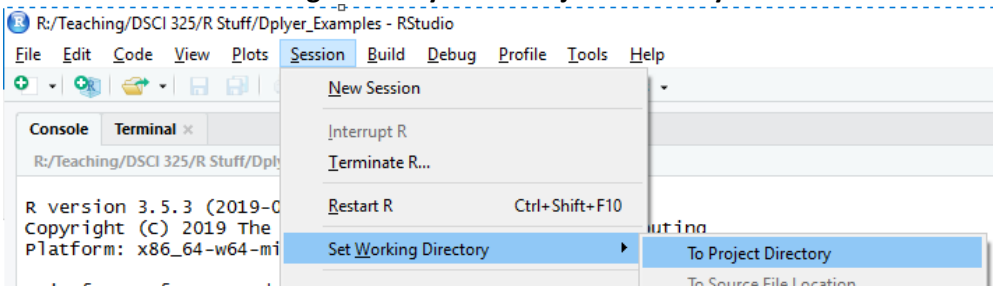
dplyr Package

The dplyr package can be used to manipulate/wrangle data in R. Most of the examples discussed in this handout were taken from the following R documentation: https://cran.rstudio.com/web/packages/dplyr/vignettes/dplyr.html.

As stated in this documentation, the dplyer package provides "simple "verbs", functions that correspond to the most common data manipulation tasks, to help you translate your thoughts into code."

We are again going to use the flight data from the Bureau of Transportation Statistics website which is saved in the file **FlightDelays.csv** on D2L. This time, we are going to import the file using a different process than we've previously done.

- Click on the **Session → Set Working Directory → To Project Directory** as shown below.



- Next, you'll want to copy the code from the R console into an R script so you can access this same file path later. The code I was given is given below as an example.

```
setwd("R:/Teaching/325/Notes/R/Handout 3")
```

- Next, we'll need to read in the file; (1) give it a name using the <- notation, (2) use the read.csv() function, (3) in the parentheses, start typing the name of the file and then hit tab. R should then autofill the name of the data set (which is saved in the working directory).

```
FlightDelays <- read.csv("FlightDelays.csv", stringsAsFactors = FALSE)
```

You'll can either install and load the dplyr package or just load the Tidyverse package:

```
Install.package(dpylr)
Library(dplyr)
```

OR

```
library(tidyverse)
```

Run the *head()* command to double-check your data was loaded correctly. Your output should match the snippet of output given below.

```
  YEAR QUARTER MONTH DAY_OF_MONTH DAY_OF_WEEK   FL_DATE
1 2019       1     1           19           6 1/19/2019
2 2019       1     1           20           7 1/20/2019
3 2019       1     1           21           1 1/21/2019
4 2019       1     1           22           2 1/22/2019
5 2019       1     1           23           3 1/23/2019
6 2019       1     1           24           4 1/24/2019
  OP_UNIQUE_CARRIER OP_CARRIER_AIRLINE_ID OP_CARRIER TAIL_NUM
1                9E                 20363         9E   N176PQ
2                9E                 20363         9E   N309PQ
3                9E                 20363         9E   N176PQ
4                9E                 20363         9E   N135EV
5                9E                 20363         9E   N294PQ
6                9E                 20363         9E   N916XJ
```

Question: What is the purpose of the *names()* command?

```
names(FlightDelays)
```

Next, let's investigate how to implement the various data verbs such as select, filter, mutate, etc. in R. This will be done using the dplyr package.

Tasks:

1. What does the following R code accomplish?

   ```
   filter(FlightDelays, MONTH == 1 & DAY_OF_MONTH == 1)
   ```

   Run the code to verify.

2. With the filter() function you can give any number of filtering conditions which are joined together with "&" or other Boolean operators. For example, consider the following commands using this function. **For each line of code given, describe how the dataset is being filtered.**

   ```
   Filter1 <- filter(FlightDelays, MONTH == 1 & DAY_OF_MONTH == 1)
   Filter2 <- filter(FlightDelays, MONTH ==1 & DAY_OF_MONTH == 1 & OP_CARRIER == "UA")
   Filter3 <- filter(FlightDelays, ORIGIN == "RST" | ORIGIN == "LSE")
   Filter4 <- filter(FlightDelays, (ORIGIN == "RST" | ORIGIN == "LSE") & DEST == "ORD")
   ```

3. How many observations are included for each of the different filters used in Task 2? The following function can be used to determine this.

```
NROW()
```

Filter	Number of observations
1	
2	
3	
4	

4. How does *slice()* differ from *filter()*? Run the following code to explore this.

```
slice(FlightDelays, 1:4)

slice(FlightDelays, c(1:2, 4))
```

5. What is the purpose of the following lines of code?

```
arrange(FlightDelays, DAY_OF_MONTH, OP_CARRIER)

arrange(FlightDelays, ORIGIN, DAY_OF_WEEK)
```

