As a high school senior, you've calculated your grades and you received stellar marks. The next step on your journey, is to decide which college to attend. You've been accepted into the top universities in the world but need help choosing the university.

When choosing your schools, you've decided that you want to maximize the *quality* of your education while minimizing the price. In the previous mission, you learned how to use a vector to answer your questions. In this mission, we'll add another dimension by using a *matrix* to answer our question.

We'll be using the [Times Higher Education World University Ranking dataset](https://www.kaggle.com/mylesoneill/world-university-rankings) to find a university recommendation. We've removed many columns for the purpose of this mission. Each row corresponds to a specific university. Here's a description of each column in the dataset:

* University: The name of the university
* world\_rank: The world rank for the university
* quality\_of\_education: Rank for quality of education
* influence: Rank for amount of influence university has on big issues
* broad\_impact: Rank for impact university has on the world
* patents: Rank for number of patents created

Let's take a vector of data on Harvard University's world ranking in different categories:



harvard <- c(1,1,1,1,3)

names(harvard) <- c("world\_rank","quality\_of\_education","influence","broad\_impact","patents")

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | **world\_rank** | **quality\_of\_education** | **influence** | **broad\_impact** | **patents** |
| Harvard | 1 | 1 | 1 | 1 | 3 |

Let's say we wanted to compare harvard to stanford:



stanford <- c(2,9,3,4,10)

names(stanford) <- c("world\_rank","quality\_of\_education","influence","broad\_impact","patents")

While using a one-dimensional vector to store our data is useful, we're limited in a few ways:

* We need to create a new vector every time we want to add a university to our data.
* Comparing the values between the two vectors, isn't intuitive. You need to visually match the categories & rankings together.
* If we were analyzing hundreds of universities, writing out hundreds of lines of vectors might not be the most efficient way of performing an analysis.

Rather than create a vector for each school, we could use a **matrix** to hold all our university data in one place. A matrix is a collection of data values arranged in a two-dimensional, rectangular layout. A matrix is also a two-dimensional vector.

Let's compare harvard and stanford, but instead, in matrix form:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | **world\_rank** | **quality\_of\_education** | **influence** | **broad\_impact** | **patents** |
| Harvard | 1 | 1 | 1 | 1 | 3 |
| Stanford | 2 | 9 | 3 | 4 | 10 |

In matrix form, it's much easier to see which school has a higher education rank. You can also access all your data in *one* location, so you don't need to store 20 different vectors. In this mission, we'll be figuring out a university recommendation for you, by learning how to manipulate matrices.

Here's what the data looks like in this mission:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | **world\_rank** | **quality\_of\_education** | **influence** | **broad\_impact** | **patents** |
| Harvard | 1 | 1 | 1 | 1 | 3 |
| Stanford | 2 | 9 | 3 | 4 | 10 |
| MIT | 3 | 3 | 2 | 2 | 1 |
| Cambridge | 4 | 2 | 6 | 13 | 48 |
| Oxford | 5 | 7 | 12 | 9 | 15 |
| Columbia | 6 | 13 | 13 | 12 | 4 |

In the previous screen, you saw how a matrix might be a more effective way of holding more data points. In our previous scenario, we had two vectors: harvard and stanford. To realize the benefits of a matrix, let's combine these two vectors into a matrix.

To transform multiple vectors into one matrix, the first thing we'll do, is combine all our disparate vectors into one *vector*. When we introduce the matrix() function, we'll dive deeper into *why* you need to combine all disparate vectors into one vector.

As review, in the previous mission, you helped your friend Johnny calculate his scores by appending 88 to the end of the tests vector:



tests <- c(76,89,78)

​

tests <- c(tests, 88)

Combining multiple vectors into one vector, follows a similar process to appending data to a vector. Instead of appending one data point, we're appending multiple data points. Let's see this in action by combining our harvard vector and stanford vector into one vector.

Let's take our two vectors:



harvard <- c(1,1,1,1,3)

stanford <- c(2,9,3,4,10)

And then combine these into a single vector called harv\_stan:



harv\_stan <- c(harvard, stanford)

If you'd like to combine more than two vectors, you can perform the same steps. Include all the vectors you want to combine within the c() function.

