Data cleaning with Regular Expressions

By Christian McDonald. Updated March 2016 for use in:

- Spring 2016 Data Visualization, UT-Austin School of Journalism
- CAR 2015 conference session (NICAR)

Regular expressions (or regex) is a programming language used to match patterns in text, so it is VERY useful in cleaning data. This lesson is just an introduction to a very powerful programming skill.

Quick start

- Launch Sublime Text
- Download this text: socrata_addresses.txt
- Go to regex101.com
- We'll be using this Regex Cheat Sheet (or this alternative). Download a copy for yourself.

Key concept: patterns

(This intro also handled in: Regular expression patterns - Intro Lecture). If you did you saw the slides, then skip to here.

Let's say you have a list of phone numbers in 10-digit format -- `512-555-1212` -- but you want the area code to be in parenthesis: `(512) 555-1212`. You could do a simple search of the 512 area code followed by a dash `512-` and replace it with an area code in parenthesis and a space: `(512) `.

But what if there are different area codes in the list?

```
512-555-1211
301-333-1212
404-123-1213
```

With regular expressions, you can search for a pattern of characters and save it as a group. Instead of searching just for `512-`, you can look for "three numbers together at the beginning of a line that are then followed by a dash".

If you save that matching pattern as a group, you can then replace that group with parenthesis outside it, no matter what the contents of the group. If that saved group is called `\1` then you can replace it with `(\1)` and it doesn't matter if `\1` is equal to `512-` or `301-`.

Special characters, commands and escape

Let's touch quickly on the syntax of regular expressions. Don't get hung up if these sounds like gibberish, because it will make sense more when we start using it. Your Regex Cheat Sheet (or this alternative) comes in handy here.

Regular expressions use special characters to do special things, like match the beginning of a line. These commands are called tokens:

- `^` will find the beginning of a line.
- `*` will find "zero or more" of whatever precedes it.

Regular expressions use the backslash (the one above return on the keyboard: `\`) with other characters to create more tokens to do special things:

- `\d` will find any number character (or digit).
- `\D` will match anything other than a number.
- `\t` is a tab character, because hitting the tab on the keyboard will perform the action instead of giving you the character.

But then sometimes, you actually need to find the character `^`, and not use it as a command. Regular expressions use the `\` to give the literal expression of a character that would otherwise be a token:

- `\*` will find the asterisk character instead of modifying the query to find "zero or more".

Enough of that. Let's do this, with the help of ...

Regex101.com

Regex101 is a great way to not only build regex patterns, but to also learn how they work. Go ahead and launch that site in a browser so we can work with it here in a minute.

We're going to use this site to split some address data into their parts. Let's talk about the data first.

Splitting Socrata addresses

Many government agencies use Socrata as their open data portal: From Austin to Boston; from Los Angeles to New York.

Some of the data sets in Austin (and I'm sure other cities) have all the address parts crammed into a single field. Regex can easily explode that into individual columns.

Our goal is to turn this:

Address
303 W 15TH ST AUSTIN, TX 78701 (30.2775019625634, -97.7425547430518)
13729 N US 183 HWY AUSTIN, TX 78729 (30.4589810894458, -97.7920758312874)
1410 S 1ST ST AUSTIN, TX 78704 (30.2503588573541, -97.7549624350228)

into this:

Address	City	State	ZIP	Latitude	Longitude
303 W 15TH ST	AUSTIN	TX	78701	30.27750196	-97.74255474
13729 N US 183 HWY	AUSTIN	TX	78729	30.45898109	-97.79207583
1410 S 1ST ST	AUSTIN	TX	78704	30.25035886	-97.75496244

Let's get started:

