In the last few missions, we've used vectors & matrices to help calculate your grades and make university recommendations. Now that you've started university, you'll need to decide your major. In this mission, we'll use a **dataframe** to decide on a major.

Previously, you learned that vectors & matrices only contain *one* data type. We've only dealt with *numeric* data types. In most scenarios, your data will be a mixture of multiple data types: character, numeric, logical and others.

If we wanted to store *multiple* data types, a vector or matrix cannot hold our data. Instead, we can use a **dataframe** to store our values. A dataframe is a two-dimensional data structure, similar to a matrix, that can hold *multiple* data types. A dataframe is the *de facto* structure we'll use when loading tabular data & performing statistical analysis.

To help decide on a major, we obtained data on recent college graduates from [FiveThirtyEight's GitHub account](https://github.com/fivethirtyeight/data/tree/master/college-majors). Here's a description of the columns:

* Rank: Rank by median earnings
* Major\_code: Major code, FO1DP in ACS PUMS
* Major: Major description
* Total: Total number of people with major
* Men: Male graduates
* Women: Female graduates
* Major\_category: Category of major
* ShareWomen: Women as share of total
* Sample\_size: Sample size (unweighted) of full-time, year-round ONLY (used for earnings)
* Employed: Number employed
* Full\_time: Employed 35 hours or more
* Part\_time: Employed less than 35 hours
* Full\_time\_year\_round: Employed at least 50 weeks and at least 35 hours
* Unemployed: Number unemployed
* Unemployment\_rate: Unemployed / (Unemployed + Employed)
* Median: Median earnings of full-time, year-round workers
* P25th: 25th percentile of earnigns
* P75th: 75th percentile of earnings
* College\_jobs: Number with job requiring a college degree
* Non\_college\_jobs: Number with job not requiring a college degree
* Low\_wage\_jobs: Number in low-wage service jobs

To decide on a major, you've decided that you have two main criteria:

**You'd like to have a decent salary where the median salary of recent graduates is above 50000.**

**You'd like to avoid male-dominated majors, where the share of women is less than 40%.**

To find the majors that'll satisfy your criteria, we'll perform the following steps:

To display the majors that satisfy our criteria, we'll need to access our dataset. To access our dataset, let's introduce our data in a file format called .csv.

In order for us to use data to make a major selection, we'll need to access data containing every major. Our current dataset, is stored in a file format called .csv. The R interpreter can load most types of data files (.xls, .txt ... etc). .csv files are common, compared to .xls files, because most data interfaces accept .csv files. .xls files can hold more complexity, but are limited in usage scope.

.csv stands for comma separated values. Each value, is separated by a distinct comma. The first row of the file, indicates all the column headers. The commas that separate each value are called *delimiters*. Also, notice how there's no comma separating the lines. To indicate a new row, the .csv file will actually push the new row to the next line in the file:

As you'll see, each cell in our spreadsheet, is defined by the commas in the .csv file. Now, to load this .csv file into an R dataframe, you'll use the read.csv() function. The read.csv() function will read in a csv file and store this file in a dataframe. The boundaries for each value is defined by the commas. Here's an example of us reading in a dummy "filename.csv" file:



df <- read.csv("filename.csv")

We're reading a file called "filename.csv" and storing this in a dataframe called df. Let's load in our recent grads data!

instructions

* Use the read.csv() function to load the csv file "recent-grads.csv".
* Store this in a dataframe called df.
* Print the dataframe using the print() function.

In the previous screen, we printed all the majors using the print() function. We're lucky that we only have 173 different majors, otherwise, scrolling through a million rows may be too cumbersome. Although we're lucky that this dataset is only 173 rows, there'll be many situations in the future, where you'll deal with millions of rows.

Since there are 173 majors, we might want a quick preview just to see how all the majors are formatted. To get a quick preview, we can use the head() function. In a previous mission, we introduced the head() function to show the top values. We can also apply head() to our dataframe to preview our dataset. When we use head() on a dataset, by default, the interpreter will return the header and the first six rows. Light blue indicates the header rows returned & darker blue indicates the row values returned:

We could also use tail() to view the *last* six rows of our dataframe:

Let's use head() & tail() function to preview our dataset.

instructions

* Apply the head() function on df to view the first six rows of our dataframe.
* Apply the tail() function on df to view the last six rows of our dataframe.
* When viewing the dataset, try to understand what the numbers mean & how the rows relate to the columns.