We've combined two of the university vectors. Let's combine the rest!

instructions

* Combine the following university vectors into one vector called uni\_vector:

harvard <- c(1,1,1,1,3)

stanford <- c(2,9,3,4,10)

MIT <- c(3,3,2,2,1)

cambridge <- c(4,2,6,13,48)

oxford <- c(5,7,12,9,15)

columbia <- c(6,13,13,12,4)

In the previous section, you learned how to combine multiple vectors into *one* vector. Now, our data is in a format, where we can transform it into a **matrix**.

A matrix is a collection of values of the *same* data type arranged in a two-dimensional rectangular shape. To create a matrix, you'll use the matrix() function. Within the matrix() function, there are a few arguments you'll enter. An argument is an input a function needs to give you an output. Here is the matrix function & its arguments:

matrix(data = NA ,nrow = 1,ncol = 1,byrow = FALSE)

* **data:** This is the data that will be transformed into a matrix. This data must be in *vector* form. This is why you combined vectors in the earlier screen.
* **nrow:** This is the number of *rows* you want your matrix to hold.
* **ncol:** This is the number of *columns* you want your matrix to hold.
* **byrow:** This is a logical value of either TRUE or FALSE. If TRUE, the matrix will be filled by *rows*. If false, by *columns*. We'll dive deeper into how this argument works, later in this mission.

Let's fill in these arguments by transforming our harvard and stanford vectors into a matrix. In the previous screen, we combined our two vectors into one vector:

Since we're only dealing with two universities with five different categories each, we'll need to create a matrix with 2 rows and 5 columns. Let's use the matrix() function to create an empty matrix with the specified dimensions. We're creating an empty matrix, because later on, we're going to *fill* our matrix with the data from harv\_stan:

Let's fill our matrix with the data from harv\_stan. However, there are multiple ways to *fill* our matrix. This is why we have the byrow parameter. By default, if we do not specify anything, byrow will be set to TRUE. This means, by default, the interpreter will fill the matrix *by row*:

If we wanted to fill our matrix *by column*, we'd set byrow=FALSE. As a result, the matrix will use the vector to fill in a different direction:

Let's use the matrix() function to transform our combined vector into a matrix!

instructions

* Here's your uni\_vector:

uni\_vector <- c(harvard, stanford, MIT, cambridge, oxford, columbia)

* Transform this uni\_vector into a six row by five column matrix. Name this uni\_matrix.
* Fill this matrix setting byrow to TRUE.

So far, we've only worked with vectors that contain values of a single type. From your previous missions, notice, that we've *never* created a vector or matrix, that contain *multiple* data types. This is because, a vector & matrix can only store *one* data type. When you used names() to *name* your vector, you are accessing something called an *attribute*, which *is not* considered a value in the object.

If you tried to create a vector or matrix containing multiple data types, the R interpreter *will not* return an error. Instead, the R interpreter will attempt to guess the correct data type and then convert all your data to that data type.

For example, let's say we tried to store the following vector of harvard & stanford values with both numeric and character data types:



harv\_stan\_dtype <- c("harvard",1,1,1,1,3,"stanford",2,9,3,4,10)

If we display harv\_stan\_dtype using the print() function, we see:



[1] "harvard" "1" "1" "1" "1" "3"

[7] "stanford" "2" "9" "3" "4" "10"

In the previous screen, you used a vector as an argument for a matrix. Because matrices are vectors in two-dimensions, when you create a matrix with multiple data types, the R interpreter will guess the correct data type. Then, it'll convert all the data into the guessed data type.

Let's try creating a vector of multiple data types to see this in action!

instructions

* Create a vector with two data types, using the following values:

"columbia",6,13,13,12,4

* Store this vector in columbia\_types.
* Use the class() function on this vector to check it's data type. Store this in type.
* Print type.

Now that you've created a matrix, let's look at what our current matrix look like:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | **[1,]** | **[2,]** | **[3,]** | **[4,]** | **[5,]** |
| [1,] | 1 | 1 | 1 | 1 | 3 |
| [2,] | 2 | 9 | 3 | 4 | 10 |
| [3,] | 3 | 3 | 2 | 2 | 1 |
| [4,] | 4 | 2 | 6 | 13 | 48 |
| [5,] | 5 | 7 | 12 | 9 | 15 |
| [6,] | 6 | 13 | 13 | 12 | 4 |

We've stored all our values, however, we don't know which row corresponds to stanford. We don't know which column corresponds to world\_rank. We don't know which row corresponds with harvard. In order to come up with a university recommendation, we need to add labels to our rows and columns.

