IT137IU: Data Analysis
Lab#2/Assignment#2: Data Wrangling

Introduction

In this guide you will learn how to download, clean and manage data - a process known as Data Wrangling - in R. You will be working with data on Housing Data - Zillow Research. The objectives of this guide are as follows

1. Learn how to read data into R
2. Learn about tidyverse
3. Learn data wrangling functions

This lab guide follows closely and supplements the material presented in Chapters 3, 7-9, and 14 in the textbook R for Data Science (RDS).

R Packages

At the end of Lab 1 we learned about R functions. Functions do not exist in a vacuum, but exist within R packages. Packages are the fundamental units of reproducible R code. They include R functions, the documentation that describes how to use them, and sample data. At the top left of a function's help documentation, you'll find in curly brackets the R package that the function is housed in. For example, type in your console `? seq`. At the top left of the help documentation, you'll find that `seq()` is in the package base. All the functions we have used so far are part of packages that have been pre-installed and pre-loaded into R. For all new packages, you need to install and load them.

In this lab and all labs from here on out, we will be using commands from the package **tidyverse**. Tidyverse is a collection of high-powered, consistent, and easy-to-use packages developed by a number of thoughtful and talented R developers. In order to use functions in a new package, you first need to install the package using the `install.packages()` command.

```
install.packages("tidyverse")
```

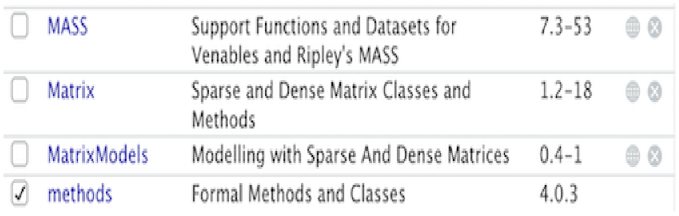
You should see a bunch of gobbledygook roll through your console screen. Don't worry, that's just R downloading all of the other packages and applications that **tidyverse** relies on. These are known as dependencies. Unless you get a message in red that indicates there is an error (like we saw in Lab 1 when we typed in hello world without quotes), you should be fine.

Next, you will need to load the package into your working environment. You need to do this every time you start RStudio. We do this using the `library()` function. Notice there are no quotes around **tidyverse** this time.

```
library(tidyverse)
```

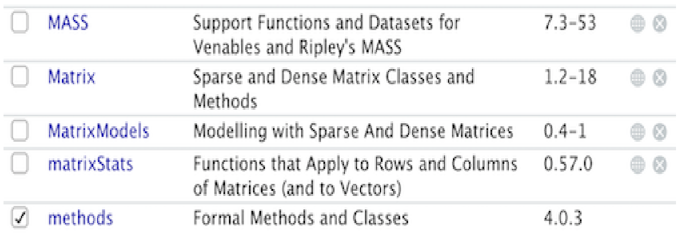
The Packages window at the lower-right of your RStudio shows all the packages you currently have installed. If you don't have a package listed in this window, you'll need to use the `install.packages()` function to install it. If the package is checked, that means it is loaded into your current R session.

For example, here is a section of my Packages window



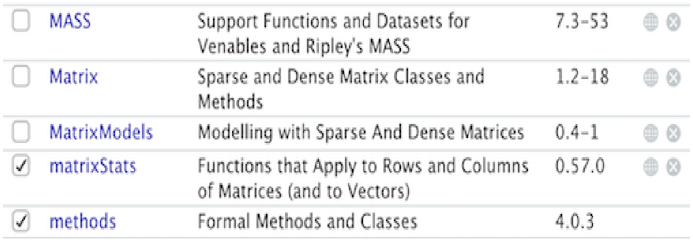
☐	MASS	Support Functions and Datasets for Venables and Ripley's MASS	7.3-53
☐	Matrix	Sparse and Dense Matrix Classes and Methods	1.2-18
☐	MatrixModels	Modelling with Sparse And Dense Matrices	0.4-1
✓	methods	Formal Methods and Classes	4.0.3

The only packages loaded into my current session is **methods**, a package that is loaded every time you open an R session. Let's say we use `install.packages()` to install the package **matrixStats**. The window now looks like



☐	MASS	Support Functions and Datasets for Venables and Ripley's MASS	7.3-53
☐	Matrix	Sparse and Dense Matrix Classes and Methods	1.2-18
☐	MatrixModels	Modelling with Sparse And Dense Matrices	0.4-1
☐	matrixStats	Functions that Apply to Rows and Columns of Matrices (and to Vectors)	0.57.0
✓	methods	Formal Methods and Classes	4.0.3

When you load it in using `library()`, a check mark appears next to **matrixStats**



☐	MASS	Support Functions and Datasets for Venables and Ripley's MASS	7.3-53
☐	Matrix	Sparse and Dense Matrix Classes and Methods	1.2-18
☐	MatrixModels	Modelling with Sparse And Dense Matrices	0.4-1
✓	matrixStats	Functions that Apply to Rows and Columns of Matrices (and to Vectors)	0.57.0
✓	methods	Formal Methods and Classes	4.0.3

Note that you only need to install packages once (`install.packages()`), **but you need to load them each time you relaunch RStudio** (`library()`). Repeat after me: Install once, library every time. This means your R Markdown should never have the function `install.packages()` but will likely always have the function `library()`.