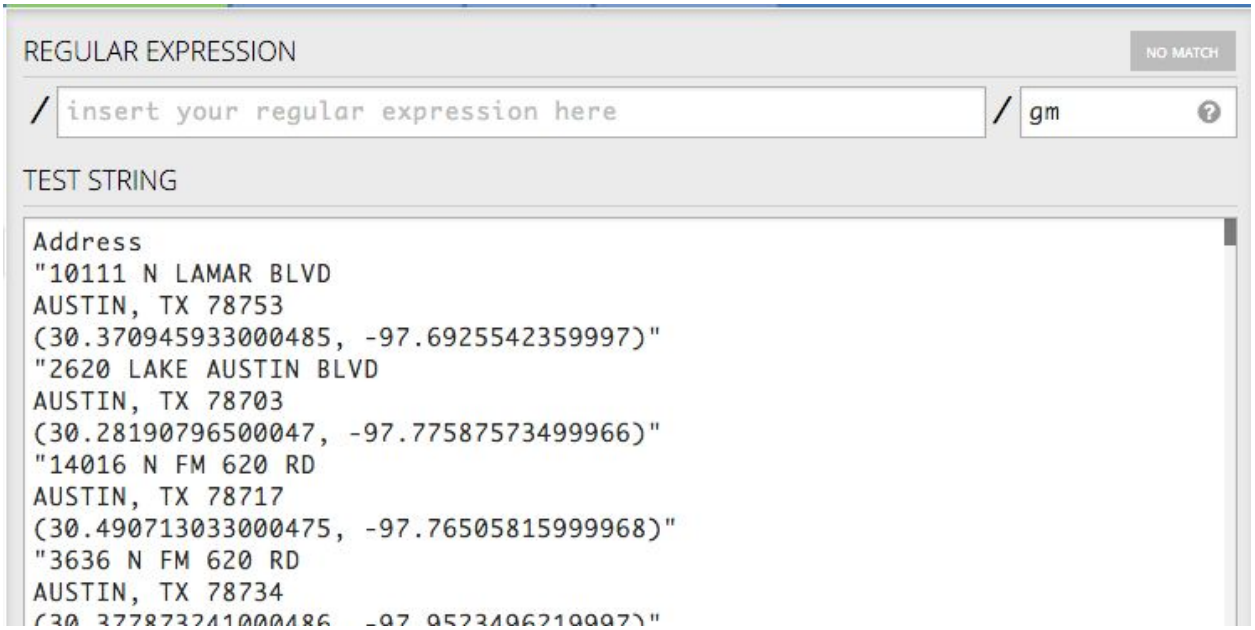
- Download the file socrata_addresses.txt and open it in your text editor. For NICAR, we'll use Sublime Text, which works on both Macs and PCs. (Atom will work with a caveat explained below. You can also use TextWrangler on Mac, or Notepad++ for PCs.)
- This is just one column from a larger Socrata data set of restaurant inspection scores in Austin, TX. (When I want to clean a single column of data, I often will just copy out one column into my text editor and work it before pasting back the results, carefully making sure they still line up.)
- Let's look at our data a little closer:

```
"10111 N LAMAR BLVD
AUSTIN, TX 78753
(30.370945933000485, -97.6925542359997)"
"2620 LAKE AUSTIN BLVD
AUSTIN, TX 78703
(30.28190796500047, -97.77587573499966)"
```

    - Notice the address, city, state, zip, latitude and longitude are all in the same "cell" (what is inside the quote marks), but the content of the cell has returns in it. We want to split these six distinct pieces into their own columns for each record. Why? Many reasons, but one is to use the latitude and longitude for data visualizations.
    - We will build a Regular Expression pattern to capture six groups and then search and replace to put tabs between each group, so we can put it back into Excel.
- Copy the contents of our test data from your text editor (or socrata_addresses.txt) into your clipboard and then go to http://regex101.com/ and paste the contents into the “TEST STRING” field.
- In the right-hand box of the REGULAR EXPRESSION string, add the modifier "gm". [1]

[1] On the modifier "gm": The "g" is for "global", as in find all occurrences, not just the first one. The "m" is for "multi-line", which allows us evaluate each line separately. I almost always use "gm" when doing search and replace like this.

- On the left-hand pane, make sure the FLAVOR is set on "pcre (php)". [2]



The goal in a nutshell

We are building a pattern in our regular expression field, creating a group to capture each part of the address that we want to keep. We'll continue the pattern outside the group until we get to the next part we want to keep, when we'll create a new group.

You'll want to reference your Regex Cheat Sheet (or this alternative).

Capturing the address

- We know that the first line starts at the beginning of the line, so we can start with this token, which signifies that: `^`
- Now, in the end, we don't want to keep the double-quote, so we won't put it inside a group. We'll just add it to the pattern: `^"`.
- Take note for a minute at the number of matches at the top right: There are 325 matches, and that is how many records we should end up with. You'll want to keep referring back to that and making sure you have 325 groups.

