Regular Expressions in Practice: R

R comes with base functions for using regular expressions. To get started, enter R or R Studio and read in the csv file.
Note that we didn't create a header in the csv file, so set header=FALSE. We also want to read in strings rather than factors,
so set as.is=TRUE.

```
> data <- read.csv('example_data_leaders.csv', header=FALSE, as.is=TRUE)
```

Let's start with the function grep(). It takes two arguments: the regular expression pattern to match, then a vector of strings to
match to the expression. By default, it returns a vector of indices of the elements in the input vector that matched the expression.
If we want it to return the strings that matched, we can set value=TRUE.

Let's say that we want to find all of our leaders' prior experience that involved a cabinet minister position. To keep this simple,
we'll just assume the prior experience is in the last column in our data frame, and we'll pass that column as a vector into grep,
to look for the values in that column that match a regular expression for "minister".

Recall that regular expressions are case sensitive, so let's search for both capitalized and lowercase ministers with [Mm]inister.

```
> grep('[Mm]inister', data[,ncol(data)], value=TRUE)
```

```
[1] "lawyer, legislator, party leader, minister of education, minister of
justice"
[2] "economist, professor, minister of finance"
[3] "physician, professor, government agency official, cabinet minister, vice
president"
[4] "legislator, mayor, party leader, cabinet minister"
[5] "minister of public works"
```

Great, we found a lot. But we see now that it's printing out the leader's entire list of positions, and we might only want
to look at the position that had "minister" in it. Let's capture the specific position in a group and just print out that
group. We want to make sure we capture the full position, not just the word "minister". So we want to get all of
the characters before and after that word that are not a comma (since the positions are separated by commas in the string).

In R, this is a two-part step. We can use the function regexpr(), which returns a vector containing the character positions
in each string in the input vector where the regular expression found a match. We can then pass the output from regexpr() into
the function regmatches(), along with the original vector of strings again, to get the exact sequences of text that matched.

```
> matches <- regexpr('[^,]*[Mm]inister[^,]*', data[,ncol(data)])
> regmatches(data[,ncol(data)], matches)
```

```
[1] " Minister of Energy"      " Minister of Education"   " Minister of
Finance"
[4] " cabinet minister"        " cabinet minister"        "Minister of Public
Works"
```

We're getting specific positions, but we're only getting one per leader, and some leaders held more than one cabinet office. We can use
the function gregexpr() to find all matches to an expression within each string. Since these R functions operate on vectors,
instead of a single vector of output, we're now going to get a list of vectors (one for each string in the input vector). Let's
assign this to a variable and then use unlist so we just end up with a vector of all of the positions that matched.

```
> matches <- gregexpr('[^,]*[Mm]inister[^,]*', data[,ncol(data)])
> positions <- regmatches(data[,ncol(data)], matches)
> unlist(positions)
```

```
[1] " Minister of Energy"      " Minister of Economy"     " Prime Minister"
[4] " Minister of Education"   " Minister of Justice"     " Minister of
Finance"
[7] " cabinet minister"        " cabinet minister"        "Minister of Public
Works"
```

What if we want to replace these different positions with a common term, e.g. use cabinet minister for all of them? We can use the
functions sub() (to replace the first instance) or gsub() (to replace all matches). Both functions take three arguments:
a regular expression pattern to find, a string to substitute wherever that expression matches, and a vector of strings in which we
want to find and replace.

Let's use the same regular expression from above, but now we'll use gsub() and add a replacement string cabinet minister.

```
> gsub('[^,]*[Mm]inister[^,]*', 'cabinet minister', data[,ncol(data)])
```

```
 [1] "economist, business executive,cabinet minister,cabinet minister,cabinet
minister"
 [2] "military officer"
 [3] "legislator, party leader"
 [4] "economist, consultant for international organizations"
 [5] "lawyer, legislator, party leader,cabinet minister,cabinet minister"
 [6] "mathematician, professor, dean"
 [7] "economist, professor,cabinet minister"
 [8] "physician, professor, government agency official,cabinet minister, vice
president"
 [9] "military colonel"
[10] "governor, ministry official, chancellor/dean, vice president"
[11] "legislator, mayor, party leader,cabinet minister"
[12] "legislator, head of legislature"
[13] "head of police service, college instructor, mayor"
[14] "cabinet minister"
[15] "legislator, professor"
```

The function gsub() returns the input string with the substitutions made. It doesn't change the original string object. So since
we didn't save that output, it just printed to the screen. Let's put that output back into the last column in our dataframe.

We can see from the printed output, though, that we've ended up with several copies of "cabinet office" where there were multiple
minister positions. We might want to replace multiple copies with just one copy. And we replaced the initial space when the position
came after a comma, so let's add a space whenever we see an instance of "cabinet office" after a comma. We'll save this to the
original data frame as well, then print them out to make sure they look right.

```
> data[,ncol(data)] <- gsub('[^,]*[Mm]inister[^,]*', 'cabinet minister',
data[,ncol(data)])
> data[,ncol(data)] <- gsub('(,cabinet minister)+', ', cabinet minister',
data[,ncol(data)])
> data[,ncol(data)]
```

```
 [1] "economist, business executive, cabinet minister"
 [2] "military officer"
 [3] "legislator, party leader"
 [4] "economist, consultant for international organizations"
 [5] "lawyer, legislator, party leader, cabinet minister"
 [6] "mathematician, professor, dean"
 [7] "economist, professor, cabinet minister"
 [8] "physician, professor, government agency official, cabinet minister, vice
president"
 [9] "military colonel"
[10] "governor, ministry official, chancellor/dean, vice president"
[11] "legislator, mayor, party leader, cabinet minister"
[12] "legislator, head of legislature"
[13] "head of police service, college instructor, mayor"
[14] "cabinet minister"
[15] "legislator, professor"
```

Now this looks right!