By default, R arranges/orders the columns in ascending order. If you want to order a column in descending order, you'll need to use the *desc()* function.

```
arrange(FlightDelays, desc(ORIGIN), desc(DAY_OF_WEEK))
```

6. Determine the purpose of the *select()* function by running the following lines of code.

```
select(FlightDelays, OP_CARRIER, ORIGIN, DEP_DELAY)

select(FlightDelays, YEAR:DAY_OF_WEEK)
```

7. You can also use various "helper functions" within *select()*. Identify the purpose of each of the helper functions given below.

```
select(FlightDelays, starts_with("OP"))

select(FlightDelays, ends_with("DELAY"))

select(FlightDelays, contains("DELAY"))
```

8. A common use of the *select()* function is to determine how many unique (or distinct) values a variable (or set of variables) takes on.

   a. How many distinct carriers are there?

```
distinct(select(FlightDelays, OP_CARRIER))
```

   b. How many distinct carrier/destination combinations are there?

```
distinct(select(FlightDelays, OP_CARRIER, DEST))
```

9. Next, let's create a new data frame using the following line of code.

```
FlightDelays2 <- select(FlightDelays, DAY_OF_WEEK, OP_CARRIER, DEST, DEP_DELAY, ARR_DELAY, AIR_TIME, DISTANCE)
```

In addition to selecting from existing columns, you can add new columns that are functions of existing functions. Run the following lines of code. Are the new columns automatically added to the data frame?

```
mutate(FlightDelays2, Gain = ARR_DELAY - DEP_DELAY, Speed = DISTANCE/AIR_TIME*60)
head(FlightDelays2)
```

Note that the newly created columns are **NOT** automatically put into the existing data frame. If you want to add these new variables to the data frame, we can do so using the following code.

```
FlightDelays2 <- mutate(FlightDelays2, Gain = ARR_DELAY - DEP_DELAY, Speed = DISTANCE/AIR_TIME*60)
Head(FlightDelays2)
```

10. What happens when the following line of code is run?

```
transmute(FlightDelays2, Gain = ARR_DELAY - DEP_DELAY, Speed = DISTANCE/AIR_TIME*60)
```

11. The *summarize()* function lets you create summaries that collapse a data frame to a single row. For example, consider the following.

```
summarize(FlightDelays2, mean_Dep_Delay = mean(DEP_DELAY))
```

```
> summarize(FlightDelays2, mean_Dep_Delay = mean(DEP_DELAY))
  mean_Dep_Delay
1             NA
```

Notice, we do not get a value returned because there are missing values (NA) in the DEP_DELAY field. We'll need to tell R to remove these values from the calculation.

```
summarize(FlightDelays2, mean_Dep_Delay = mean(DEP_DELAY, na.rm = TRUE))
```

```
> summarize(FlightDelays2, mean_Dep_Delay = mean(DEP_DELAY, na.rm = TRUE))
  mean_Dep_Delay
1       9.766091
```

The na.rm = TRUE gets rid of the missing values (NA) in the data frame. This doesn't change the structure of the data frame; it just removes these observations from the calculation. The downside, per se, is that we don't know how many observations were actually used in the calculation.

This function is more useful when used in conjunction with others (which you will see later on). Finally, note that you can also call this function as follows:

```
summarise(FlightDelays2, mean_Dep_Delay = mean(DEP_DELAY, na.rm = TRUE))
```

```
> summarise(FlightDelays2, mean_Dep_Delay = mean(DEP_DELAY, na.rm = TRUE))
  mean_Dep_Delay
1       9.766091
```

Commonalities of functions in the dplyr package

Note that all of these functions are similar in the following ways:

- The first argument is a data frame
- Subsequent arguments tell R what to do with that data frame
- The result is a new data frame

As stated in the aforementioned R documentation, these five functions together "provide the basis of a language of data manipulation." At the most basic level, we alter data sets in the following ways:

dplyr function	Purpose
filter() or slice()	
arrange()	
select()	
mutate()	
summarize()	

12. Finally, note that you can also use all of the above functions to process a data set "by group." Investigate the following lines of code. Provide a snippet of the output from each line(s) of code.

```
group.carrier <- group_by(FlightDelays2, OP_CARRIER)
summarize(group.carrier, mean(DEP_DELAY, na.rm = TRUE))


summarize(group.carrier, mean_Dep_Delay = mean(DEP_DELAY, na.rm = TRUE), mean_A
rr_Delay = mean(ARR_DELAY, na.rm = TRUE))


summarize(group.carrier, Count = n())
```

Other common summaries

Some of the other common summaries that are utilized are:

- Standard deviation – sd()
- Minimum – min()
- Maximum – max()
- Count – n()