In the previous missions, we've equipped you with a variety of tools to start manipulating datasets. When manipulating datasets, you'll encounter many data types. In the previous course, we learned how to use arithmetic operators, comparison operators to make calculations using numerical values. But what if we wanted to do the same thing with strings or dates?

Note: Strings are the same as character data types.

When you want to build a model that predicts stock prices, time/date will be a huge component of your model. Are there any trends on specific days? Does time of day influence our results? If you ever want to do any text analysis such as predicting if an e-mail is spam or not, you might want to look at different key words. To analyze key words, we'll need to understand how to manipulate strings/dates.

In this mission, we'll be using the same 2014 world cup dataset. Rather than analyze the goals scored, we'll be writing a program that re-formats the date. We'll show you a variety of techniques to split strings, replace characters, concatenate strings etc. Near the end of this mission, we'll aggregate everything we've learned to write a *headline generator*.

To start off, let's load and preview our dataset!

instructions

* Load the "scores.csv" dataset into df.
* Preview the dataset using head() or str().

In the previous screen, notice that any columns containing text are Factor variables. When loading a csv file into a dataframe, the default action is to turn every character value into a factor.

Let's say we had the strings "Brazil" and "Croatia". Now, let's say we wanted to combine these two strings into one "Brazil Croatia" with a space in between. To combine two strings into one, we'll use the paste() function.

The paste() function will automatically convert the arguments into character type and concatenate them together. To use the paste() function, you'll need the following arguments:



paste(x..., sep = " ")

**x**: x is the object to be converted to character vectors. However, we are not limited to one object. We can combine multiple R objects into one.

**sep**: This is the value that will separate the values we're concatenating. This is not required. By default, the separator will be a space " ". You could also change the separator to a comma , or a dash - or any value you'd like separating the strings.

Now, to combine "Brazil" and "Croatia" into one string, we'd use paste() as follows:



paste("Brazil","Croatia",sep = " ")

This would return:



'Brazil Croatia'

Now, what if we had a vector of Brazil's opponents:



opponents <- c("Croatia","Mexico","Chile","Colombia","Germany","Netherlands")

Now, what if we wanted to concatenate "Brazil" onto each value within the vector to produce the following:



opponents <- c("Brazil Croatia","Brazil Mexico","Brazil Chile","Brazil Colombia","Brazil Germany","Brazil Netherlands")

To get this result, we can write the following:



paste("Brazil", opponents, sep=" ")

Similar to behavior in arithmetic operators, the recycling rule will take place. "Brazil" will be re-used and concatenated onto each value in the vector.

After previewing your dataset, notice that the match\_date column is missing a year. Let's concatenate "2014" to the match\_date.

instructions

* Use the head() function to preview your dataset so you can see what the values look like.
* Concatenate "-2014" to the end of each entry in thematch\_date column.
* Separate by a space.
* Store this in add\_2014.

In the previous section, we added "-2014" to the match\_date column. And although we used paste() to concatenate them together, this *does not* update the column on the original dataframe. To update a column in our dataframe, we'll need to *store* the values of the new column back into the original column. Like this:



original\_column <- new\_column

Using our current dataframe, let's say we wanted to subtract 1 goal from every row in the home goals column. To do this, we'd write the following:



df$home\_goals - 1

Now to actually see the difference in our original dataframe, we'll store this back into the original column:



df$home\_goals <- df$home\_goals - 1

Printing df$home\_goals would look like:



1] 2 0 0 2 2 1 0 0 1 2 1 -1 3 0 1 0 -1 -1 -1 1 1 -1 1 -1 1

[26] 0 0 0 1 1 0 1 0 0 -1 1 0 1 -1 -1 -1 -1 1 2 -1 1 -1 0 0 1

[51] 1 0 1 1 0 1 1 -1 -1 0 0 -1 -1 0

Now let's update the df$match\_date column with the added "-2014" from the previous screen!

instructions

* Update the df$match\_date with the new column containing "-2014".
* Store df$match\_date in update\_2014 like this: update\_2014 <- df$match\_date. This is for answer-checking purposes.

In the previous screen, we learned how to concatenate strings together. Looking at our current string, what if we wanted to extract a part of the string, like the month?

To extract values from a string, we'll be using the substr() function. The syntax for substr() is as follows:



substr(x, start, stop)

**x**: This is a character vector.

**start**: The first element by position number to be replaced.

**stop**: The last element by position number to be replaced.

Now, let's return to the string "Brazil Croatia". What if we wanted to extract the string "Brazil" from "Brazil Croatia"? Let's use substr():



substr("Brazil Croatia",1,6)

Since the first letter of "Brazil" is "B", this is at index 1, so we'll have the extract start at 1. The last letter of "Brazil" is "l" which falls on index 6. We'll have it end on 6.