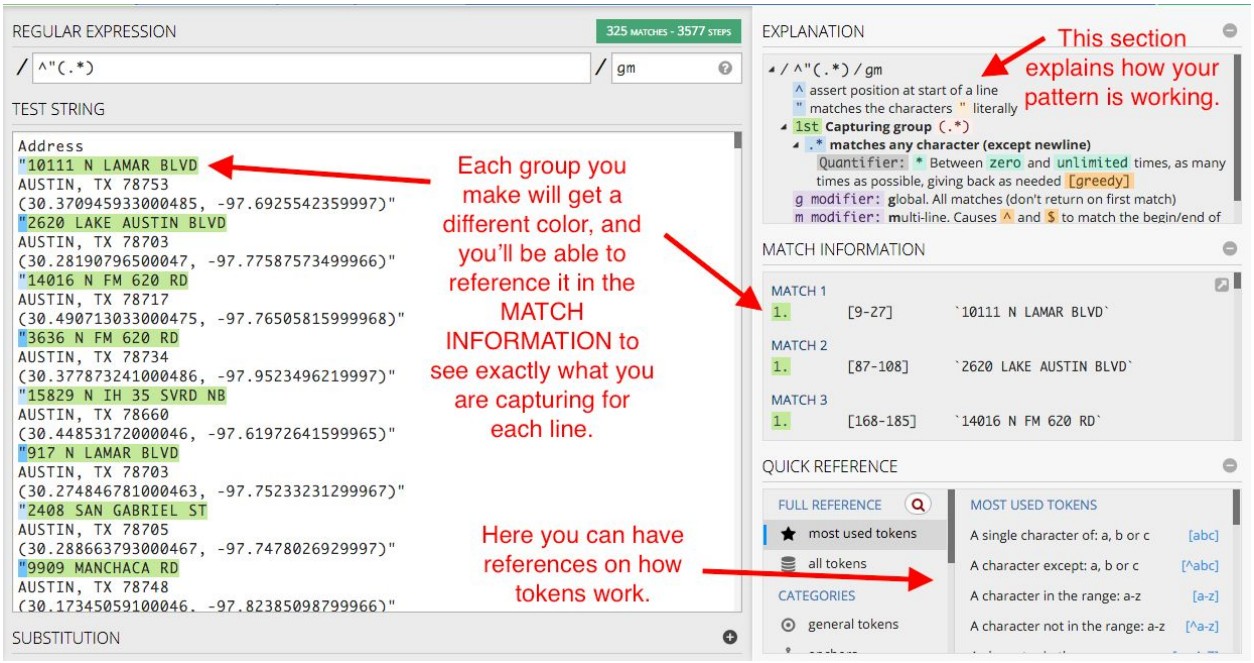
- Next, we'll start our first group with parenthesis: `^"()`.

---

[2] Different programming languages have little differences in how they handle Regular Expressions. The concepts are the same, but sometimes the syntax is different. TextWrangler and Sublime use this "pcre" flavor, so we'll use that. If you are using the Atom text editor, you should use the "javascript" flavor.

- In this first group, we want the whole address, which is everything on this line, until the line ending. The period token `.` means "any character", and `*` means "zero or more", so put these together and we get everything: `^"(.*)`.

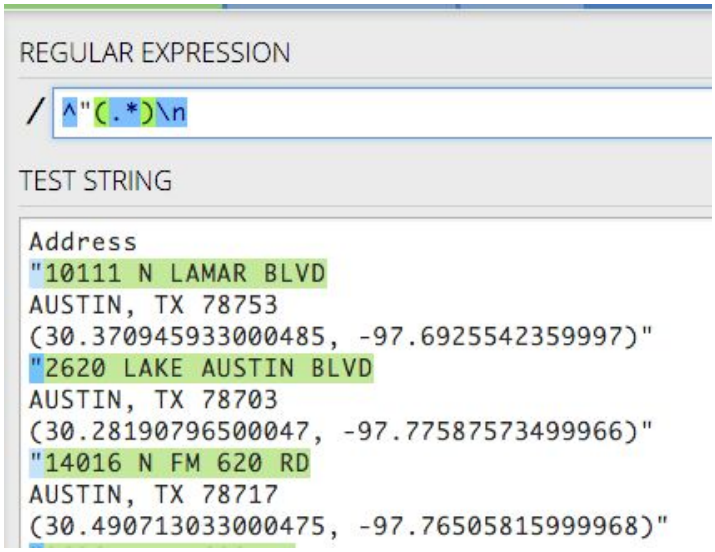
Let's take a minute to explain more about Regex101 and how it helps you.



Each matching group gets a color, and the contents of the match is shown in MATCH INFORMATION. The EXPLANATION section tells you exactly how each token is used, and the QUICK REFERENCE section is a list of tokens you can use, more in depth than the cheat sheet I've started you with.

OK .. on with it.