Are you ready to enter the **tidyverse**? Yes, of course, so here is a badge! Wear it proudly my young friends! And away we go!

Reading data

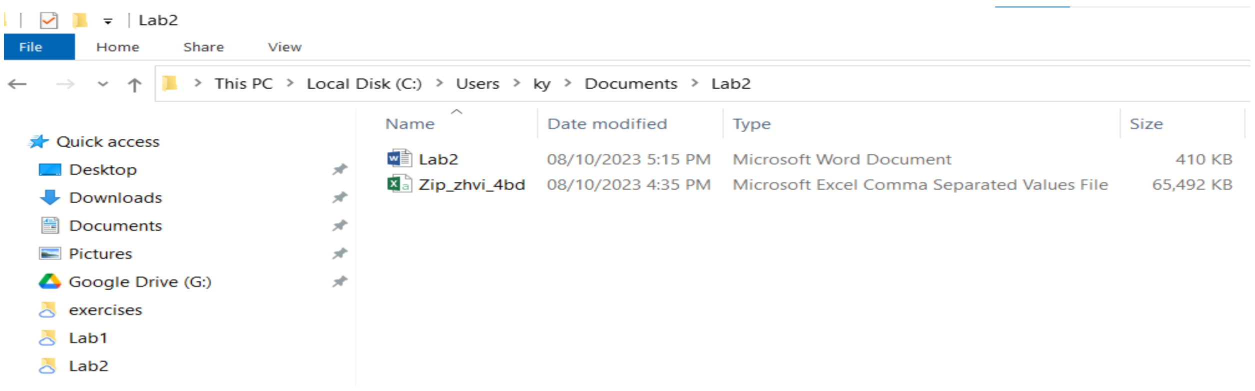
One of the first steps in the Data Wrangling process is to read in your dataset. Most of the data files you will encounter are comma-delimited (or comma-separated) files, which have `.csv` extensions. Comma-delimited means that columns are separated by commas. I uploaded a file on Blackboard containing housing price data at the metro, city and zipcode levels taken from Zillow Group Housing Data - Zillow Research. Download the file ***Zip_zhvi_4bd.csv*** and save it into the same folder where your Lab 2 file resides.

To read in the csv file, first make sure that R is pointed to the folder you saved your data into. Type in `getwd()` to find out the current directory.

```
getwd()
```

and `setwd("directory name")` to set the directory to the folder containing the data, where "directory name" is the path to the folder on your hard drive containing this lab's data.

In my Window OS computer, the file is located in the folder shown in the figure below.



Name	Date modified	Type	Size
Lab2	08/10/2023 5:15 PM	Microsoft Word Document	410 KB
Zip_zhvi_4bd	08/10/2023 4:35 PM	Microsoft Excel Comma Separated Values File	65,492 KB

Using a Window laptop, I type in the following command to set the directory to the folder containing my data.

```
setwd("C:/Users/ky/Documents/Lab2")
```

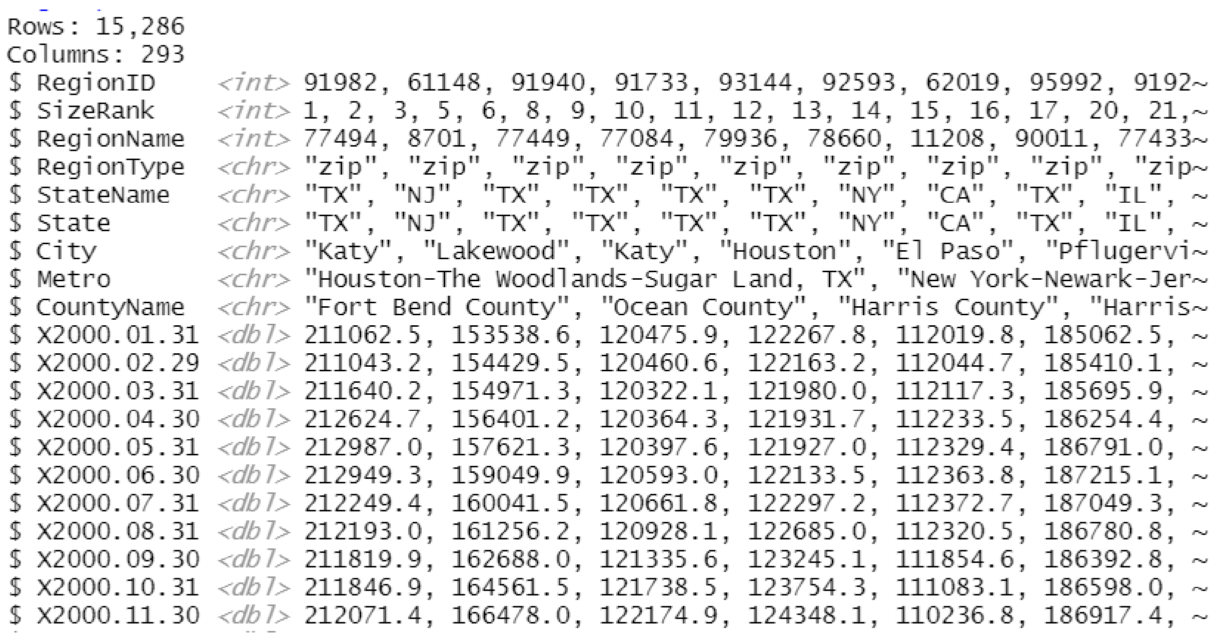
You can also manually set the working directory by clicking on Session -> Set Working Directory -> Choose Directory.

