Introduction to R

Greg Ridgeway (gridge@upenn.edu)
Ruth Moyer (moyruth@upenn.edu)

October 23, 2017

Introduction

A regular expression is a sequence of characters that defines a search pattern. Sometimes we use regular expressions to help us find a pattern in text that we want, like the Find functionality in Word. Other times, we use a regular expression to help us find and *replace* a piece of text, like the Find and Replace Functionality in Word. The two main R functions that we will learn about are `grep()` and `gsub()`. You may use them in verb form (e.g., "I need to grep all the gun cases," "I was grepping like crazy to find all the opioid cases," "I first had to gsub the commas with semicolons"). Regular expressions are available in numerous other software packages, so everything you learn here will port over to using regular expressions in Linux, Stata, Python, Java, ArcGIS, and many others.

We've already used regular expressions in previous work that we've done. We had to clean up the NCVS data since the text description of crime types changed from one year to the next. In 2012 the NCVS crime type had a leading 0, like "(01) Completed rape" and then in 2013 the crime type did not have the leading 0 like "(1) Completed rape". In order to tabulate the number of rapes in the calendar year 2012, we had to combine the data collected in 2012 and 2013 and reformat the crime types with rather mysterious looking code `gsub("\\(([1-9])\\)", "(0\\1)", data.inc$V4529)`. This will all become clear in these notes.

Finding patterns in text with `grep()`

First we'll learn about `grep()`, which was first developed in the early 1970s. `grep()` searches data for lines that match a given expression. The name `grep` is an acronym for "globally search a regular expression and print." Some tangible examples will help us see how `grep()` works.

First, run the following line that provides a string of text with a diverse range of capitalization, letters, punctuation, and other features.

```
dataText <- c("suspect",
              "three suspects",
              "3 suspects",
              "9mm firearm",
              "483 McNeil Building",
              "3718 Locust Walk",
              "4 Privet Drive",
              "10880 Malibu Point",
              "Philadelphia, PA",
              "Philly",
              "Phila",
              "Phil.",
              "Phil Dunphy",
              "19104-6286",
              "20015",
              "90291",
              "90210",
              "(215) 573-9097",
              "215-573-9097",
              "2155739097")
```

Let's find all of the items that have the letter "a" in `dataText`.

```
grep("a",dataText)
```

```
[1]  4  6  8  9 11
```

This shows the indices of which elements of `dataText` have an "a" in them. Sure enough the fourth element, "9mm firearm", has an "a" and you can confirm that the other indices `grep()` returned also indicate elements of `dataText` that have an "a".

For most of these notes we're going to set `value=TRUE`, which will return the actual elements matched rather than their indices. This will make it easier to check that `grep()` is finding the elements that we are expecting to match.

```
grep("a",dataText,value=TRUE)
```

```
[1] "9mm firearm"        "3718 Locust Walk"   "10880 Malibu Point"
[4] "Philadelphia, PA"   "Phila"
```

As you can see, `grep()` uses the following general syntax: grep("what we're searching for", the text, and an optional value=TRUE if we want to return text). If we put `value=FALSE` or don't put `value=TRUE`, we will just receive the index in the text where we can find what we're searching for.

Let's try another example. Instead of a letter, let's try to find a number such as "1". Specifically, which items in `dataText` have a "1" in them?

```
grep("1",dataText,value=TRUE)
```

```
[1] "3718 Locust Walk"   "10880 Malibu Point" "19104-6286"
[4] "20015"              "90291"              "90210"
[7] "(215) 573-9097"     "215-573-9097"       "2155739097"
```

We can also search for multiple characters - instead of individual characters. We place our list of desired characters within square brackets `[ ]` within the quotes. For example, let's find items that contain numbers.

```
grep("[0123456789]",dataText,value=TRUE)
```

```
 [1] "3 suspects"         "9mm firearm"        "483 McNeil Building"
 [4] "3718 Locust Walk"   "4 Privet Drive"     "10880 Malibu Point"
 [7] "19104-6286"         "20015"              "90291"
[10] "90210"              "(215) 573-9097"     "215-573-9097"
```

```
[13] "2155739097"
```

If we wanted items that contained an odd number, we could do this.

```
grep("[13579]",dataText,value=TRUE)
```

```
 [1] "3 suspects"          "9mm firearm"         "483 McNeil Building"
 [4] "3718 Locust Walk"    "10880 Malibu Point"  "19104-6286"
 [7] "20015"               "90291"               "90210"
[10] "(215) 573-9097"      "215-573-9097"        "2155739097"
```

So the [] in a regular expression means "match any of these characters." We can also place square brackets next to each other. For example, let's say we wanted to find an item in dataText that has four adjacent numbers.

```
grep("[0-9][0-9][0-9][0-9]",dataText,value=TRUE)
```

```
[1] "3718 Locust Walk"   "10880 Malibu Point" "19104-6286"
[4] "20015"              "90291"              "90210"
[7] "(215) 573-9097"     "215-573-9097"       "2155739097"
```

Note that we can use the shorthand 0-9 to mean any number between 0 and 9, including 0 and 9. This regular expression says "find a number, followed by a number, followed by another number, followed by another number." Alternatively we can use {4} to mean "match four of the previous character".

```
grep("[0-9]{4}",dataText,value=TRUE)
```

```
[1] "3718 Locust Walk"   "10880 Malibu Point" "19104-6286"
[4] "20015"              "90291"              "90210"
[7] "(215) 573-9097"     "215-573-9097"       "2155739097"
```

{n} means the preceding item will be matched exactly n times. Note that this also matches text that has five or more numbers in a row, since if they have five numbers in a row, then they will also have four numbers in a row. Later we will learn about how to find exactly four numbers in a row.