- Before we can start capturing the second group with the city, we need to add the return to the pattern so it can start recognizing the next line. But here's the drive-you-crazy thing: Windows and Macs treat these differently. On a Mac, `\n` is a "new line". If you are on a Windows machine, you'll need to use `\R`. The regex101 editor will recognize either, but later when we do the search and replace in your text editor, you have to use the character that works with your operating system. This tutorial will use Macs, so: `^"(.*)\n`.
- (If we end up with PC's in this lab, you'll need to use `\R` every time you see `\n` in this tutorial.)
- Sanity check: This what your screen should look like:



Capturing the city

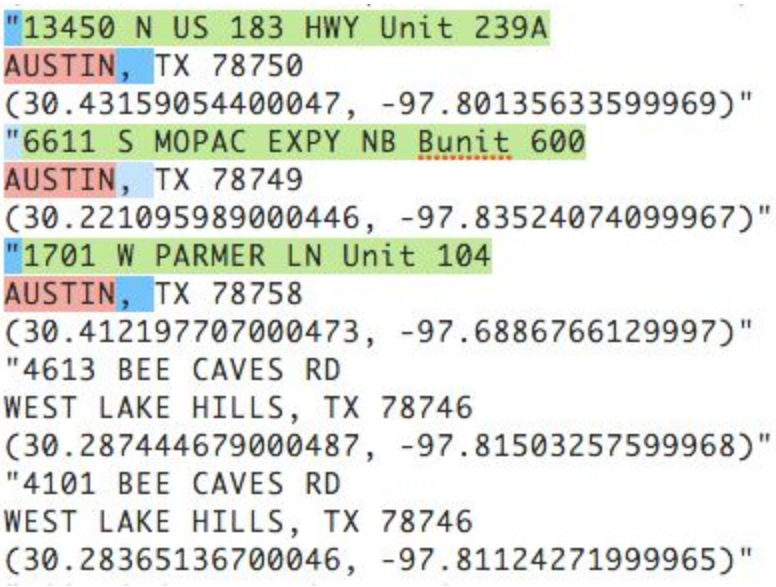
- Now, let's grab the city. As we look through the list, we can see there is more than Austin, and some of these names have spaces so we can't find just letters alone. There are MANY ways to do this, but we'll do it here by creating a group first: `^"(.*)\n()`.
- Then inside the group we'll put a token that looks for word characters: `^"(.*)\n(\w)`.
- Then we'll add the quantifier `+` to find "one or more" of what's in the character class: `^"(.*)\n(\w+)`.
- Make sure we are capturing all 325 groups. We should be good.

Capturing the state

- These are all the same, all in TX, so we don't really have to save it at all, but we will. We'll use this to remind ourselves that you can also just match a string literally. First we put the comma and space outside the second group, since we don't want to keep it: `^"(.*)\n(\w+), `.

Catching errors

- Now, take a moment and check how many matching groups we have. Wait ... WHOA ... we have only 314. What could've gone wrong?
- Scroll down the Test string until you find something amiss. The colors help you spot problems easily.

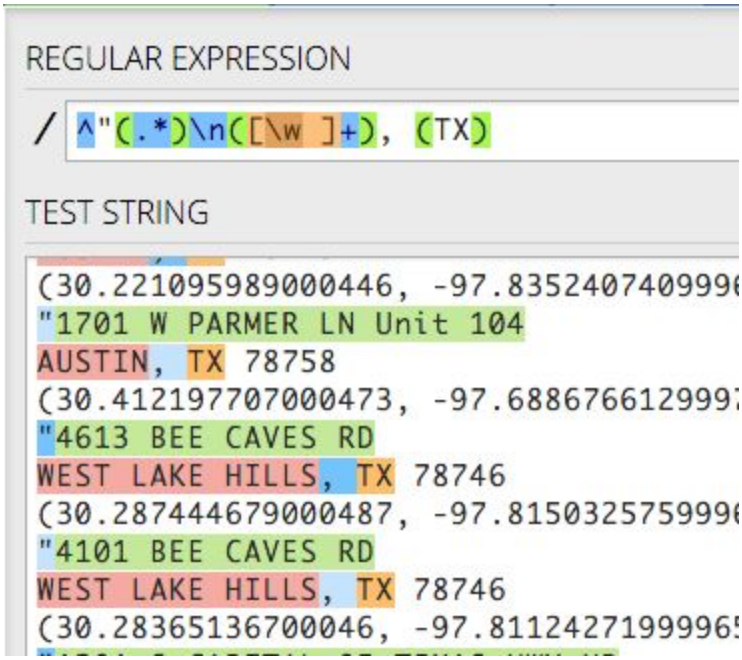


- What is the difference between the working lines and the ones that aren't? Something about the city.
- WEST LAKE HILLS has spaces while AUSTIN and PFLUGERVILLE do not. It looks like our city group did not capture a needed space, which wasn't really revealed until we tried to carry out the pattern with the `,` that is found after the city. This happens … we need to back up and fix the error.
- Right now, our expression is this: `^"(.*)\n(\w+),` and the part that captures the city is `(\w+)`. We need use use something called a "character class" that allows us to use more than one token within a group. We signify this by putting what we want inside of square brackets: So we need to put the `\w` inside square brackets along with the space so we can catch both: `^"(.*)\n([\w ]+),` .