Once you've set your directory, use the function `read_csv()` and plug in the name of the file in quotes inside the parentheses. Make sure you include the `.csv` extension.

```
data<-read.csv("Zip_zhvi_4bd.csv",header = TRUE, stringsAsFactors = FALSE)
```

You should see the object `data` pop up in your Environment window (top right). First thing you should do when you bring in a data set is to look at it to make sure you know what you have and nothing went drastically wrong when reading in the data. One way to look at the data is to print it on your screen. Do this using the `glimpse()` command

```
glimpse(data)
```



```
Rows: 15,286
Columns: 293
$ RegionID     <int> 91982, 61148, 91940, 91733, 93144, 92593, 62019, 95992, 9192~
$ SizeRank     <int> 1, 2, 3, 5, 6, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 20, 21,~
$ RegionName   <int> 77494, 8701, 77449, 77084, 79936, 78660, 11208, 90011, 77433~
$ RegionType   <chr> "zip", "zip", "zip", "zip", "zip", "zip", "zip", "zip", "zip~
$ StateName    <chr> "TX", "NJ", "TX", "TX", "TX", "TX", "NY", "CA", "TX", "IL", ~
$ State        <chr> "TX", "NJ", "TX", "TX", "TX", "TX", "NY", "CA", "TX", "IL", ~
$ City         <chr> "Katy", "Lakewood", "Katy", "Houston", "El Paso", "Pflugervi~
$ Metro        <chr> "Houston-The Woodlands-Sugar Land, TX", "New York-Newark-Jer~
$ CountyName   <chr> "Fort Bend County", "Ocean County", "Harris County", "Harris~
$ X2000.01.31 <dbl> 211062.5, 153538.6, 120475.9, 122267.8, 112019.8, 185062.5, ~
$ X2000.02.29 <dbl> 211043.2, 154429.5, 120460.6, 122163.2, 112044.7, 185410.1, ~
$ X2000.03.31 <dbl> 211640.2, 154971.3, 120322.1, 121980.0, 112117.3, 185695.9, ~
$ X2000.04.30 <dbl> 212624.7, 156401.2, 120364.3, 121931.7, 112233.5, 186254.4, ~
$ X2000.05.31 <dbl> 212987.0, 157621.3, 120397.6, 121927.0, 112329.4, 186791.0, ~
$ X2000.06.30 <dbl> 212949.3, 159049.9, 120593.0, 122133.5, 112363.8, 187215.1, ~
$ X2000.07.31 <dbl> 212249.4, 160041.5, 120661.8, 122297.2, 112372.7, 187049.3, ~
$ X2000.08.31 <dbl> 212193.0, 161256.2, 120928.1, 122685.0, 112320.5, 186780.8, ~
$ X2000.09.30 <dbl> 211819.9, 162688.0, 121335.6, 123245.1, 111854.6, 186392.8, ~
$ X2000.10.31 <dbl> 211846.9, 164561.5, 121738.5, 123754.3, 111083.1, 186598.0, ~
$ X2000.11.30 <dbl> 212071.4, 166478.0, 122174.9, 124348.1, 110236.8, 186917.4, ~
```

You get a quick, compact summary of your data The function takes one argument: the name of a data frame. It then tells us how many rows it has, how many variables there are, what these variables are called, and what kind of data are associated with each variable. This function is useful when we're working with a data set containing many variables.

If you like viewing your data through an Excel style worksheet, type in `View(data)`, and data should pop up in the top left window of your R Studio interface. Scroll up and down, left and right. Take some time to understand the format of the data set. What do the columns represent? Rows?

tidyverse can read in more than just .csv files. It has a suite of `read_` functions that are a part of the subpackage **readr**. The goal of **readr** is to provide a fast and friendly way to read rectangular data (like csv, tsv, and fwf). It is designed to flexibly parse many types of data found in the wild, while still cleanly failing when data unexpectedly changes. To learn more about these functions, see **readr**'s dedicated site.