We can also use the squiggly brackets to make a range.

```
grep("[0-9]{5,10}",dataText,value=TRUE)
```

```
[1] "10880 Malibu Point" "19104-6286"         "20015"
[4] "90291"              "90210"              "2155739097"
```

Thus {n,m} will find something matched at least n times, but not more than m times. We've used examples above with numbers, but you can apply this syntax to letters.

```
grep("[a-zA-Z]{5}",dataText,value=TRUE)
```

```
 [1] "suspect"             "three suspects"      "3 suspects"
 [4] "9mm firearm"         "483 McNeil Building" "3718 Locust Walk"
 [7] "4 Privet Drive"      "10880 Malibu Point"  "Philadelphia, PA"
[10] "Philly"              "Phila"               "Phil Dunphy"
```

Note how we used the shorthand a-z to mean any lowercase letter and A-Z to mean any uppercase letter. grep() will match a different set of elements if we just search for lower case letters.

```
grep("[a-z]{5}",dataText,value=TRUE)
```

```
 [1] "suspect"              "three suspects"      "3 suspects"
 [4] "9mm firearm"          "483 McNeil Building" "3718 Locust Walk"
 [7] "4 Privet Drive"       "10880 Malibu Point"  "Philadelphia, PA"
[10] "Philly"               "Phil Dunphy"
```

Searches using `grep()` are case-sensitive. For example, let's find all items that contain capital letters.

```
grep("[A-Z]",dataText,value=TRUE)
```

```
[1] "483 McNeil Building" "3718 Locust Walk"    "4 Privet Drive"
[4] "10880 Malibu Point"  "Philadelphia, PA"    "Philly"
[7] "Phila"               "Phil."               "Phil Dunphy"
```

Exercises

1. Write a regular expression that will match these `c("A1","B1","C1")` but not these `c("D1","E1","F1")`. Hint: Think about how A1, B1, C1 differ from the other three and fill in the pattern in the first part of `grep("", c("A1","B1","C1","D1","E1","F1"), value=TRUE)`
2. Write a regular expression that will match these `c("1A","2B","3C")` but not these `c("A1","B2","C3")`. Hint: As in the previous one, what is different about 1A, 1B, 1C? Then fill in the pattern in the first part of `grep("", c("1A","2B","3C","A1","B2","C3"), value=TRUE)`

More Symbols That Help Us with grepping

So far you have seen that we use `[]` to match any character listed between the `[]`, the `-` to specify a range to match like `0-9`, `a-z`, and `A-Z`, and the `{}` to match the previous character multiple times.

In additional to these, there are several more symbols or sequences of symbols that are useful. The number of symbols on the keyboard are fairly limited compared to the numerous combinations of patterns of text we might wish to find. As a result you will see that some of these symbols are used in very different ways depending on the context.

Carets ^

Let's look at the caret ^ symbol. When we put a ^ within square brackets, it means "not". Let's try to find text in `dataText` that has something that is not a letter immediately followed by a letter.

```
grep("[^A-Za-z ][A-Za-z]",dataText,value=TRUE)
```

```
[1] "9mm firearm"
```

"9mm firearm" has a character that is not a letter (9) immediately followed by a letter (m).

When we put the ˆ outside of square bracket, it means "the beginning of the text". For example, the following regular expression matches text where the first character is either an upper-case or a lower-case letter.

```
grep("^[A-Za-z]",dataText,value=TRUE)
```

```
[1] "suspect"           "three suspects"    "Philadelphia, PA"
[4] "Philly"            "Phila"             "Phil."
[7] "Phil Dunphy"
```

Dollar Signs $

While we use carets to signal the beginning of the text, the dollar sign signals the end of the text. For example, to search for all items that end with either an upper-case or lower-case letter, we would do the following:

```
grep("[A-Za-z]$",dataText,value=TRUE)
```

```
 [1] "suspect"              "three suspects"      "3 suspects"
 [4] "9mm firearm"          "483 McNeil Building" "3718 Locust Walk"
 [7] "4 Privet Drive"       "10880 Malibu Point"  "Philadelphia, PA"
[10] "Philly"               "Phila"               "Phil Dunphy"
```

Note how all of these have a letter as the last character. We can be more specific and ask for text that end with the letter "y" or end with a 7.

```
grep("y$",dataText,value=TRUE)
grep("7$",dataText,value=TRUE)
```

```
[1] "Philly"      "Phil Dunphy"
[1] "(215) 573-9097" "215-573-9097"   "2155739097"
```

Exercises

3. Write a regular expression that will match these `c("123","567","314")` but not these `c("1234","5678","3141")`. Hint: How can you use ˆ and $? Fill in in the first part of `grep("",c("123","567","314","1234","5678","3141"), value=TRUE)`
4. Write a regular expression that will match these `c("123ABC","234BCDEF","435C")` but not these `c("1ABC23","2468BC","1234C5")`. Hint: How can you use ˆ and $? Fill in the first part of `grep("", c("123ABC","234BCDEF","435C","1ABC23","2468BC","1234C5"), value=TRUE)`

Plus Sign +

The + means at least one of the previous. For example, suppose we wanted to find items in our list that have numbers and those numbers are followed by letters.

```
grep("[0-9]+[A-Za-z]+",dataText,value=TRUE)
```

```
[1] "9mm firearm"
```

Or search for text that starts with some numbers, then has a space, followed by some letters, and then ends.

```
grep("^[0-9]+ [A-Za-z]+$",dataText,value=TRUE)
```

```
[1] "3 suspects"
```

Parentheses `()`

Parentheses group together characters as words. For example, suppose we wanted to find the word "suspects".

```
grep("(suspects)",dataText,value=TRUE)
```

```
[1] "three suspects" "3 suspects"
```

On its own the parentheses are no different than just running the regular expression "suspects" with no parentheses. However, in combination with `|` and `?` it gets more interesting and powerful.

Vertical Bar `|`

The vertical bar functions as an "or." Suppose we wanted to get both the phrases "three suspects" and "3 suspects" from `dataText`. We would use parentheses as well as a vertical bar.

```
grep("(three|3) suspects",dataText,value=TRUE)
```

```
[1] "three suspects" "3 suspects"
```

Let's `grep()` both DC and LA ZIP codes.

```
grep("(902|200)[0-9]{2}",dataText,value=TRUE)
```

```
[1] "20015" "90291" "90210"
```

Here's how we can find words that have exactly four characters. To be exactly four characters, on either side of them there needs to be a space or the beginning or end of the line.

```
grep("( |^)[A-Za-z]{4}( |$)",dataText,value=TRUE)
```

```
[1] "3718 Locust Walk" "Phil Dunphy"
```

Note that this did not select "Phil." since it has a period following it. We'll need a more generaly regular expression to select that one as well.