Back to the state

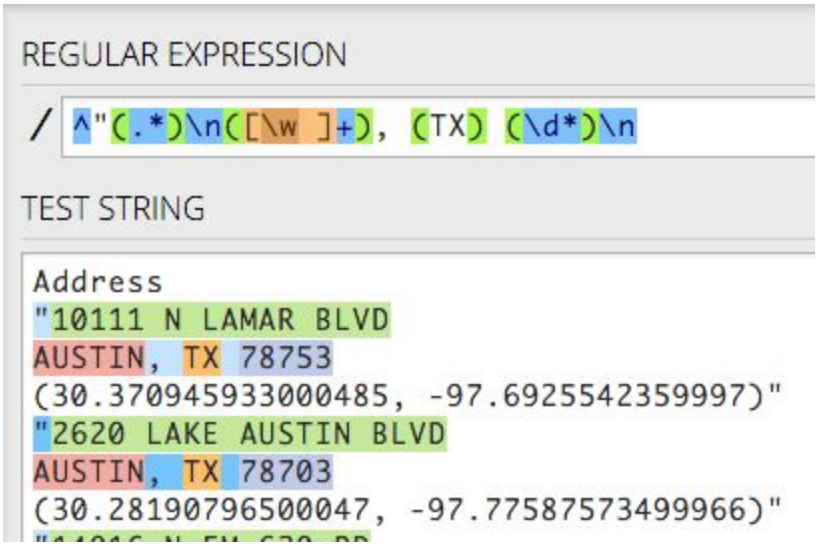
- Then we create our third group with TX inside it: `^"(.*)\n([\w ]+), (TX)`. We are looking for the literal text TX because there are no other states in this data set. We can't skip it because we need the pattern to continue.

Sanity check. Here is what you should have:



Capturing the ZIP

- Again, we don't want to keep the space between the state and ZIP, so we'll put it outside the third group, and start our fourth one for ZIP: `^"(.*)\n([\w ]+), (TX) ()`.
- All of these zip codes are of the 5-digit variety, so this can be less complicated than it would with the 9-digit version. Again, many ways to do this, but we'll use `\d` for the numbers and `*` to to capture zero or more of them: `^"(.*)\n([\w ]+), (TX) (\d*)`.
- Complete the pattern for this line with the new line token: `^"(.*)\n([\w ]+), (TX) (\d*)\n`.
- Sanity check: This is where we are …