Now that we've seen a preview of the majors & their relevant columns, we'll need to dig one level deeper to find a major that satisfies our requirements. Since we're looking to filter & possibly manipulate values in our dataframe, we'll need to understand the types of data within our dataframe.

Accessing the data types in our dataframe requires us to examine the internal structure. Revealing the internal structure will show us the data types, the number of observations, variables/columns & even a preview of the values in each column.

To access this internal structure, we'll use the [str()](https://stat.ethz.ch/R-manual/R-devel/library/utils/html/ls_str.html) function. The str() function can extract the internal structure of dataframes, matrices, vectors and any R object. Understanding the data types for each column will be important when we want to transform or reformat specific columns:



str(df)

Let's use the str() function to examine the internal structure of our dataframe.

instructions

* Use str() on our current dataframe.
* Note the different data types for each column.

In order to know how to manipulate our dataframe to answer our question, let's understand the data types that make up our df. Notice the variety of data types: int, Factor, num. In the previous missions, we've learned that num is the numeric data type. However, we notice that there's a different data type for for numerical values called int.

Integer data type is a subset of numeric data types. Numeric data types are either whole numbers (5) or a decimal numbers (5.3). Integers are the whole numbers (5) in our dataset. Keep in mind, you can have whole numbers that are numeric but *are not* integers. You *cannot* have decimal numbers as integers:

You can use the is.numeric() or is.integer() functions to check whether a value is of integer or numeric data type:



petroleum\_engineering\_unemp\_rate = 0.0184

​

is.integer(petroleum\_engineering\_unemp\_rate)

This would return:

FALSE

Let's check the data type of a few values!

instructions

* For the variable petro\_eng\_med\_salary , check if the variable an integer. Store this in pet\_integer
* For the variable finance\_med\_salary , check if the variable an integer. Store this in fin\_integer

When examining the structure of our dataframe, the other data type is Factor. Factors are used to represent categorical data like Major\_category. Since these are categorical, they have a limited range of values. The number of unique categories are called **levels**. Whenever you're dealing with factor variables, each value will belong to one of the levels.

Let's look at an example using a miniature version of the Major\_category column within our dataset:

Let's say we wanted to convert these character values into a factor variable. To convert, we'll use the factor() function on our values.



Major\_category <- c("Engineering","Business","Engineering","Business","Biology","Literature","Biology")

​

maj\_factor <- factor(Major\_category)

To check our levels, we'll use the levels() function on our factor variables to access the levels:



levels(maj\_factor)

And this should display:

[1] "Biology" "Business" "Engineering" "Literature"

When we convert a vector, or any object into a factor variable, the R interpreter will automatically use these values to create **levels**. Notice, that the levels of a factor variable, are all the unique categories. In addition, each level is assigned an integer value. Integer values are useful for when we want to *order* our categorical values:

For the context of this mission, we won't need to order our categorical values. The most important thing to understand, is that factor variables will allow us to model, visualize & analyze categorical variables.

To better understand this concept, let's create factor variables!

instructions

* We've created a vector of majors. Turn this vector into a factor called factor\_majors.
* Use the levels() function on factor\_majors, see the levels within the vector.
* After examining the levels, store all the levels in major\_levels.

Now that we understand all the data types, we're now equipped to start pulling out data regarding specific majors. We can now start selecting rows & columns in our dataframe. Selecting rows & columns is similar to selecting rows & columns from a matrix. In this screen, we'll focus on selecting rows.

To select a row, we'll use the format: df[row,column].

In addition, we can select a row by using the *name* of the index. Here are the varying ways of selecting rows:

Keep in mind, there are multiple ways to select a single row. Let's say you're curious about architechtural engineering, computer science and the top 100 majors. Let's use our current dataset to find these values!

instructions

* Select rows 1-100 and store them in rank\_1\_100.
* Select row 19 and store this in architectural\_engineering.
* Select row 21 and store this in computer\_science.

Only selecting a few majors(rows) we're curious about can give us some solid information. There are a few columns that give us relevant information, like Median & Women. There are other columns, like Non-College Job that may not be applicable to answering our question.

To make these specifications to our selection, we'll need to learn how to select columns. Selecting columns is similar to selecting rows, except now, we're leaving the rows section empty: df[row,column]. Here are the ways to select a column:

We can also select columns by column index. However, in most cases, we'll be using names to select columns. Let's select a few columns from our dataframe.

instructions

* Select the columns Major,Unemployment\_rate,Median,Men,Women from df and store this in select\_df.