By learning how to read in data, you've earned another **tidyverse** badge! Hooray!

The data set that you brought in above using `read_csv()` saved the file in a special R object known as a *tibble*. Although the **tidyverse** works with all data objects, its fundamental object type is the tibble. Tibbles are essentially a special variant of data frames that have desirable properties for printing and joining.

Data Wrangling

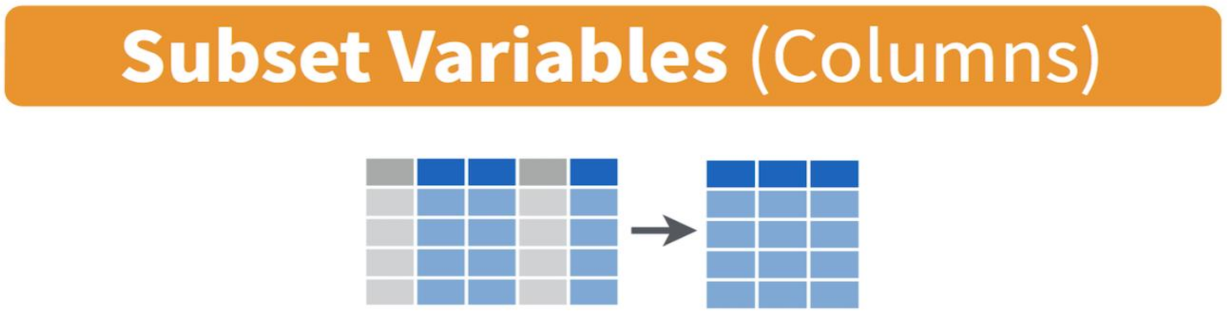
It is rare that the data you download are in exactly the right form for analysis. For example, rather than all cities in the tibble `data`, you might want to analyze just New York city. Or you might want to discard certain variables from the dataset to reduce clutter.

In this lab, we won't have time to go through all of the methods and functions in R that are associated with the data wrangling process. We will cover more in Lab 3 and later labs. Many methods you will have to learn on your own given the specific tasks you will need to accomplish. In the rest of this guide, we'll go through some of the basic data wrangling techniques using the functions found in the package **dplyr**, which was automatically installed and loaded when you brought in the **tidyverse** package.

Selecting variables

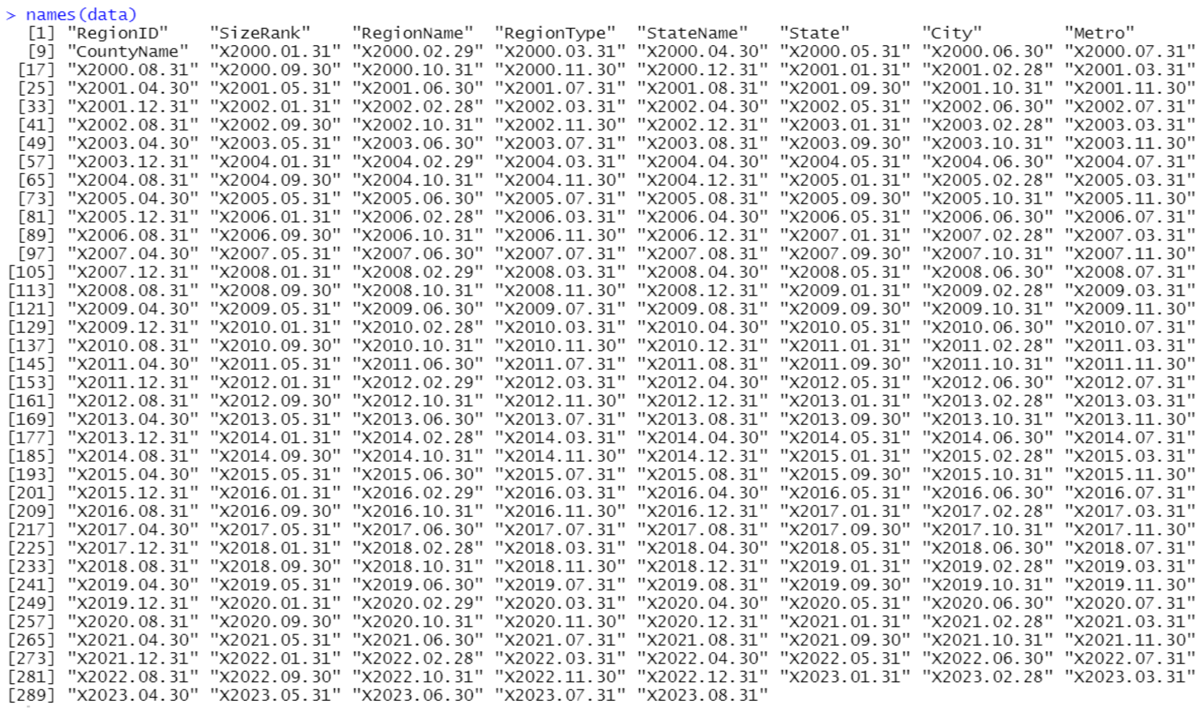
In practice, most of the data files you will download will contain variables you won't need. It is easier to work with a smaller dataset as it reduces clutter and clears up memory space, which is important if you are executing complex tasks on a large number of observations. Use the command `select()` to keep variables by name.