Let's try to find phone numbers in `dataText`. The problem is that phone numbers have three different formats in our data. The easiest phone number pattern to find is the one with 10 digits in a row.

```
grep("[0-9]{10}",dataText,value=TRUE)
```

```
[1] "2155739097"
```

It is also not too hard to find the hypenated phone number pattern.

```
grep("[0-9]{3}-[0-9]{3}-[0-9]{4}",dataText,value=TRUE)
```

```
[1] "215-573-9097"
```

The phone number format with parentheses needs a little more caution. We pointed out earlier that parentheses have special meaning in regular expressions. In fact, there are several "special characters" in regular expressions, `\ ^ $ {} [] () . * + ? | -`. If you actually want to search for the symbol itself you need to "protect" it with a backslash `\`. However, the `\` is a special character for R too and R will think that something special is coming next after the `\`. So to tell R "no really, I really want a backslash here" you have to protect the backslash too. So to look for those phone numbers with parentheses we use a regular expression like this.

```
grep("\\([0-9]{3}\\) [0-9]{3}-[0-9]{4}",dataText,value=TRUE)
```

```
[1] "(215) 573-9097"
```

Now, let's put all of these patterns together use `|` to search for any phone number in any of the three formats.

```
grep("[0-9]{10}|[0-9]{3}-[0-9]{3}-[0-9]{4}|\\([0-9]{3}\\) [0-9]{3}-[0-9]{4}",
     dataText, value=TRUE)
```

```
[1] "(215) 573-9097" "215-573-9097"   "2155739097"
```

Question Mark `?`

The question mark indicates optional text. To illustrate,

```
grep("Phil(adelphia)",dataText,value=TRUE)
grep("Phil(adelphia)?",dataText,value=TRUE)
```

```
[1] "Philadelphia, PA"
[1] "Philadelphia, PA" "Philly"           "Phila"
[4] "Phil."            "Phil Dunphy"
```

The first one is no different from searching for the word "Philadelphia," but the second one says that the "adelphia" part is optional. Note that this regular expression also picked up "Phil Dunphy". Again we will need a more careful regular expression to avoid matching this name and only select abbreviations and nicknames for Philadelphia.

Boundaries `\b`

`\b` will try to find boundaries around words. This includes spaces, punctuation, and the beginning and end of the text. So another way to find all text with four letter words, including "Phil.", is

```
grep("\\b[A-Za-z]{4}\\b",dataText,value=TRUE)
```

```
[1] "3718 Locust Walk" "Phil."            "Phil Dunphy"
```

Remember, the \ is a special character for R. To "protect" the backslash we put another backslash in front of it. If we did not include the `\b` in this regular expression, then we would have also matched words with five or more letters too.

Exercises

Write regular expressions to find the following items in `dataText`.

5. Find commas
6. Find ZIP codes
7. Find those with six letter word
8. Find mentions of Philadelphia
9. Find capitalized words
10. Find the addresses
11. Find a shorter way to get phone numbers using `?`

Finding and replacing patterns in text with `gsub()`

`gsub()` stands for "global substitution." It is useful for automating the editing of text, including deleting characters from text or extracting only parts of text. For example, let's try to remove all numbers from our `dataText` list.

```
gsub("[0-9]","",dataText)
```

```
 [1] "suspect"           "three suspects"   " suspects"
 [4] "mm firearm"        " McNeil Building" " Locust Walk"
 [7] " Privet Drive"     " Malibu Point"    "Philadelphia, PA"
[10] "Philly"            "Phila"            "Phil."
[13] "Phil Dunphy"       "-"                ""
[16] ""                  ""                 "() -"
[19] "--"                ""
```

The first parameter of the `gsub()` function is the "find pattern," what we are asking `gsub()` to find. The second parameter is a "replacement pattern" that will replace whatever `gsub()` matched with the find pattern. Lastly, like `grep()`, we need to tell `gsub()` the name of the data object in which it should look for the pattern.

With `gsub()` we will be using a lot of the same grammar, such as carats, brackets, and dashes, that we used with `grep()`. To illustrate, let's remove anything that is not a number from text.

```
gsub("[^0-9]","",dataText)
```

```
 [1] ""           ""           "3"          "9"          "483"
 [6] "3718"       "4"          "10880"      ""           ""
[11] ""           ""           ""           "191046286"  "20015"
[16] "90291"      "90210"      "2155739097" "2155739097" "2155739097"
```

Exercises

12. Remove the commas from `c("9,453","23,432","4,334,645","1,234")`

Back-Referencing

A bit more complicated aspect of `gsub()` is "backreferencing." Any part of the regular expression in the find pattern that is wrapped in parentheses gets stored in a "register." You can have multiple pairs of parentheses and, therefore, save different parts of what gets matched in the find pattern. You can then recall what `gsub()` stored in the registers in the replacement pattern, using `\\1` for what was stored in the first register, `\\2` for what was stored in the second register, and so on.

For example, let's delete the "plus four" part of the ZIP codes in our data.

```
gsub("([0-9]{5})-[0-9]{4}","\\1",dataText)
```

```
 [1] "suspect"              "three suspects"       "3 suspects"
 [4] "9mm firearm"          "483 McNeil Building"  "3718 Locust Walk"
 [7] "4 Privet Drive"       "10880 Malibu Point"   "Philadelphia, PA"
[10] "Philly"               "Phila"                "Phil."
[13] "Phil Dunphy"          "19104"                "20015"
[16] "90291"                "90210"                "(215) 573-9097"
[19] "215-573-9097"         "2155739097"
```

We wrapped the parentheses around the first five digits, so only those first five digits get stored in register 1. Then the replacement pattern replaces everything that the find pattern matched (the entirety of the five digits, the hyphen, and the plus four digits) with the contents of register 1, containing just the first five digits.