Now that we've learned how to select both rows & columns, we can now get more *specific* about the types of questions we ask our dataset. Rather than asking "What are the stats for petroleum engineering?", we can ask "What is the median salary of petroleum engineering?" To find specific values, we'll add both row & column index in one expression: df[row,column].

To see this in action, let's return to our sample dataset:

Let's pull the rank of the engineering major. To pull the rank, we'd write the following code:



df["Engineering","Rank"]

Here, we can see the selection in diagram form:

As a result, this would return 1. Keep in mind, that the major names here are added to the *index* of the dataframe. This is why we can select a row with Engineering.

Let's pull a few more values!

instructions

* We've gone ahead and used the rownames() accessor function and stored the major names as the row names.
* Select the "Median" of "MECHANICAL ENGINEERING". Store this in mech\_eng\_salary. Keep in mind Median is the median salary.
* Select the "Median" of "COMPUTER SCIENCE". Store this in comp\_sci\_salary.
* Select the "Median" of "FINANCE". Store this in finance\_salary.

Now that we can answer more granular questions like "what is the rank of the engineering major?", we can take our questions one step further by introducing comparison operators. Comparison operators, is what will allow us to filter our dataframe by the conditions we want: Median salary above 50000 & share of women above 40%. If you'd like to review the different comparison operators, feel free to turn back to the vectors mission.

To add a comparison operator to our selection, we'll use the same indexing format, df[row,column], except we'll add a comparison operator to the index. Let's return to our sample dataset:

Let's say we wanted to return all the majors with more than 400 women. For the first step, first select the column we're adding a condition to:



df$Women

Now, let's add our comparison operator:



df$Women > 400

As a result, the R interpreter will create a vector of logical data types:



TRUE TRUE FALSE FALSE

Now, let's take this vector, and index this back into our dataframe. The tricky part here, is that even though we're adding a comparison operator to our columns, we need to place this vector into the *rows* section of df[rows,columns]. In our example, df$Women > 400 translates in english to *return every major that has more than 400 women*:

Let's add our conditions to our dataframe!

instructions

* Return all rows that have a median salary above 50000. Store this in above\_50. Use Median column for this.
* Return all rows that are engineering majors. Store this in engineering. Use Major\_category column for this.
* Return all rows that have share of women greater than 0.40. Store in great\_40.Use ShareWomen column for this.

In the previous screen, we've filtered our dataframe by salary & share of women. However, we've stored our results in two different objects. We want to find the majors that satisfy *both* our conditions. To find the major that satisfies both our conditions, we'll use **logical operators**.

Logical operators don't behave like comparison operators. While comparison operators look to compare the value of two objects, a logical operator bridges two comparisons into one expression using AND or OR. Let's dive into the concepts of AND, OR to understand their role.

**&**: This represents AND. This operator compares two conditions and return the results that satisfy *both* conditions. Let's return to our previous example, where we wanted to return majors that have more than 400 women. Let's add a condition, where we want to see the majors that have more than 400 women *and* more than 450 men.

As you can see, the blue denotes all the values that satisfy our first condition. The red denotes all the values that satisfy our second condition. The purple, denotes all the values that satisfy *both* conditions. As a result, this condition would return the following row:

**|**: This represents OR. This operator compares two conditions and return the results that satisfy *either* conditions. Let's re-create our previous example using the |:

And this would display:

Let's use our logical operators to filter our dataframe!

instructions

* Filter the dataframe by majors that have salary greater than 50000(Median) **and** share of women(ShareWomen) greater than 0.4. Store this in majors.

We'ved wittled our list of majors down to three majors. Now that we have three majors, since you'd like to avoid unemployment, let's rank these majors by Unemployment\_rate.

To rank our majors, we'll be using the order() function. The order function will automatically order any vector you feed it. Let's return to our sample dataset and order our dataframe, by Rank:



order(df$Rank)

However, we're only ordering by one column. To re-organize our dataframe with this order, we'll plug this order into rows in df[rows,columns:



df[order(df$Rank), ]

The resulting dataframe will display:

Let's order our resulting majors dataframe by Unemployment\_rate!

instructions

* Order the majors dataframe by Unemployment\_rate. Store this in major\_choice.

Congratulations! You've used R to select a major. So far, we've been using pre-made functions to perform actions for us. In the next lesson, you'll be nearing graduation from college. As you ramp up the job search, you'll be using a new data structure to organize your search: **lists**.