*Naming* the rows and columns of our matrix is similar to naming a vector, except we need to name both our rows *and* columns. In a previous mission, you named the values in a vector, by using the accessor function names(). When naming a matrix, we'll be using an accessor function. Here are the accessor functions for rows and columns:

**rows**: rownames(matrix)

**columns**: colnames(matrix)

If we wanted to *name* a vector of rankings from harvard, this would look like:



harvard <- c(1,1,1,1,3)

​

categories <- c("world\_rank","quality\_of\_education","influence","broad\_impact","patents")

​

names(harvard) <- categories

To name a 2 x 5 harv\_stan *matrix*, follow the same steps, but use colnames() and rownames() instead of names():



harv\_stan\_matrix <- matrix(harv\_stan, ncol = 5, nrow = 2, byrow = TRUE)

​

rownames(harv\_stan\_matrix) <- c("harvard","stanford")

​

colnames(harv\_stan\_matrix) <- c("world\_rank","quality\_of\_education","influence","broad\_impact","patents")`

Let's label all the rows and columns of uni\_matrix.

instructions

* Name the columns and rows of our uni\_matrix with the following vectors:

categories <- c("world\_rank","quality\_of\_education","influence","broad\_impact","patents")

universities <- c("Harvard","Stanford","MIT","Cambridge","Oxford","Columbia")

* Store the resulting uni\_matrix in named\_uni\_matrix.

Here's what uni\_matrix looks like:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | **world\_rank** | **quality\_of\_education** | **influence** | **broad\_impact** | **patents** |
| Harvard | 1 | 1 | 1 | 1 | 3 |
| Stanford | 2 | 9 | 3 | 4 | 10 |
| MIT | 3 | 3 | 2 | 2 | 1 |
| Cambridge | 4 | 2 | 6 | 13 | 48 |
| Oxford | 5 | 7 | 12 | 9 | 15 |
| Columbia | 6 | 13 | 13 | 12 | 4 |

To find which universities have the best education, while minimizing costs, you'll notice we're missing data on the cost of the university. As a result, you did some additional research on the costs for each university:



tuition <- c(43280,45000,45016,49350,28450,55161)

As a result, we'd like to add this vector of information to the current dataset to compare costs between schools. When we want to add a new column or row to a matrix, we want to first make sure the additional vector matches the dimensions of the matrix. To do this, we'll find the length of the vector and see if it matches either the rows or columns.

When adding a new column, length of vector must match number of rows:

When adding a new row, length of vector must match number of columns:

It's good practice to check the dimensions to see if they match. If they do not match, the R interpreter *will not* throw an error. Instead the *recycling rule* will come into play. In many cases, the recycled values aren't useful in our analysis.

To find the dimensions of the matrix, you can use the dim() function.

Like this:



dim(harv\_stan)

The dimensions of harv\_stan matrix is:



[1] 2 5

Keep in mind, when you call dim(harv\_stan), this returns a *vector* of the *rows* by *columns*, in this order. In our example, 2 5 says we have two rows and five columns.

To find the length of a vector, you can use the length() function. Let's use this function on two tuition values: harv\_stan\_tuition <- c(43280,45000)



length(harv\_stan\_tuition)

The R interpreter will return:



[1] 2

Once you have both the dimensions of the matrix and the length of the vector you want to add, depending on whether you're adding a new row or column, make sure the length the vector matches the dimensions of the matrix. To check if they match, you can write the following:



dim(harv\_stan)[1] == length(tuition)

This should return:



TRUE

We're indexing [1] into dim(harv\_stan)[1] since using dim() will return a vector of two values: number of rows & number of columns. Since we're adding a new column we want the length of the column to match the number of rows. [1] will tell us the number of rows.