Now, let's extract the *month* from match\_date.

instructions

* Extract the month of the match from match\_date.
* To extract the months of June and July, you'll notice that this solution isn't robust to different character positions. In your answer, include the dash for "July-".
* Store this in months.

In the previous screen, printing months will show 'July -'. Using substr() can extract the month, but the method isn't robust. Specifying a specific character position to extract will depend on the number of letters in each month. In addition,what if we wanted to use both the day of the month *and* the month from '12 July -2014'

Let's take our current extraction method on the string "Brazil Croatia":



# Extracting Brazil

substr("Brazil Croatia",1,6)

​

# Extracting Croatia

substr("Brazil Croatia",8,14)

Writing this will return:



'Brazil'

'Croatia'

However, what if we're extracting a string with a different number of characters? Using the same code on the string "Brazil Netherlands":



# Extracting Brazil

substr("Brazil Netherlands",1,6)

​

# Extracting Croatia

substr("Brazil Netherlands",8,14)

This would return:



'Brazil'

'Netherl'

The last four letters of "Netherlands" is cut off. While we could re-write our substr() code to return the full "Netherlands", we'd have to keep re-writing the code for different string lengths. An easier method, would be to *split* this string by a specific condition. To split a string, we'll use the strsplit() function. The syntax for strsplit() is as follows:



strsplit(x,split)

**x**: The character vector where each element is to be split.

**split**: The values to be split on. For example, if we want to split by spaces, we'd specify split= " ".

Let's try splitting our "Brazil Netherlands" string by space:



strsplit("Brazil Netherlands", split = " ")

This will return:



1. 'Brazil' 'Netherlands'

There's a key difference between the behavior of strsplit() & paste(). paste() will *automatically* convert a Factor vector into character type. strsplit() *will not* convert the vector automatically. If you input a factor variable into strsplit(), splitting this will cause an error. Use vector <- as.character(vector) to convert to character.

instructions

* Extract the match\_date column from df.
* Convert the match\_datecolumn into character type using as.character()
* Split all the dates in this column by space.
* Store this in date\_split.

Continuing with our match\_date column, notice the format of '12 July -2014'. Notice that the month(July) is in text while the day of the month is a number(12). We know, when manipulating dates, that there are a variety of formats we could use to format our dates. Looking at our current date, what if we wanted the entire date written in *numbers*. Rather than "June", it would indicate 6 instead.

To replace June with 6, we'll use the sub() function. The sub() function replaces the first occurence of a pattern with the new string. The syntax for sub() is as follows:



sub(pattern ,new\_string , x)

**pattern**: This is the string/expression we want to find to replace.

**new\_string**: This is the new string that will go in place of the old.

**x**: This is the character vector containing all the strings.

Let's look at the following string: "Brazil won the game.". Let's replace game with match:



sub("game","match","Brazil won the game.")

This would display:



'Brazil won the match.'

Now, let's use sub() to replace the June & July dates with 6 & 7.

instructions

* Within the df$match\_date column, replace all occurrences of June with "-06". Store this back in df$match\_date.
* Within the df$match\_date column, replace all occurrences of July with "-07". Store this back in df$match\_date.
* Store df$match\_date in updated\_dates.
* Print the column to see the results.

When printing df$match\_date, notice that there are spaces between our dates. To get our date into the correct format, we'll need to *remove* these spaces from our strings. In the previous section, we used sub() to replace our months with numerical months. To remove the whitespaces, we could use the same function to replace the spaces " " with no space "".

Let's look at the following string "Brazil defeated Croatia.". Let's say we wanted to remove the spaces in this string by using sub():



sub( " ", "", "Brazil defeated Croatia.")

This expression would display:



'Brazildefeated Croatia.'

Notice that not all the spaces were eliminated. This is because sub() only find thes *first* occurrence of the pattern and replaces. We'll need a function that replaces *all* occurences of a pattern. To replace all occurences, we'll use gsub().

The syntax of gsub() is the same as sub():



gsub(pattern ,new\_string , x)

**pattern**: This is the string/expression we want to replace.

**new\_string**: This is the new string that will go in place of the old.

**x**: This is the character vector containing all the strings.

Let's use gsub() on "Brazil defeated Croatia.":



gsub(" ", "", "Brazil defeated Croatia.")

And this will return:



'BrazildefeatedCroatia.'

We've removed all the spaces. Let's remove the spaces to our match\_date column!

instructions

* Remove all the spaces from the match\_date column.
* Store df$match\_date in remove\_space.