Let's use `gsub()` to just keep the first two "words" in each element of `dataText`.

```
gsub("^([^ ]+ [^ ]+) .*$","\\1",dataText)
```

```
 [1] "suspect"          "three suspects"   "3 suspects"
 [4] "9mm firearm"      "483 McNeil"       "3718 Locust"
 [7] "4 Privet"         "10880 Malibu"     "Philadelphia, PA"
[10] "Philly"           "Phila"            "Phil."
[13] "Phil Dunphy"      "19104-6286"       "20015"
[16] "90291"            "90210"            "(215) 573-9097"
[19] "215-573-9097"     "2155739097"
```

This regular expression says: start at the beginning of the text, find a bunch of characters that are not spaces (remember that the + means one or more of the previous), then find a space, then find another bunch of characters that are not spaces, followed by another space, followed by anything else until the end of the text. The `.` is like a wild card. It matches any one character. the `*` is like the + but it means zero or more of the previous character (the + matches or more of the previous character). Note how we have used the parentheses. They are wrapped around those first two words. Those words get stored in register 1 and the replacement pattern just recalls whatever got stored in register 1.

An alternative strategy is to use `\w`, which means a "word character", any numbers of letters.

```
gsub("^(\\w+ \\w+) .*$","\\1",dataText)
```

```
 [1] "suspect"           "three suspects"   "3 suspects"
 [4] "9mm firearm"       "483 McNeil"       "3718 Locust"
 [7] "4 Privet"          "10880 Malibu"     "Philadelphia, PA"
[10] "Philly"            "Phila"            "Phil."
[13] "Phil Dunphy"       "19104-6286"       "20015"
[16] "90291"             "90210"            "(215) 573-9097"
[19] "215-573-9097"      "2155739097"
```

When working with `grep()`, we wrote a regular expression to find all the phone numbers in all the various formats. Now let's use `gsub()` to standardize all the phone numbers to have the hyphenated format, like 123-456-7890. We use the same regular expression we used when using `grep()` to find phone numbers, but we insert pairs of parentheses to capture the phone number digits.

```
gsub("^\\(?([0-9]{3})(\\) |-)?([0-9]{3})-?([0-9]{4})","\\1-\\3-\\4",dataText)
```

```
 [1] "suspect"             "three suspects"       "3 suspects"
 [4] "9mm firearm"         "483 McNeil Building" "3718 Locust Walk"
 [7] "4 Privet Drive"      "10880 Malibu Point"  "Philadelphia, PA"
[10] "Philly"              "Phila"                "Phil."
[13] "Phil Dunphy"         "19104-6286"           "20015"
[16] "90291"               "90210"                "215-573-9097"
[19] "215-573-9097"        "215-573-9097"
```

Note that register 2 captures whatever text matches the optional text `(\\) |-)`. That's why you don't see \\2 in the replacement pattern.

Now let's go back to that regular expression we used to standardize the NCVS crime types, `gsub("\\(([1-9])\\)", "(0\\1)", data.inc$V4529)`. You can now see what it is doing: find a single digit number inside parentheses, store that number in register 1, replace it with the same number but with a leading 0.

Class exercise

13. Add commas to these numbers `c("9453","2332","4645","1234")`. That is, make these numbers look like 9,453 and 2,332 and so on. Fill in the find pattern and replacement pattern in `gsub("", "", c("9453","2332","4645","1234"))`

Lookaheads

Although not as commonly used as a backreference, a "lookhead" can be helpful to find what comes next - or more generally to `gsub()` items that are a bit more complicated. Here is an illustration of what a lookahead does. Let's say you wanted to check that every q is followed by a u. If it's not, then insert the u after the q.

Maybe you would do something like this

```
gsub("q[^u]","qu",c("quiet","quick","quagmire","qixotic"))
```

```
[1] "quiet"    "quick"    "quagmire" "quxotic"
```

As you can see, it doesn't quite do what we wanted. It problematically drops the "i" from "quixotic." Remember that the replacement pattern will overwrite whatever matches the find pattern. This find pattern will match the "qi," the q *and* the adjacent character that is not a u.

A "lookahead" just peeks at the next character to see what it is, but does not consider it part of the match. In parentheses we signal a lookahead with `?` and then to ask for a character that is not a "u" we use `!u`.

```
gsub("q(?!u)","qu",c("quiet","quick","quagmire","qixotic"),perl=TRUE)
```

```
[1] "quiet"    "quick"    "quagmire" "quixotic"
```

Now we have "quixotic" spelled properly as `gsub()` looked ahead to check for a "u" but did not consider it part of the match to be replaced. Note that we have set `perl=TRUE`. There is not a single regular expression standard. Lookaheads (and there are lookbehinds too) are not part of the POSIX 1003.2 standard that R uses by default. However, you can ask R to use perl-style regular expressions that do support lookaheads (and lookbehinds) by simply setting `perl=TRUE`.

Here's how lookaheads are going to be very important for us in working with data. We often get datasets in comma-separated value format, typically with a ".csv" extension on the file name. The R function `read.csv()` can read in data in this format. Problems can occur when values in the dataset have commas inside of them.

Here's some example text that could be problematic.

```
text.w.quotes <- 'white,male,"loitering,narcotics",19,"knife,gun"'
cat(text.w.quotes,"\n")
```

```
white,male,"loitering,narcotics",19,"knife,gun"
```

Some of the commas in this text are separating values for race, sex, arrest charge, age, and recovered contraband. However, there are other commas inside the quotes listing the arrest charges and the contraband items. Fortunately, `read.csv()` is smart enough to keep quoted items together as one data element, as long as the parameter `quote = "\"'"`, which is the default. Other functions are not so kind. Later we will use `dbWriteTable()` to build an SQL database and it will think that all the commas separate the values. So it will think there is a separate `"loitering` data element and then a `narcotics"`. And the same for `"knife` and `gun"`.

So here is a very handy regular expression using lookaheads that changes commas that are not inside quotes to semicolons. You can also choose a stranger symbol, like `|` or `@`. This regular expression looks for a `,` and then it uses a lookahead for the reminder of the line. It will match that comma is there are no quotes for the rest of the line, `[^\"]`, or if there are only pairs of quotes each with non-quote characters inside of them, `\"[^\"]*\"`.

```
gsub(",(?=([^\"]|\"[^\"]*\")*$)",";",text.w.quotes,perl=TRUE)
```

```
[1] "white;male;\"loitering,narcotics\";19;\"knife,gun\""
```

As you can see, commas inside the quotes are preserved and those outside have been transformed into `;`. Now we would be able to tell a function like `dbWriteTable()` that the data elements are separated by `;` and it will read in the data properly.