Let's use the dim() and length() function to make sure the tuition vector can fit with our matrix.

instructions

* Find the dimensions of the matrix: uni\_matrix. Display using print() statement.
* Find the length of the tuition vector. Display using print() statement.
* Use a comparison to see if the number of rows equals to the length of the vector. Store in equality. Display using print() statement.

In the previous section, we learned how to find the dimensions of our matrix to see if our tuition vector fits on our matrix. Once we confirm this fits, let's attach tuition to uni\_matrix. To attach a vector to our matrix, we'll need to learn about how to add new columns & rows.

Let's first look at adding new columns:

To add the tuition column, you'll use the cbind() function using the following format: matrix <- cbind(matrix, new\_column). Using cbind() with our harv\_stan vector looks like:



harv\_stan <- cbind(harv\_stan, tuition)

Now that we've added new columns, let's look at adding new rows:

To add rows, you'll use the rbind(). The rbind() function follows the same format as cbind(), except we're adding a new row: matrix <- rbind(matrix, new\_row). Let's see this in action by adding a new row called MIT to the harv\_stan vector:



MIT <- c(3,3,2,2,1)

​

harv\_stan <- rbind(harv\_stan, MIT)

Let's try adding the full tuition vector to our original matrix!

instructions

* Add the tuition vector to uni\_matrix:

tuition <- c(43280,45000,45016,49350,28450,55161)

Now that we added labels to our matrix and added a new column, we can now pull individual values from our matrix. First, we'll learn how to select values from a matrix by individual values. Then we'll pull entire rows or columns.

Let's look at uni\_matrix again:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | **world\_rank** | **quality\_of\_education** | **influence** | **broad\_impact** | **patents** |
| Harvard | 1 | 1 | 1 | 1 | 3 |
| Stanford | 2 | 9 | 3 | 4 | 10 |
| MIT | 3 | 3 | 2 | 2 | 1 |
| Cambridge | 4 | 2 | 6 | 13 | 48 |
| Oxford | 5 | 7 | 12 | 9 | 15 |
| Columbia | 6 | 13 | 13 | 12 | 4 |

Let's say you were curious about MIT's rank in quality\_of\_education. Since you're only looking for one, specific value, you could subset a matrix *by element*. Subsetting/Indexing into a matrix is similar to indexing into a *vector*, except we'll need to index both rows and columns. Indexing into a a matrix would look like this: matrix[row,column].

When selecting an individual value from a matrix, we'll now be indexing by both row *and* column. In the following diagram, we've color-coded the row index & column index and the corresponding values they affect. The green indicates the intersection:

And this would return:

Now that we've learned how to pull specific values, let's pull a few values from our matrix!

instructions

* Here's our original matrix:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | **world\_rank** | **quality\_of\_education** | **influence** | **broad\_impact** | **patents** |
| Harvard | 1 | 1 | 1 | 1 | 3 |
| Stanford | 2 | 9 | 3 | 4 | 10 |
| MIT | 3 | 3 | 2 | 2 | 1 |
| Cambridge | 4 | 2 | 6 | 13 | 48 |
| Oxford | 5 | 7 | 12 | 9 | 15 |
| Columbia | 6 | 13 | 13 | 12 | 4 |

* Pull the following values from uni\_matrix and store them in the corresponding variable:
  + Oxford's influence rank and store in oxford\_influence
  + Stanford's broad impact rank and store in stanford\_impact
  + Cambridge's patents rank and store in cambridge\_patents
  + MIT's world rank and store in MIT\_world\_rank

In the previous section, you learned how to select a specific element from the matrix. However, there are scenarios, where it may be better to select an entire row or column.

To select an entire row or column, you'll perform the same steps as you would if you were selecting a specific element. However, you'll leave either the row or column field blank. Let's see this in action by returning the MIT row:



harv\_stan["MIT",]

And this would return:

In this scenario, we want to return all rankings for MIT. As a result, we'll be selecting an entire row and leaving the column field blank. If you wanted to select the column, you would leave the row field blank instead. Let's select the world\_rank column as an example:

This would return:

Let's get some practice returning specific rows and columns from our uni\_matrix!

instructions

* Index the world\_rank column and store this in world\_rank.
* Index the Columbia row and store this in columbia.
* Index the patents column and store this in patents. Here's the full matrix:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | **world\_rank** | **quality\_of\_education** | **influence** | **broad\_impact** | **patents** |
| Harvard | 1 | 1 | 1 | 1 | 3 |
| Stanford | 2 | 9 | 3 | 4 | 10 |
| MIT | 3 | 3 | 2 | 2 | 1 |
| Cambridge | 4 | 2 | 6 | 13 | 48 |
| Oxford | 5 | 7 | 12 | 9 | 15 |
| Columbia | 6 | 13 | 13 | 12 | 4 |

Returning to our original question, in order to extract the top schools in quality of education while minimizing costs, let's learn how to **sort** our matrix.