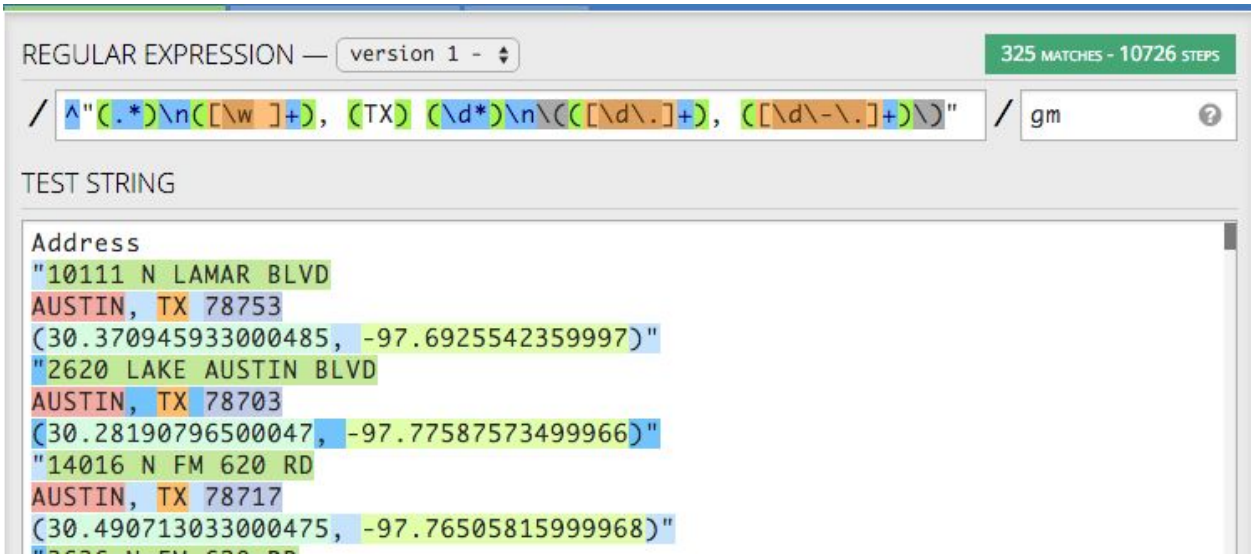


Capturing latitude

- We don't want to keep the parenthesis that starts this last line, so we'll put it outside a group. However, since parenthesis mean something special in regex, we need to escape it with a backslash so it will find the character and not start the new group: `^"(.*)\n([\w ]+), (TX) (\d*)\n\(`.
- Now we can start our new group, so go ahead and add the beginning and end parentheses: `^"(.*)\n([\w ]+), (TX) (\d*)\n\(()`.
- Inside our fifth group, we need numbers and the decimal point. We can create a character class and put inside it `\d` for numbers and `\.` for the decimal point, which we have to escape since . means "any character". We finish it off by looking for one or more. `^"(.*)\n([\w ]+), (TX) (\d*)\n\(([\d\.]+)`.

Capturing longitude

- We don't need the comma and space in our next group, so we put it outside to keep the pattern going: `^"(.*)\n([\w ]+), (TX) (\d*)\n\(([\d\.]+),` .
- We can get the longitude like we did latitude, but we have to add the hyphen to the character class. So, create the group: `^"(.*)\n([\w ]+), (TX) (\d*)\n\(([\d\.]+), ()`.
- Add the character class: `^"(.*)\n([\w ]+), (TX) (\d*)\n\(([\d\.]+), ([])`.
- And inside of it, put `\d` for digits, `\-` for the hyphen and `\.` for the decimal point: `^"(.*)\n([\w ]+), (TX) (\d*)\n\(([\d\.]+), ([\d\-\.])`.
- Add our quantifier to get one or more: `^"(.*)\n([\w ]+), (TX) (\d*)\n\(([\d\.]+), ([\d\-\.]+)`.
- Because the trailing parentheses and quote are at the end of a line, we could ignore them, but we won't. We'll add them to the end of the pattern, escaping the close parentheses just to be sure: `^"(.*)\n([\w ]+), (TX) (\d*)\n\(([\d\.]+), ([\d\-\.]+)\)"`.



You have it all! 325 matches into 6 different groups.

The substitution string

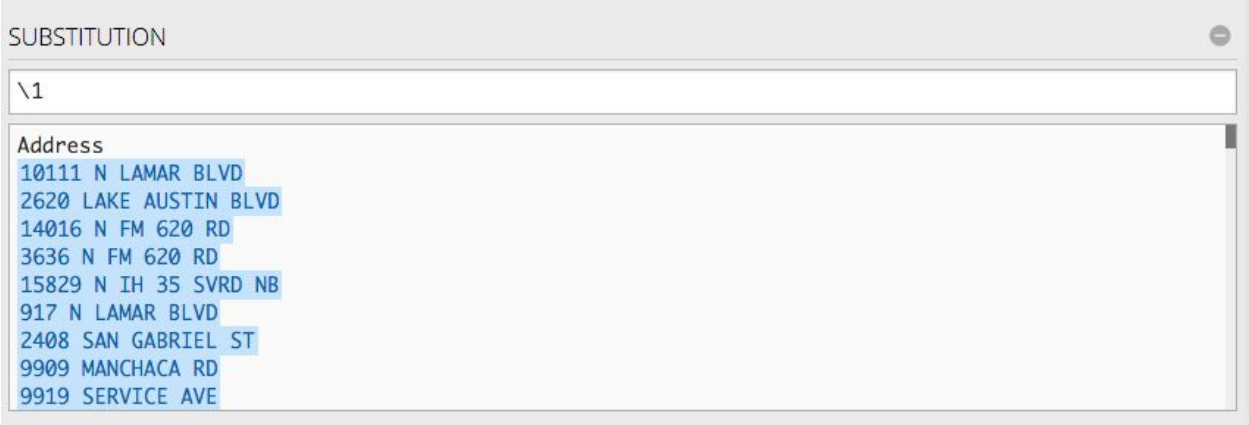
Now that we have a pattern with our six groups of data, we can substitute them in any order want using a search and replace, and we can build our substitution string right here in Regex101 as well.