Exercises

Make the following changes to `dataText`.

14. Spell out PA as Pennsylvania
15. Spell out Philadelphia
16. Change Phil to Phillip (where appropriate)
17. Keep just the first word or first number
18. Keep only area codes of phone numbers

Introduction to Webscraping: A Practical Application of Regular Expressions

As you've already seen, regular expressions are an extremely valuable tool when working with data. In fact, we're going to learn about webscraping next. Webscraping enables us to extract data from a website by searching the underlying HTML code for the website and extracting the desired data. To do webscraping, you're going to rely heavily on regular expressions.

Suppose we want a list of the 10,000 most common words in English. wiktionary.org provides this list. Have a look at the webpage. Although it says there are 10,000 words listed on this page, #2781 is missing, so really there are only 9,999 words listed. We hope this does not come as a surprise to you, but some things posted on the web are not quite accurate.

The `scan()` function in R can read data from a text file, but if given a URL, it will download the HTML code for that page

```
words <- scan("http://en.wiktionary.org/wiki/Wiktionary:Frequency_lists/PG/2006/04/1-10000",
          what="",sep="\n")
```

If you get an error saying this page is forbidden, try changing the `http` to `https`. If instead you got a message like "Read 50247 items" then you're in luck. You just used R to scrape a webpage. You might get a number different from 50247 and it might change if you run the script another day. Websites make updates to headers, footers, and banners and these changes change the number of lines of HTML code required to generate the page.

The first several lines of HTML code in `words` all contain HTML code to set up the page

```r
words[1:6]
```

```
[1] "<!DOCTYPE html>"
[2] "<html class=\"client-nojs\" lang=\"en\" dir=\"ltr\">"
[3] "<head>"
[4] "<meta charset=\"UTF-8\"/>"
[5] "<title>Wiktionary:Frequency lists/PG/2006/04/1-10000 - Wiktionary</title>"
[6] "<script>document.documentElement.className = document.documentElement.className.replace( /(
```

There's nothing of interest to us here. We just want the part with the 9,999 most common words. Let's look further down the page.

```r
words[300:310]
```

```
 [1] "<td>1980046</td>"
 [2] "</tr>"
 [3] "<tr>"
 [4] "<td>51</td>"
 [5] "<td><a href=\"/wiki/there\" title=\"there\">there</a></td>"
 [6] "<td>1961200</td>"
 [7] "</tr>"
 [8] "<tr>"
 [9] "<td>52</td>"
[10] "<td><a href=\"/wiki/if\" title=\"if\">if</a></td>"
[11] "<td>1951102</td>"
```

Here we start finding some words! Note that every line that has one of the words we are looking for has `title=\"`. We can use that phrase to find lines that have the 9,999 words on them. Use `grep()` to find those lines and print out the first 10 of them to see if this will work for us.

```r
words <- grep("title=\"", words, value=TRUE)
words[1:10]
```

```
 [1] "<link rel=\"alternate\" type=\"application/x-wiki\" title=\"Edit\" href=\"/w/index.php?tit
 [2] "<link rel=\"edit\" title=\"Edit\" href=\"/w/index.php?title=Wiktionary:Frequency_lists/PG/
 [3] "<link rel=\"search\" type=\"application/opensearchdescription+xml\" href=\"/w/opensearch_d
 [4] "<link rel=\"alternate\" type=\"application/atom+xml\" title=\"Wiktionary Atom feed\" href=
 [5] "\t\t\t\t\t\t\t\t\t<div id=\"contentSub\"><span class=\"subpages\">&lt; <a href=\"/wiki/Wikti
 [6] "\t\t\t\t<div id=\"mw-content-text\" lang=\"en\" dir=\"ltr\" class=\"mw-content-ltr\"><div
 [7] "<td><a href=\"/wiki/the\" title=\"the\">the</a></td>"
 [8] "<td><a href=\"/wiki/of\" title=\"of\">of</a></td>"
 [9] "<td><a href=\"/wiki/and\" title=\"and\">and</a></td>"
[10] "<td><a href=\"/wiki/to\" title=\"to\">to</a></td>"
```

While the first 6 lines do have the phrase `title=\"`, they aren't the ones with the 9,999 words. But starting at line 7 we start seeing the most common words: the, of, and, to. So let's cut the first 6 lines from `words` and keep the next 9,999 lines. `title=\"` shows up more in the HTML code in the footer after the 9,999th word.

```r
words <- words[-(1:6)]
words <- words[1:9999]
```

```r
head(words) # look at the first several rows
tail(words) # look at the last several rows
```

```
[1] "<td><a href=\"/wiki/the\" title=\"the\">the</a></td>"
[2] "<td><a href=\"/wiki/of\" title=\"of\">of</a></td>"
[3] "<td><a href=\"/wiki/and\" title=\"and\">and</a></td>"
[4] "<td><a href=\"/wiki/to\" title=\"to\">to</a></td>"
[5] "<td><a href=\"/wiki/in\" title=\"in\">in</a></td>"
[6] "<td><a href=\"/wiki/i\" title=\"i\">i</a></td>"
[1] "<td><a href=\"/wiki/film\" title=\"film\">film</a></td>"
[2] "<td><a href=\"/wiki/repressed\" title=\"repressed\">repressed</a></td>"
[3] "<td><a href=\"/wiki/cooperation\" title=\"cooperation\">cooperation</a></td>"
[4] "<td><a href=\"/wiki/sequel\" title=\"sequel\">sequel</a></td>"
[5] "<td><a href=\"/wiki/wench\" title=\"wench\">wench</a></td>"
[6] "<td><a href=\"/wiki/calves\" title=\"calves\">calves</a></td>"
```

These lines have the text we need, but there are still a lot of HTML tags, all that HTML code wrapped in `< >`. `<td>` is the HTML tag marking a cell in a table and `<a>` is the HTML tag for creating hyperlinks to other pages. Each of these also has ending tags `</td>` and `</a>`. Fortunately for us they are all contained within the < and > characters. So let's use `gsub()` to remove all the HTMl tags.

```r
words <- gsub("<[^>]*>","",words)
```