select(data, RegionID)



To see the names of variables in the dataset `data`, use the `names()` command.

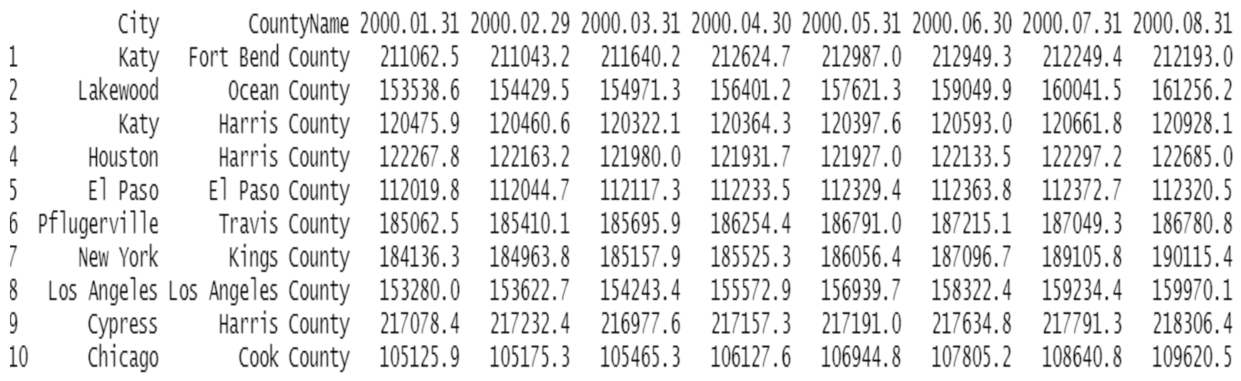
```
names(data)
```



```
> names(data)
  [1] "RegionID"    "SizeRank"    "RegionName"  "RegionType"  "StateName"   "State"       "City"        "Metro"      
  [9] "CountyName"  "X2000.01.31" "X2000.02.29" "X2000.03.31" "X2000.04.30" "X2000.05.31" "X2000.06.30" "X2000.07.31"
 [17] "X2000.08.31" "X2000.09.30" "X2000.10.31" "X2000.11.30" "X2000.12.31" "X2001.01.31" "X2001.02.28" "X2001.03.31"
 [25] "X2001.04.30" "X2001.05.31" "X2001.06.30" "X2001.07.31" "X2001.08.31" "X2001.09.30" "X2001.10.31" "X2001.11.30"
 [33] "X2001.12.31" "X2002.01.31" "X2002.02.28" "X2002.03.31" "X2002.04.30" "X2002.05.31" "X2002.06.30" "X2002.07.31"
 [41] "X2002.08.31" "X2002.09.30" "X2002.10.31" "X2002.11.30" "X2002.12.31" "X2003.01.31" "X2003.02.28" "X2003.03.31"
 [49] "X2003.04.30" "X2003.05.31" "X2003.06.30" "X2003.07.31" "X2003.08.31" "X2003.09.30" "X2003.10.31" "X2003.11.30"
 [57] "X2003.12.31" "X2004.01.31" "X2004.02.29" "X2004.03.31" "X2004.04.30" "X2004.05.31" "X2004.06.30" "X2004.07.31"
 [65] "X2004.08.31" "X2004.09.30" "X2004.10.31" "X2004.11.30" "X2004.12.31" "X2005.01.31" "X2005.02.28" "X2005.03.31"
 [73] "X2005.04.30" "X2005.05.31" "X2005.06.30" "X2005.07.31" "X2005.08.31" "X2005.09.30" "X2005.10.31" "X2005.11.30"
 [81] "X2005.12.31" "X2006.01.31" "X2006.02.28" "X2006.03.31" "X2006.04.30" "X2006.05.31" "X2006.06.30" "X2006.07.31"
 [89] "X2006.08.31" "X2006.09.30" "X2006.10.31" "X2006.11.30" "X2006.12.31" "X2007.01.31" "X2007.02.28" "X2007.03.31"
 [97] "X2007.04.30" "X2007.05.31" "X2007.06.30" "X2007.07.31" "X2007.08.31" "X2007.09.30" "X2007.10.31" "X2007.11.30"
[105] "X2007.12.31" "X2008.01.31" "X2008.02.29" "X2008.03.31" "X2008.04.30" "X2008.05.31" "X2008.06.30" "X2008.07.31"
[113] "X2008.08.31" "X2008.09.30" "X2008.10.31" "X2008.11.30" "X2008.12.31" "X2009.01.31" "X2009.02.28" "X2009.03.31"
[121] "X2009.04.30" "X2009.05.31" "X2009.06.30" "X2009.07.31" "X2009.08.31" "X2009.09.30" "X2009.10.31" "X2009.11.30"
[129] "X2009.12.31" "X2010.01.31" "X2010.02.28" "X2010.03.31" "X2010.04.30" "X2010.05.31" "X2010.06.30" "X2010.07.31"
[137] "X2010.08.31" "X2010.09.30" "X2010.10.31" "X2010.11.30" "X2010.12.31" "X2011.01.31" "X2011.02.28" "X2011.03.31"
[145] "X2011.04.30" "X2011.05.31" "X2011.06.30" "X2011.07.31" "X2011.08.31" "X2011.09.30" "X2011.10.31" "X2011.11.30"
