Street or Road?

address
PO BOX 74
16 COVEWOOD ROAD
PO BOX 573
103 DANBURY ST SW
PO BOX 933
NCSU BOX 22425
PO BOX 37322
PO BOX 1125
PO BOX 2362
NCSU BOX 15644
NCSU BOX 03349
209 ODUM AVE
P O BOX 43
PO BOX 4202
NCSU BOX 15102
... and so on for 15,483 rows

People's addresses involve streets, lanes, courts, avenues, and so on. How many such road-related words are in common use?

In answering this question, you would presumably want to look at lots of addresses and extract the road-related term. You could do this by eye, reading down a list of a few hundred or thousand addresses. But if you want to do it on a really large scale, a city or state or country, you would want some automated help, for instance, a computer program that discards the sorts of entries you have already identified to give a greater concentration of unidentified terms. In this activity, you're going to build such a program.

Some resources:

1. The file `http://tiny.cc/dcf/street-addresses.csv` contains about 15000 street addresses of registered voters in Wake County, North Carolina.

2. The file "`http://tiny.cc/dcf/CMS_ProvidersSimple.rds`" has street address of about 900,000 medicare service providers. Download the file to save it on your own system, then read it in under a convenient name.

```
download.file(url="http://tiny.cc/dcf/CMS_ProvidersSimple.rds",
              destfile = "YourNameForTheFile.rds")
DataTable <- readRDS("YourNameForTheFile.rds")
```

To solve such problems, start by looking at a few dozen of the addresses to familiarize yourself with common patterns. With those few dozen

1. In everyday language, describe a pattern that you think will identify the information you are looking for.

2. Translate (1) into the form of a regular expression.

3. Filter to retain the cases that match the expression. Hint: `filter()` and `grepl()` are useful for this.

4. Filter to retain the cases that do not match the expression.
5. Examine the results of (2) and (3) to identify shortcomings in your patterns.
6. Improve or extend the pattern to deal with the mistaken cases.
7. Repeat until satisfied.
8. Put extraction parentheses around the parts of the regular expression that contain the info you want.

Solved Example:

Suppose you wanted to extract the PO Box number from an address.

Read the street address data and pull out a sample of a few dozen cases.

```
Addresses <- read.file("http://tiny.cc/dcf/street-addresses.csv")
Sample <- Addresses %>%
  sample_n(size = 50)
```

Following each of the steps listed above:

1. The PO Box cases tend to have a substring "PO".
2. The regular expression for "PO" is simply `"PO"`.
3. Find some cases that match:

```
Matches <-
  Sample %>%
  filter(grepl("PO", address))
```

4. Find cases that don't match:

```
Dont <-
  Sample %>%
  filter( ! grepl("PO", address))
```

5. Find any cases in the `Matches` that shouldn't be there. (None are seen in this example.) Find any cases in `Dont` that should have matched. (There are several, three of which appear in the table to the right.)

address
PO BOX 74
PO BOX 573
PO BOX 933
PO BOX 37322
PO BOX 1125
... and so on for 22 rows

address
16 COVEWOOD ROAD
103 DANBURY ST SW
NCSU BOX 22425
NCSU BOX 15644
NCSU BOX 03349
... and so on for 28 rows

6. It looks like "BOX" is a better pattern. Since the box number is wanted, the regex should include an identifier for the number inside extraction parentheses. So `"BOX\\s+(\\d+)"`.

Note the double slashes, `\\`, in `\\s` and `\\d`. Ordinarily, `\` is a special character in R character strings used to designate special characters like new-line `\n` or tab `\t`. The double `\\` means, "just an ordinary slash, please." Confusing. But whenever characters are used to signal something special, you have to take an extra step to say that you don't want the special meaning.

```
pattern <- "BOX\\s+(\\d+)"
Matches <-
  Sample %>%
  filter(grepl(pattern, address))
Dont <-
  Sample %>%
  filter( ! grepl(pattern, address))
```

The result seems satisfactory.

So, use `tidyr::extract()` to pull out the part of the pattern identified by extraction parentheses.

address
16 COVEWOOD ROAD
103 DANBURY ST SW
209 ODUM AVE
6221-104 ST REGIS CIRCLE
233 RIVERBEND RD
... and so on for 16 rows

```
BoxNumbers <-
  Sample %>%
  filter(grepl(pattern, address)) %>%
  tidyr::extract(address, into="boxnum", regex=pattern)
```

boxnum
74
573
933
22425
37322
... and so on for 34 rows

Note that `tidyr::extract()` should be given only those cases that match the regular expression, so `filter()` is applied before `tidyr::extract()`

Back to the Streets

Street endings (e.g. "ST", "LANE") are often found at the end of the address string. Use this as a starting point to find the most common endings.

Once you have a set of specific street endings, you can use the regex "or" symbol, e.g. `"(ST|RD|ROAD)"`. The parentheses are not incidental. They are there to mark a pattern that you want to extract. In this case, in addition to knowing that there is a `ST` or `RD` or `ROAD` in an address, you want to know which one of those possibilities it is so that you can count the occurance of each of the possibilities.

To find street endings that aren't in your set, you can filter out the street endings or non-street addresses you already know about.