As we've manipulated dates in the string format, we run into a variety of limitations when using string dates. We can't perform any operations between two string dates to find the number of days. We can't check the day of the week. String dates limit the amount of information we can extract from our dataset. The solution to this problem, would be to *convert* our string dates to a date format.

To convert a string to a date, we'll be using the as.Date() function. as.Date() assumes the format ISO 8601 format: *yyyy-mm-dd*. Let's take the sample date "2014-06-25" and convert using as.Date():



as.Date("2014-06-25")

The result of this expression would produce this date in a datetime object:



2014-06-25

However, looking at our current df, our dates are structured as '12-06-2014' which is *dd-mm-yyyy*. This isn't in the correct format to convert to a date object. Let's try running as.Date() and see what happens if it's incorrect:



as.Date("12-06-2014")

This returns:



0012-06-20

The interpreter *will not* return an error. Instead, it'll return an *incorrect* date object. To ensure that as.Date() takes in the correct input, we'll add a format parameter when writing the function:



as.Date("25-06-2014", format="%d-%m-%Y")

d stands for the day. m stands for the month. Y stands for the year. Now, let's convert our match\_date column into date objects!

instructions

* Use the as.Date() function to convert the df$match\_date column into a column containing the dates as a date object.
* Store df$match\_date in date\_convert.

Now that we've transformed our match\_date column into a date object, what if we wanted to extract the day of the week to identify relationships between day of week & goals scored? We'll need to perform one more step where we convert the Date object into a POSIXlt object. POSIXlt is a list of date parts which will allow us to extract the values we want.

To convert our date object into a POSIXlt object, similar to converting the date object, we'll use the as.POSIXlt() function:



d <- as.Date("12-06-2014", format="%d-%m-%Y")

p <- as.POSIXlt(d)

Let's transform our date object column match\_date into a POSIXlt object!

instructions

* Transform the match\_date column into a POSIXlt object.
* Store the transformed column in pos\_obj.

Now that we've transformed our column into a POSIXlt object, we can now extract the dates from our match\_date column. We can use the following functions to extract from our object column, depending on what we're looking for:

object**$sec**: seconds(0-61)

object**$min**: minutes(0-59)

object**$hour**: hour(0-23)

object**$mday**: Day of the Month (1-31)

object**$mon**: Month(0-11)

object**$year**: Year(Years since 1900)

object**$wday**: Day of the week (0-6, 0 = Sunday)

object**$yday**: Day of the year (0-365)

If we wanted to extract the year from:



d <- as.Date("12-06-2014", format="%d-%m-%Y")

p <- as.POSIXlt(d)

We'd write it as:



p$year

This would return:



114

114 being 114 years from 1900. Be weary of the resulting format. Feel free to return to the list of date extracts if you forget. Now, let's extract the day of the week from our match\_date column!

instructions

* Extract the day of the week from our match\_date column
* Store this in dayofweek.

Now that we've created a new dayofweek vector, we'll need to store this as a column. Creating a column is similar to updating a column on our dataframe. Rather than store the vector back into an original column, we'll create a new column by doing the following:



df$new\_column <- vector

This way, we're not replacing our match\_date column. Replacing this would cause us to lose information. Instead, we're creating a separate column to store these values. Let's create a new dayofweek column!

instructions

* Create a new column called dayofweek in df. Store df$match\_date$wday in the new column.
* Store the new df$dayofweek column in dayofweek. This is for purpose of answer checking.

Now that we've learned the basics of how to manipulate strings, let's use our current soccer/football data to build an automatic headline generator! This headline generator will tell us the winner of the match and the score. An example headline would look like this:

" On June 12th, Brazil won the match 3-1"

Let's create this headline generator!

instructions

* First thing we'll want to do, is extract the month from match\_date.
  + Do this by creating a new column called month.
  + Use object$mon to extract month.
  + Replace 5 and 6 with "June" and "July" using sub or gsub. Month starts at 0.
* Identify the columns needed in the phrase
* Identify what additional strings we'll need to add i.e the "th" in "12th"
* For answer checking purposes, don't worry about "st" in "1st" or "nd" in "2nd"
* Remove any unnecessary spaces.
* Create the headline column
* Store this headline column in variable called headline.

Throughout this course, we've explored a variety of functions that we can use to manipulate strings & dates. We learned how to extract values from date objects to get useful information. We also learned how to create new columns & gave you a taste of what "data cleaning" looks like.

To finish up this course, in the next mission we'll help you get set up with RStudio so you can start writing your own R code on your own machine. RStudio is an integrated environment that lets you run and manage R code, test small chunks of code using the shell, visualize plots easily, and more!