Underneath our TEST STRING window, you'll see another header called SUBSTITUTION with a big plus sign on the right side of the window. Click the plus to expand the substitution window.

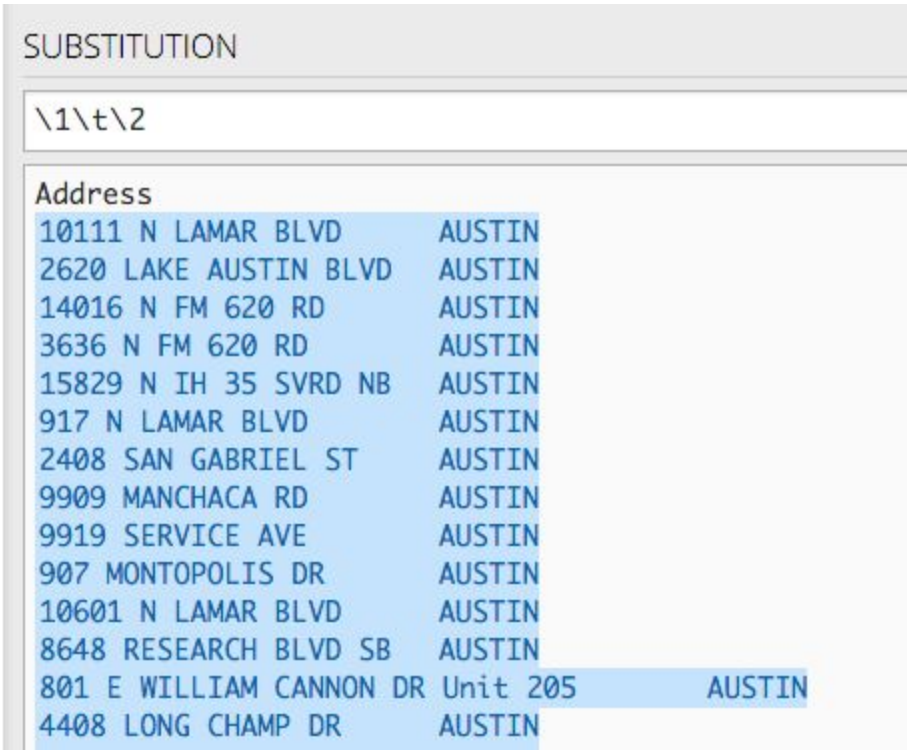
Now we can build a substitution string, and we can see the cleaned data in the window below.

Our goal with the substitution string is to pull back our six groups, but to put tabs in between each of them. If we can build a search and replace like this, then we can paste the result back into Excel, and each group will become its own column.

A quick refresher from our intro: Once we've built a group, we can reference it in our substitution string by its order using a backslash before it's order number. So, if we want to reference our first group, we use this: `\1`. Put that in the SUBSTITUTION string box, like this:



- You can see that Regex101 is now pulling back our address.
- We can't just type a tab key after our group because they keyboard command will move us to another box, so we use the token for tab, which is `\t`. So, add that to the end of our substitution string to get this: `\1\t`. You'll see space get added into our substiution example.
- Now we can add our next ordered group to our substitution string and see our city get added on: `\1\t\2`. Here is a sanity check:



- Now that you see how it works, let's go ahead and add the rest of the groups, all with tabs in between them: `\1\t\2\t\3\t\4\t\5\t\6`.

SUBSTITUTION

```
\1\t\2\t\3\t\4\t\5\t\6
```

```
Address
10111 N LAMAR BLVD      AUSTIN  TX      78753   30.370945933000485      -97.6925542359997
2620 LAKE AUSTIN BLVD   AUSTIN  TX      78703   30.28190796500047       -97.77587573499966
14016 N FM 620 RD       AUSTIN  TX      78717   30.490713033000475      -97.76505815999968
3636 N FM 620 RD        AUSTIN  TX      78734   30.377873241000486      -97.9523496219997
15829 N IH 35 SVRD NB   AUSTIN  TX      78660   30.44853172000046       -97.61972641599965
```

Now, I wish we could just copy 'n' paste this result into Excel and it work, but that's not happening. But what we have done is build a very powerful search and a replace string so we can do this in a text editor.

You can actually save this in regex101 under SAVE & SHARE in the far left panel.