Your turn: Read the following R statements. Next to each line, give a short explanation of what the line contributes to the task. For each of the regexes, explain in simple everyday language what pattern is being matched.

```
pattern <- "(ST|RD|ROAD)"
LeftOvers <-
  Addresses %>%
  filter( ! grepl(pattern, address),
          ! grepl("\\sAPT|UNIT\\s[\\d]+$", address),
          ! grepl(" BOX ", address)
  )
```

address
2117 MARINER CIRCLE
101 EPPING WAY
04-I ROBIN CIRCLE
NCSU BoX 15637
4719 BROWN TRAIL
... and so on for 2,411 rows

For each set of patterns that you identify, compute the `LeftOvers`. Examine them visually to find new street endings to add to the pattern, e.g. `LANE`.

When you have this working on the small sample, use a larger sample and, eventually, the whole data set. It's practically impossible to find a method that will work perfectly on new data, but do the best you can.

Your turn: In your report, implement your method and explain how it works, line by line. Present your result: how many addresses there are of each kind of road word.

For the professional ...

Breaking addresses into their components is a common task. People who work on this problem intensively sometimes publish their regular expressions. Here's one from Ross Hammer published at `http://regexlib.com/Search.aspx?k=street`

```
^\s*((?:(?:\d+(?:\x20+\w+\.?)+(?:(?:\x20+STREET|ST|DRIVE|DR|AVENUE|AVE|ROAD|RD|LOOP|COURT
|CT|CIRCLE|LANE|LN|BOULEVARD|BLVD)\.?)?)|(?:(?:P\.\x20?O\.|P\x20?O)\x20*Box\x20+\d+)|
(?:General\x20+Delivery)|(?:C[\\\/]O\x20+(?:\w+\x20*)+))\,?\x20*(?:(?:(?:APT|BLDG|DEPT|
FL|HNGR|LOT|PIER|RM|S(?:LIP|PC|T(?:E|OP))|TRLR|UNIT|\x23)\.?\x20*(?:[a-zA-Z0-9\-]+))|
(?:BSMT|FRNT|LBBY|LOWR|OFC|PH|REAR|SIDE|UPPR))?)\,?\s+((?:(?:\d+(?:\x20+\w+\.?)+
(?:(?:\x20+STREET|ST|DRIVE|DR|AVENUE|AVE|ROAD|RD|LOOP|COURT|CT|CIRCLE|LANE|LN|BOULEVARD|
BLVD)\.?)?)|(?:(?:P\.\x20?O\.|P\x20?O)\x20*Box\x20+\d+)|(?:General\x20+Delivery)|
(?:C[\\\/]O\x20+(?:\w+\x20*)+))\,?\x20*(?:(?:(?:APT|BLDG|DEPT|FL|HNGR|LOT|PIER|RM|
S(?:LIP|PC|T(?:E|OP))|TRLR|UNIT|\x23)\.?\x20*(?:[a-zA-Z0-9\-]+))|(?:BSMT|FRNT|LBBY|
LOWR|OFC|PH|REAR|SIDE|UPPR))?)?\,?\s+((?:[A-Za-z]+\x20*)+)\,\s+(A[LKSZRAP]|C[AOT]|
D[EC]|F[LM]|G[AU]|HI|I[ADLN]|K[SY]|LA|M[ADEHINOPST]|N[CDEHJMVY]|O[HKR]|P[ARW]|RI|
S[CD]|T[NX]|UT|V[AIT]|W[AIVY])\s+(\d+(?:-\d+)?)\s*$
```

Scraping Nuclear Reactors

In this project,[2] you're going to look at data about nuclear reactors. Let's use Japan as an example. Often, when you are doing a quick project, sources like Wikipedia are useful.

Go to the page `http::://en.wikipedia.org/wiki/List_of_nuclear_reactors"`. Find the reactor list for Japan. Figure A.24 shows part of the list[3] as a cut-and-paste image from a web browser.

[2] Devised initially by Prof. Nicholas Horton, Amherst College

[3] on March 23, 2015

Name	Reactor	Reactor		Status	Capacity in MW		Construction Start Date	Commercial Operation Date	Closure
		Type	Model		Net	Gross			
Fukushima Daiichi	1	BWR	BWR-3	Shutdown	439	460	25 Jul, 1967	26 Mar, 1971	19 May 2011
Fukushima Daiichi	2	BWR	BWR-4	Shutdown	760	784	09 Jun, 1969	18 Jul, 1974	19 May 2011
Fukushima Daiichi	3	BWR	BWR-4	Shutdown	760	784	28 Dec, 1970	27 Mar, 1976	19 May 2011
Fukushima Daiichi	4	BWR	BWR-4	Shutdown	760	784	12 Feb, 1973	12 Oct, 1978	19 May 2011

Figure A.24: Part of the Wikipedia table describing nuclear reactors in Japan.

Unfortunately, it is not a matter of cut-and-paste to get the tables in Wikipedia into the form of a data table in R. The tables often have a complex, non-tidy form. In addition, the tables are written using HTML tags, which can have be confusing. For instance, here a bit of the HTML behind the table of reactors in Japan.

```
<table class="wikitable sortable">
<tr>
<th rowspan="2" style="background:#FFDEAD;">Name</th>
<th rowspan="2" style="background:#FFDEAD;">Reactor</th>
<th colspan="2" style="background:#FFDEAD;">Reactor</th>
<th colspan="2" style="background:#FFDEAD;">Capacity in MW</th>
...
</tr>
<tr>
<td>Fukushima Daiichi</td>
<td>1</td><td>BWR</td><td>BWR-3</td><td>Shutdown</td>
```