This regular expression looks for a <, then a bunch of characters that are not a >, followed by a > and replaces them with nothing. Now `words` contains just our list of 9,999 most common words. Here are the first 50 of them.

```r
words[1:50]
```

```
 [1] "the"   "of"    "and"   "to"    "in"    "i"     "that"  "was"
 [9] "his"   "he"    "it"    "with"  "is"    "for"   "as"    "had"
[17] "you"   "not"   "be"    "her"   "on"    "at"    "by"    "which"
[25] "have"  "or"    "from"  "this"  "him"   "but"   "all"   "she"
[33] "they"  "were"  "my"    "are"   "me"    "one"   "their" "so"
[41] "an"    "said"  "them"  "we"    "who"   "would" "been"  "will"
[49] "no"    "when"
```

Exercises

19. We were told "i before e except after c, or when sounded like a as in neighbor or weigh". Is that true?
20. Find words with punctuation
21. Find words that do not have aeiou in the first four letters

Solutions to the exercises

1. Write a regular expression that will match these `c("A1","B1","C1")` but not these `c("D1","E1","F1")`.

```
grep("[ABC]", c("A1","B1","C1","D1","E1","F1"), value=TRUE)
```

```
[1] "A1" "B1" "C1"
```

2. Write a regular expression that will match these `c("1A","2B","3C")` but not these `c("A1","B2","C3")`.

```
grep("[123][ABC]", c("1A","2B","3C","A1","B2","C3"),value=TRUE)
```

```
[1] "1A" "2B" "3C"
```

or

```
grep("[0-9][A-Z]", c("1A","2B","3C","A1","B2","C3"),value=TRUE)
```

```
[1] "1A" "2B" "3C"
```

3. Write a regular expression that will match these `c("123","567","314")` but not these `c("1234","5678","3141")`.

```
grep("^[0-9]{3}$", c("123", "567", "314", "1234", "5678", "3141"), value=TRUE)
```

```
[1] "123" "567" "314"
```

4. Write a regular expression that will match these `c("123ABC","234BCDEF","435C")` but not these `c("1ABC23","2468BC","1234C5")`.

```
grep("^[0-9]{3}[A-Z]", c("123ABC","234BCDEF","435C", "1ABC23","2468BC","1234C5"),
     value = TRUE)
```

```
[1] "123ABC"   "234BCDEF" "435C"
```

5. Find commas

```
grep(",", dataText,value=TRUE)
```

```
[1] "Philadelphia, PA"
```

Remember that not every regular expression is complicated. Sometimes you just need to search for something specific and it requires no fancy regular expression.

6. Find ZIP codes

```
grep("^[0-9]{5}$|^[0-9]{5}-", dataText, value=TRUE)
```

```
[1] "19104-6286" "20015"      "90291"      "90210"
```

7. Find those with six letter word

```
grep("\\b[A-Za-z]{6}\\b", dataText, value=TRUE)
```

```
[1] "483 McNeil Building" "3718 Locust Walk"    "4 Privet Drive"
```

```
[4] "10880 Malibu Point"  "Philly"              "Phil Dunphy"
```

8. Find mentions of Philadelphia

```
grep("Phil[^ ]", dataText, value=TRUE)
```

```
[1] "Philadelphia, PA" "Philly"           "Phila"
[4] "Phil."
```

9. Find capitalized words

```
grep("\\b[A-Z]", dataText, value=TRUE)
```

```
[1] "483 McNeil Building" "3718 Locust Walk"    "4 Privet Drive"
[4] "10880 Malibu Point"  "Philadelphia, PA"    "Philly"
[7] "Phila"               "Phil."               "Phil Dunphy"
```

10. Find the addresses

```
grep("[0-9]+ [A-Za-z ]+ (Drive|Walk|Point)",dataText,value=TRUE)
```

```
[1] "3718 Locust Walk"   "4 Privet Drive"     "10880 Malibu Point"
```

11. Find a shorter way to get phone numbers using ?

```
grep("\\(?[0-9]{3}(\\) |-| )?[0-9]{3}(-| )?[0-9]{4}",dataText,value=TRUE)
```

```
[1] "(215) 573-9097" "215-573-9097"   "2155739097"
```

12. Remove the commas from `c("9,453","23,432","4,334,645","1,234")`

```
gsub(",","",c("9,453","23,432","4,334,645","1,234"))
```

```
[1] "9453"    "23432"   "4334645" "1234"
```

13. Add commas to these numbers `c("9453","2332","4645","1234")`.

```
gsub("([0-9])([0-9]{3})", "\\1,\\2", c("9453","2332","4645","1234"))
```

```
[1] "9,453" "2,332" "4,645" "1,234"
```

or

```
gsub("^([0-9])", "\\1,", c("9453","2332","4645","1234"))
```

```
[1] "9,453" "2,332" "4,645" "1,234"
```

14. Spell out PA as Pennsylvania

```
gsub("PA","Pennsylvania",dataText)
```

```
 [1] "suspect"                    "three suspects"
 [3] "3 suspects"                 "9mm firearm"
 [5] "483 McNeil Building"        "3718 Locust Walk"
 [7] "4 Privet Drive"             "10880 Malibu Point"
 [9] "Philadelphia, Pennsylvania" "Philly"
[11] "Phila"                      "Phil."
[13] "Phil Dunphy"                "19104-6286"
```

```
[15] "20015"                        "90291"
[17] "90210"                        "(215) 573-9097"
[19] "215-573-9097"                 "2155739097"
```

Again, remember that sometimes the regular expression is simple and requires nothing fancy.