To sort our matrix, we'll use the sort(x, decreasing = FALSE) function. Here are the parameter definitions.

**x**: is the object you'd like to sort.

**decreasing**: will tell you whether to sort from largest to smallest or smallest to largest. The input here will be a boolean value of either TRUE or FALSE.

The first thing we'll do, is select the column we'd like to sort our matrix by. In this example, we'll use the influence column:



uni\_matrix[,"influence"]

Next, we'll apply sort() to our column. We won't change decreasing since its FALSE by default:



sort(uni\_matrix[,"influence"])

And this would return the influence column, in sorted form:

Let's sort the schools by quality\_of\_education and tuition.

instructions

* Sort the uni\_matrix by quality\_of\_education and store this in top\_edu.
* Sort the uni\_matrix by tuition and store this in low\_cost.

Now that we have our column in order, let's find a way to return the top two values in each column. To grab the first part or the last part of a vector or matrix, you can use the [head()](http://stat.ethz.ch/R-manual/R-devel/library/utils/html/head.html) or tail() function.

To use these functions, the syntax looks like this:

head(x, n)

**x**: is the vector or matrix you're using.

**n**: is the number of elements you want to display.

Let's pull the top 2 universities with the most influence from uni\_matrix. From the last exercise, we have our matrix sorted by influence:



sort(uni\_matrix[,"influence"], decreasing = FALSE)

Let's store this in another object, so the code doesn't get too messy:



sorted\_influence <- sort(uni\_matrix[,"influence"], decreasing = FALSE)

To select the top two, let's wrap sorted\_influence with the head() function, and only select the top two:



head(sorted\_influence, 2)

This would return:

Let's use the head() function to pull the top two from quality\_of\_education and cost!

instructions

* Use the head() function to pull the top two quality\_of\_education schools from top\_edu.
* Store this value in top\_two\_edu.
* Use the head() function to pull the two lowest cost schools from low\_cost.
* Store this value in two\_low\_cost.

In review, we've gone from building our own matrix to sorting, creating new columns & filtering our matrix. We've also extracted the top university's(Harvard) that match your criteria using data.

In the next mission, we'll learn how to use & manipulate dataframes, a data structure similar to matrices that can store *multiple* data types. Now that you've chosen a university, we'll use dataframes to choose a major:

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Rank** | **Major\_code** | **Major** | **Total** | **Men** | **Women** | **Major\_category** | **ShareWomen** | **Sample\_size** | **Employed** | **⋯** | **Part\_time** | **Full\_time\_year\_round** | **Unemployed** | **Unemployment\_rate** | **Median** | **P25th** | **P75th** | **College\_jobs** | **Non\_college\_jobs** | **Low\_wage\_jobs** |
| 1 | 2419 | PETROLEUM ENGINEERING | 2339 | 2057 | 282 | Engineering | 0.1205643 | 36 | 1976 | ⋯ | 270 | 1207 | 37 | 0.01838053 | 110000 | 95000 | 125000 | 1534 | 364 | 193 |
| 2 | 2416 | MINING AND MINERAL ENGINEERING | 756 | 679 | 77 | Engineering | 0.1018519 | 7 | 640 | ⋯ | 170 | 388 | 85 | 0.11724138 | 75000 | 55000 | 90000 | 350 | 257 | 50 |
| 3 | 2415 | METALLURGICAL ENGINEERING | 856 | 725 | 131 | Engineering | 0.1530374 | 3 | 648 | ⋯ | 133 | 340 | 16 | 0.02409639 | 73000 | 50000 | 105000 | 456 | 176 | 0 |