[153] "X2011.12.31" "X2012.01.31" "X2012.02.29" "X2012.03.31" "X2012.04.30" "X2012.05.31" "X2012.06.30" "X2012.07.31"
[161] "X2012.08.31" "X2012.09.30" "X2012.10.31" "X2012.11.30" "X2012.12.31" "X2013.01.31" "X2013.02.28" "X2013.03.31"
[169] "X2013.04.30" "X2013.05.31" "X2013.06.30" "X2013.07.31" "X2013.08.31" "X2013.09.30" "X2013.10.31" "X2013.11.30"
[177] "X2013.12.31" "X2014.01.31" "X2014.02.28" "X2014.03.31" "X2014.04.30" "X2014.05.31" "X2014.06.30" "X2014.07.31"
[185] "X2014.08.31" "X2014.09.30" "X2014.10.31" "X2014.11.30" "X2014.12.31" "X2015.01.31" "X2015.02.28" "X2015.03.31"
[193] "X2015.04.30" "X2015.05.31" "X2015.06.30" "X2015.07.31" "X2015.08.31" "X2015.09.30" "X2015.10.31" "X2015.11.30"
[201] "X2015.12.31" "X2016.01.31" "X2016.02.29" "X2016.03.31" "X2016.04.30" "X2016.05.31" "X2016.06.30" "X2016.07.31"
[209] "X2016.08.31" "X2016.09.30" "X2016.10.31" "X2016.11.30" "X2016.12.31" "X2017.01.31" "X2017.02.28" "X2017.03.31"
[217] "X2017.04.30" "X2017.05.31" "X2017.06.30" "X2017.07.31" "X2017.08.31" "X2017.09.30" "X2017.10.31" "X2017.11.30"
[225] "X2017.12.31" "X2018.01.31" "X2018.02.28" "X2018.03.31" "X2018.04.30" "X2018.05.31" "X2018.06.30" "X2018.07.31"
[233] "X2018.08.31" "X2018.09.30" "X2018.10.31" "X2018.11.30" "X2018.12.31" "X2019.01.31" "X2019.02.28" "X2019.03.31"
[241] "X2019.04.30" "X2019.05.31" "X2019.06.30" "X2019.07.31" "X2019.08.31" "X2019.09.30" "X2019.10.31" "X2019.11.30"
[249] "X2019.12.31" "X2020.01.31" "X2020.02.29" "X2020.03.31" "X2020.04.30" "X2020.05.31" "X2020.06.30" "X2020.07.31"
[257] "X2020.08.31" "X2020.09.30" "X2020.10.31" "X2020.11.30" "X2020.12.31" "X2021.01.31" "X2021.02.28" "X2021.03.31"
[265] "X2021.04.30" "X2021.05.31" "X2021.06.30" "X2021.07.31" "X2021.08.31" "X2021.09.30" "X2021.10.31" "X2021.11.30"
[273] "X2021.12.31" "X2022.01.31" "X2022.02.28" "X2022.03.31" "X2022.04.30" "X2022.05.31" "X2022.06.30" "X2022.07.31"
[281] "X2022.08.31" "X2022.09.30" "X2022.10.31" "X2022.11.30" "X2022.12.31" "X2023.01.31" "X2023.02.28" "X2023.03.31"
[289] "X2023.04.30" "X2023.05.31" "X2023.06.30" "X2023.07.31" "X2023.08.31"
```

It looks like the dataset got imported correctly. However, there are clear issues that require correction during data preparation. For example, the dates for the value taken are shown as an individual variable in wider format. This needs to be tidy up into the long format as well there is a character X added to these dates which needs to be removed from every column.