15. Spell out Philadelphia

```
gsub(("Phil(adelphia|ly|\\.|a)"), "Philadelphia", dataText)
```

```
 [1] "suspect"             "three suspects"      "3 suspects"
 [4] "9mm firearm"         "483 McNeil Building" "3718 Locust Walk"
 [7] "4 Privet Drive"      "10880 Malibu Point"  "Philadelphia, PA"
[10] "Philadelphia"        "Philadelphia"        "Philadelphia"
[13] "Phil Dunphy"         "19104-6286"          "20015"
[16] "90291"               "90210"               "(215) 573-9097"
[19] "215-573-9097"        "2155739097"
```

or

```
gsub("Phil[^ ]+","Philadelphia",dataText)
```

```
 [1] "suspect"             "three suspects"      "3 suspects"
 [4] "9mm firearm"         "483 McNeil Building" "3718 Locust Walk"
 [7] "4 Privet Drive"      "10880 Malibu Point"  "Philadelphia PA"
[10] "Philadelphia"        "Philadelphia"        "Philadelphia"
[13] "Phil Dunphy"         "19104-6286"          "20015"
[16] "90291"               "90210"               "(215) 573-9097"
[19] "215-573-9097"        "2155739097"
```

16. Change Phil to Phillip (where appropriate)

```
gsub("Phil ","Phillip ",dataText)
```

```
 [1] "suspect"             "three suspects"      "3 suspects"
 [4] "9mm firearm"         "483 McNeil Building" "3718 Locust Walk"
 [7] "4 Privet Drive"      "10880 Malibu Point"  "Philadelphia, PA"
[10] "Philly"              "Phila"               "Phil."
[13] "Phillip Dunphy"      "19104-6286"          "20015"
[16] "90291"               "90210"               "(215) 573-9097"
[19] "215-573-9097"        "2155739097"
```

17. Keep just the first word or first number

```
gsub("^([^ ]+) .*$","\\1",dataText)
```

```
 [1] "suspect"       "three"         "3"             "9mm"
 [5] "483"           "3718"          "4"             "10880"
 [9] "Philadelphia," "Philly"        "Phila"         "Phil."
[13] "Phil"          "19104-6286"    "20015"         "90291"
[17] "90210"         "(215)"         "215-573-9097"  "2155739097"
```

or

```
gsub("^([^ ]*).*","\\1",dataText)
```

```
 [1] "suspect"       "three"         "3"             "9mm"
 [5] "483"           "3718"          "4"             "10880"
 [9] "Philadelphia," "Philly"        "Phila"         "Phil."
[13] "Phil"          "19104-6286"    "20015"         "90291"
[17] "90210"         "(215)"         "215-573-9097"  "2155739097"
```

or

```
gsub(" .*","",dataText)
```

```
 [1] "suspect"       "three"         "3"             "9mm"
 [5] "483"           "3718"          "4"             "10880"
 [9] "Philadelphia," "Philly"        "Phila"         "Phil."
[13] "Phil"          "19104-6286"    "20015"         "90291"
[17] "90210"         "(215)"         "215-573-9097"  "2155739097"
```

18. Keep only area codes of phone numbers

```
gsub("^\\(?([0-9]{3})(\\) |-| )?[0-9]{3}-?[0-9]{4}","\\1",dataText)
```

```
 [1] "suspect"              "three suspects"       "3 suspects"
 [4] "9mm firearm"          "483 McNeil Building" "3718 Locust Walk"
 [7] "4 Privet Drive"       "10880 Malibu Point"   "Philadelphia, PA"
[10] "Philly"               "Phila"                "Phil."
[13] "Phil Dunphy"          "19104-6286"           "20015"
[16] "90291"                "90210"                "215"
[19] "215"                  "215"
```

19. We were told "i before e except after c, or when sounded like a as in neighbor or weigh". Is that true?

```
grep("[^c]ei", words, value=TRUE)
grep("cie", words, value=TRUE)
```

```
 [1] "their"          "being"         "neither"       "seeing"
 [5] "foreign"        "weight"        "seized"        "height"
 [9] "reign"          "sovereign"     "beings"        "wherein"
[13] "veil"           "therein"       "leisure"       "seize"
[17] "neighborhood"   "heir"          "theirs"        "neighbors"
[21] "veins"          "heights"       "neighbour"     "neighbouring"
[25] "weighed"        "neighbor"      "reigned"       "sovereignty"
[29] "foreigners"     "neighboring"   "reins"         "seizing"
[33] "weigh"          "vein"          "Keith"         "sovereigns"
[37] "leisurely"      "foreigner"     "weird"         "deity"
[41] "weighing"       "freight"       "deities"       "rein"
[45] "reigns"         "weighty"       "Raleigh"       "albeit"
[49] "heightened"     "Seine"         "well-being"    "forfeit"
[53] "Marseilles"     "twenty-eight"  "herein"
 [1] "ancient"       "society"       "sufficient"    "species"
 [5] "science"       "conscience"    "sufficiently"  "scientific"
 [9] "fancied"       "fancies"       "efficient"     "efficiency"
[13] "tendencies"    "conscientious" "insufficient"  "deficient"
[17] "deficiency"
```

There are a lot of words with "ei" where the letter before the "ei"" is not a "c". Also, there are a lot of words that have "ie" immediately following a "c".

20. Find words with punctuation

```
grep("[^a-zA-Z0-9]", words, value=TRUE)
```

```
  [1] "I'll"             "can't"            "I've"             "didn't"
  [5] "an'"              "won't"            "man's"            "c."
  [9] "that's"           "etc."             "o'"               "I'd"
 [13] "v."               "father's"         "wouldn't"         "he's"
 [17] "couldn't"         "isn't"            "'em"              "king's"
 [21] "there's"          "ain't"            "you'll"           "god's"
 [25] "mother+'s"        "wasn't"           "doesn't"          "haven't"
 [29] "you've"           "woman's"          "we'll"            "she's"
 [33] "'tis"             "you'd"            "what's"           "cf."
 [37] "hadn't"           "th'"              "pp."              "rest."
 [41] "o'er"             "u.s."             "he'd"             "they're"
 [45] "we're"            "day's"            "girl's"           "he'll"
 [49] "shouldn't"        "husband's"        "twenty-five"      "men's"
 [53] "other's"          "i.e."             "brother's"        "wife's"
 [57] "we've"            "well-known"       "lady's"           "twenty-four"
 [61] "ma'am"            "enemy's"          "imp."             "hasn't"
 [65] "majesty's"        "viz."             "mustn't"          "old-fashioned"
 [69] "friend's"         "so-called"        "re-use"           "aren't"
 [73] "they'll"          "needn't"          "pub."             "middle-aged"
 [77] "she'd"            "t'"               "of."              "oe."
 [81] "prof."            "she'll"           "shan't"           "ne'er"
 [85] "as."              "they'd"           "we'd"             "here]'s"
 [89] "they've"          "adj."             "weren't"          "sq."
 [93] "gen."             "cong."            "col."             "brother-in-law"
 [97] "son-in-law"       "e.g."             "ch."              "e-mail"
[101] "twenty-one"       "twenty-two"       "it'll"            "ft."
[105] "self-control"     "anglo-saxon"      "ff."              "forty-five"
[109] "'ll"              "Icel."            "good-looking"     "p.m."
[113] "e'er"             "twenty-three"     "thirty-six"       "a.m."
[117] "father-in-law"    "e'en"             "seventy-five"     "'cause"
[121] "well-being"       "Mass."            "thirty-two"       "self-"
[125] "twenty-eight"     "twenty-six"       "sister-in-law"    "three-quarters"
[129] "'n'"
```

or

```
grep("[[:punct:]]", words, value=TRUE)
```

```
  [1] "I'll"             "can't"            "I've"             "didn't"
  [5] "an'"             "won't"           "man's"           "c."
  [9] "that's"          "etc."            "o'"              "I'd"
 [13] "v."              "father's"        "wouldn't"        "he's"
 [17] "couldn't"        "isn't"           "'em"             "king's"
 [21] "there's"         "ain't"           "you'll"          "god's"
 [25] "mother+'s"       "wasn't"          "doesn't"         "haven't"
 [29] "you've"          "woman's"         "we'll"           "she's"
 [33] "'tis"            "you'd"           "what's"          "cf."
 [37] "hadn't"          "th'"             "pp."             "rest."
 [41] "o'er"            "u.s."            "he'd"            "they're"
 [45] "we're"           "day's"           "girl's"          "he'll"
 [49] "shouldn't"       "husband's"       "twenty-five"     "men's"
 [53] "other's"         "i.e."            "brother's"       "wife's"
 [57] "we've"           "well-known"      "lady's"          "twenty-four"
 [61] "ma'am"           "enemy's"         "imp."            "hasn't"
 [65] "majesty's"       "viz."            "mustn't"         "old-fashioned"
 [69] "friend's"        "so-called"       "re-use"          "aren't"
 [73] "they'll"         "needn't"         "pub."            "middle-aged"
 [77] "she'd"           "t'"              "of."             "oe."
 [81] "prof."           "she'll"          "shan't"          "ne'er"
 [85] "as."             "they'd"          "we'd"            "here]'s"
 [89] "they've"         "adj."            "weren't"         "sq."
 [93] "gen."            "cong."           "col."            "brother-in-law"
 [97] "son-in-law"      "e.g."            "ch."             "e-mail"
[101] "twenty-one"      "twenty-two"      "it'll"           "ft."
[105] "self-control"    "anglo-saxon"     "ff."             "forty-five"
[109] "'ll"             "Icel."           "good-looking"    "p.m."
[113] "e'er"            "twenty-three"    "thirty-six"      "a.m."
[117] "father-in-law"   "e'en"            "seventy-five"    "'cause"
[121] "well-being"      "Mass."           "thirty-two"      "self-"
[125] "twenty-eight"    "twenty-six"      "sister-in-law"   "three-quarters"
[129] "'n'"
```

[:punct:] is a special set containing all punctuation characters. or

```
grep("\\W", words, value=TRUE)
```

```
  [1] "I'll"            "can't"           "I've"            "didn't"
  [5] "an'"             "won't"           "man's"           "c."
  [9] "that's"          "etc."            "o'"              "I'd"
 [13] "v."              "father's"        "wouldn't"        "he's"
 [17] "couldn't"        "isn't"           "'em"             "king's"
 [21] "there's"         "ain't"           "you'll"          "god's"
 [25] "mother+'s"       "wasn't"          "doesn't"         "haven't"
 [29] "you've"          "woman's"         "we'll"           "she's"
 [33] "'tis"            "you'd"           "what's"          "cf."
 [37] "hadn't"          "th'"             "pp."             "rest."
 [41] "o'er"            "u.s."            "he'd"            "they're"
 [45] "we're"           "day's"           "girl's"          "he'll"
 [49] "shouldn't"      "husband's"      "twenty-five"    "men's"
 [53] "other's"        "i.e."           "brother's"      "wife's"
 [57] "we've"          "well-known"     "lady's"         "twenty-four"
 [61] "ma'am"          "enemy's"        "imp."           "hasn't"
 [65] "majesty's"      "viz."           "mustn't"        "old-fashioned"
 [69] "friend's"       "so-called"      "re-use"         "aren't"
 [73] "they'll"        "needn't"        "pub."           "middle-aged"
 [77] "she'd"          "t'"             "of."            "oe."
 [81] "prof."          "she'll"         "shan't"         "ne'er"
 [85] "as."            "they'd"         "we'd"           "here]'s"
 [89] "they've"        "adj."           "weren't"        "sq."
 [93] "gen."           "cong."          "col."           "brother-in-law"
 [97] "son-in-law"     "e.g."           "ch."            "e-mail"
[101] "twenty-one"     "twenty-two"     "it'll"          "ft."
[105] "self-control"   "anglo-saxon"    "ff."            "forty-five"
[109] "'ll"            "Icel."          "good-looking"   "p.m."
[113] "e'er"           "twenty-three"   "thirty-six"     "a.m."
[117] "father-in-law"  "e'en"           "seventy-five"   "'cause"
[121] "well-being"     "Mass."          "thirty-two"     "self-"
[125] "twenty-eight"   "twenty-six"     "sister-in-law"  "three-quarters"
[129] "'n'"
```

`\W` matches any character that is not a number or a letter.

21. Find words that do not have aeiou in the first four letters

```
grep("^[^aeiouAEIOU]{4}", words, value=TRUE)
```

```
 [1] "system"        "physical"      "style"         "sympathy"
 [5] "mysterious"    "mystery"       "physician"     "thyself"
 [9] "sympathetic"   "mysteries"     "systems"       "Dryden"
[13] "symptoms"      "symbol"        "crystal"       "10th"
[17] "hymn"          "mystic"        "sympathies"    "Sydney"
[21] "12th"          "syllable"      "11th"          "rhyme"
[25] "symbols"       "physically"    "rhythm"        "systematic"
[29] "psychology"    "psychological" "p.m."          "mystical"
[33] "mythology"     "myth"          "syllables"     "hysterical"
```