Exercise 1: [20pts] Let's keep City, CountyName, and all the dates remove the prefix X to bring the dates back to their standard format and display the first ten rows and columns of the dataset as shown in below Figure.



```
          City         CountyName 2000.01.31 2000.02.29 2000.03.31 2000.04.30 2000.05.31 2000.06.30 2000.07.31 2000.08.31
1         Katy   Fort Bend County   211062.5   211043.2   211640.2   212624.7   212987.0   212949.3   212249.4   212193.0
2     Lakewood       Ocean County   153538.6   154429.5   154971.3   156401.2   157621.3   159049.9   160041.5   161256.2
3         Katy      Harris County   120475.9   120460.6   120322.1   120364.3   120397.6   120593.0   120661.8   120928.1
4      Houston      Harris County   122267.8   122163.2   121980.0   121931.7   121927.0   122133.5   122297.2   122685.0
5      El Paso     El Paso County   112019.8   112044.7   112117.3   112233.5   112329.4   112363.8   112372.7   112320.5
6  Pflugerville     Travis County   185062.5   185410.1   185695.9   186254.4   186791.0   187215.1   187049.3   186780.8
7     New York       Kings County   184136.3   184963.8   185157.9   185525.3   186056.4   187096.7   189105.8   190115.4
8  Los Angeles Los Angeles County   153280.0   153622.7   154243.4   155572.9   156939.7   158322.4   159234.4   159970.1
9      Cypress      Harris County   217078.4   217232.4   216977.6   217157.3   217191.0   217634.8   217791.3   218306.4
10     Chicago        Cook County   105125.9   105175.3   105465.3   106127.6   106944.8   107805.2   108640.8   109620.5
```

Exercise 2: **[20pts]** Is this data tidy or untidy? Explain your reasoning.

Pivoting

The **tidyverse** strongly encourages the use of tidy data, or at least almost tidy data. If your data is (almost) tidy, you can be reasonably sure that you can plot and analyze the data without additional wrangling. If your data is not (almost) tidy because it is too wide or too long (see below), what is required is a joyful round of pivoting. There are two directions of pivoting: making data longer, and making data wider.

Making too wide data longer with *`pivot_longer`*

Consider the example of messy data again:

```
exam_results_visual <- tribble(
  ~exam,        ~"Rozz",    ~"Andrew",    ~"Siouxsie",
  "midterm",    "1.3",      "2.0",        "1.7",
  "final"   ,   "2.3",      "1.7",        "1.0"
)
exam_results_visual
```

```
## # A tibble: 2 × 4
##   exam    Rozz  Andrew Siouxsie
##   <chr>   <chr> <chr>  <chr>
## 1 midterm 1.3   2.0    1.7
## 2 final   2.3   1.7    1.0
```

This data is “too wide”. We can make it longer with the function `pivot_longer` from the `tidyr` package. Check out the example below before we plunge into a description of `pivot_longer`.

```
exam_results_visual %>%
  pivot_longer(
    # pivot every column except the first
    # (a negative number here means "exclude column with that index number")
    cols = - 1,
    # name of new column which contains the
    # names of the columns to be "gathered"
    names_to = "student",
    # name of new column which contains the values
    # of the cells which now form a new column
    values_to = "grade"
  ) %>%
  # optional reordering of columns (to make
  # the output exactly like `exam_results_tidy`)
  select(student, exam, grade)
```

```
## # A tibble: 6 × 3
##   student  exam    grade
##   <chr>    <chr>   <chr>
## 1 Rozz     midterm 1.3
## 2 Andrew   midterm 2.0
## 3 Siouxsie midterm 1.7
## 4 Rozz     final   2.3
## 5 Andrew   final   1.7
## 6 Siouxsie final   1.0
```

What `pivot_longer` does, in general, is take a bunch of columns and gather the values of all cells in these columns into a single, new column, the so-called value column, i.e., the column with the values of the cells to be gathered. If `pivot_longer` stopped here, we would lose information about which cell values belonged to which original column. Therefore, `pivot_longer` also creates a second new column, the so-called name column, i.e., the column with the names of the original columns that we gathered together. Consequently, in order to do its job, `pivot_longer` minimally needs three pieces of information:

1. which columns to spin around (function argument `cols`)
2. the name of the to-be-created new value column (function argument `values_to`)
3. the name of the to-be-created new name column (function argument `names_to`)

Making too long data wider with *`pivot_wider`*

Consider the following example of data which is untidy because it is too long:

```
mixed_results_too_long <-
  tibble(student = rep(c('Rozz', 'Andrew', 'Siouxsie'), times = 2),
         what    = rep(c('grade', 'participation'), each = 3),
         howmuch = c(2.7, 2.0, 1.0, 75, 93, 33))
mixed_results_too_long
```

```
## # A tibble: 6 × 3
##   student  what          howmuch
##   <chr>    <chr>           <dbl>
## 1 Rozz     grade             2.7
## 2 Andrew   grade             2
## 3 Siouxsie grade             1
## 4 Rozz     participation    75
## 5 Andrew   participation    93
## 6 Siouxsie participation    33
```

This data is untidy because it lumps two types of different measurements (a course grade, and the percentage of participation) in a single column. These are different variables, and so should be represented in different columns.

To fix a data representation that is too long, we can make it wider with the help of the `pivot_wider` function from the `tidyr` package. We look at an example before looking at the general behavior of the `pivot_wider` function.

```
mixed_results_too_long %>%
  pivot_wider(
    # column containing the names of the new columns
    names_from = what,
    # column containing the values of the new columns
    values_from = howmuch
  )
```

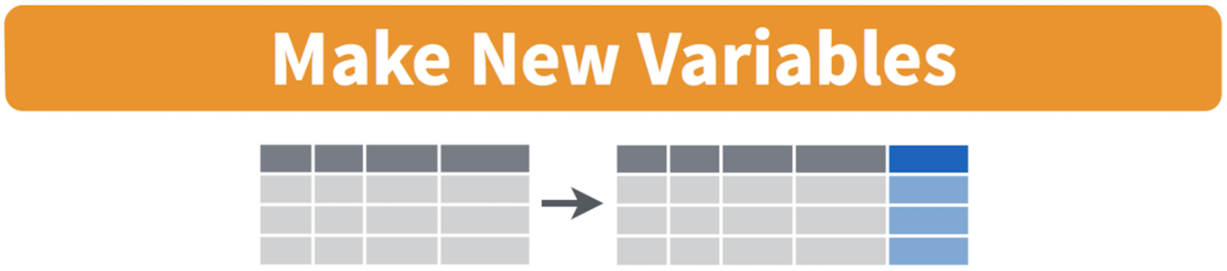
```
## # A tibble: 3 × 3
##   student  grade participation
##   <chr>    <dbl>         <dbl>
## 1 Rozz       2.7            75
## 2 Andrew     2              93
## 3 Siouxsie   1              33
```

In general, `pivot_wider` picks out two columns, one column of values to distribute into new to-be-created columns, and one vector of names or groups which contains the information about the, well, names of the to-be-created new columns. There are more refined options for `pivot_wider`, some of which we will encounter in the context of concrete cases of application.

Exercise 3: **[20pts]** In exercise 2, if you argue that the dataset `data` is not tidy then make it to become a tidy data.

Creating new variables

The `mutate()` function allows you to create new variables within your dataset. This is important when you need to transform variables in some way - for example, calculating a ratio or adding two variables together. Visually, you are doing this

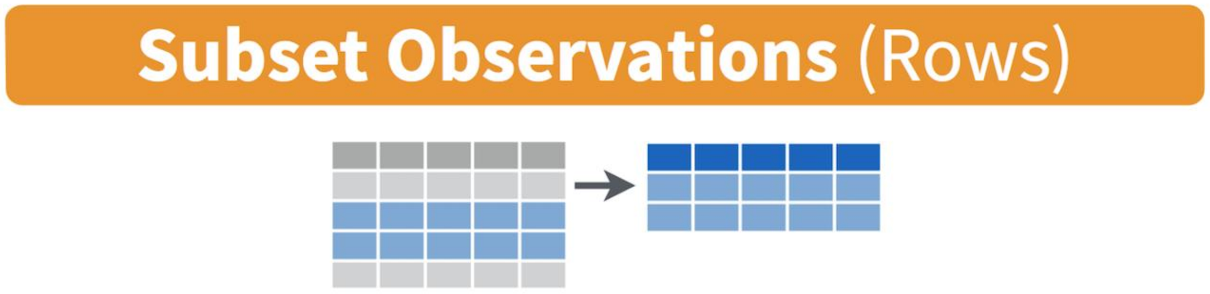


You can use the `mutate()` command to generate as many new variables as you would like.

Exercise 4: **[20pts]** Let's construct two new variables in the dataset `data:` namely monthly and price where monthly represents the dates and price is the values/prices associated with the dates.

Filtering

Filtering means selecting rows/observations based on their values. To filter in R, use the command `filter()`. Visually, filtering rows looks like this.



The first argument in the parentheses of this command is the name of the data frame. The second and any subsequent arguments (separated by commas) are the expressions that filter the data frame.

<u>Exercise 5</u>: **[10pts]** Let's filter houses in New York city has prices lower than 300.000 USD after August 31st, 2010.

<u>Exercise 6</u>: [10pts] Let's save the dataset that you filter in Exercise 5 as a csv file in the folder of your current working directory.

<u>What to submit</u>:

Your submission should include the following:

1. Lab report answers to the sĩx exercises above and source code.
2. Please create a folder called "yourname_studentID_lab2" that includes all the required files and generate a zip file called "yourname_studentID_lab2.zip".
3. Please submit your work (.zip) to Blackboard.