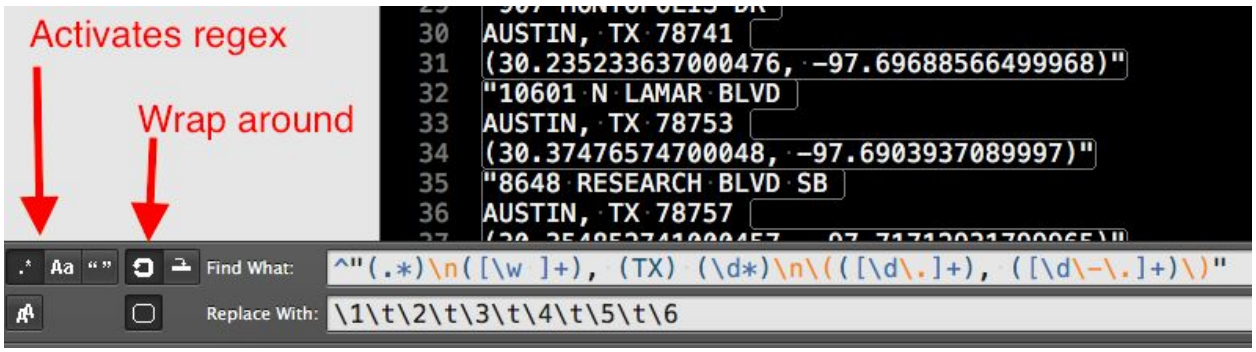
My example is saved here: https://regex101.com/r/pG9zC6/1

Search and replace in the text editor

We are almost done. Go ahead and launch your text editor if you haven't already. We'll show this first in Sublime Text.

- Take your TEST STRING data (not the expression, but the data) and paste it into a new next file in your text editor.
- Go under the Find menu to "Replace" (or do control-H for PC, command-F for Mac) to bring up the search window at the bottom of your text box.
- Copy the Regular Expression pattern you built in Regex101 and insert into the "Find what" in Sublime.
- Copy the Substitution pattern and insert it into the Replace field in "Replace with".
- Click on the button on the far-left of the Find what line, the one that has `.*` in it.
- Check the fourth box with a circle symbol, which is "Wrap around".

Here is what the search and replace screen looks like:



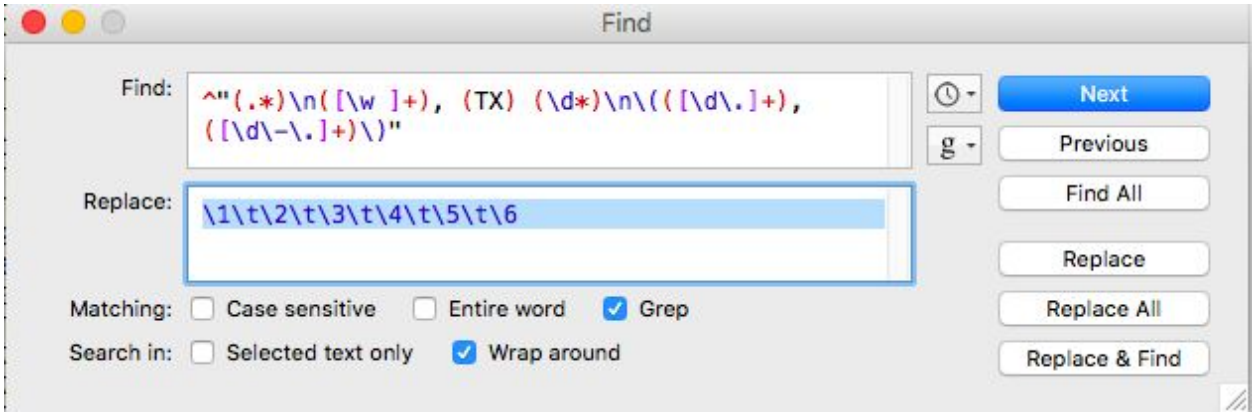
- Now, hit the "Replace All" button and watch the magic happen.
- Once you run the search and replace, you can copy and paste the results into a spreadsheet and it will be six distinct columns.

A	B	C	D	E	F
Address					
10111 N LAM	AUSTIN	TX	78753	30.3709459	-97.692554
2620 LAKE A	AUSTIN	TX	78703	30.281908	-97.775876
14016 N FM	AUSTIN	TX	78717	30.490713	-97.765058
3636 N FM 6	AUSTIN	TX	78734	30.3778732	-97.95235
15829 N IH 3	AUSTIN	TX	78660	30.4485317	-97.619726
917 N LAMAI	AUSTIN	TX	78703	30.2748468	-97.752332

Other text editors

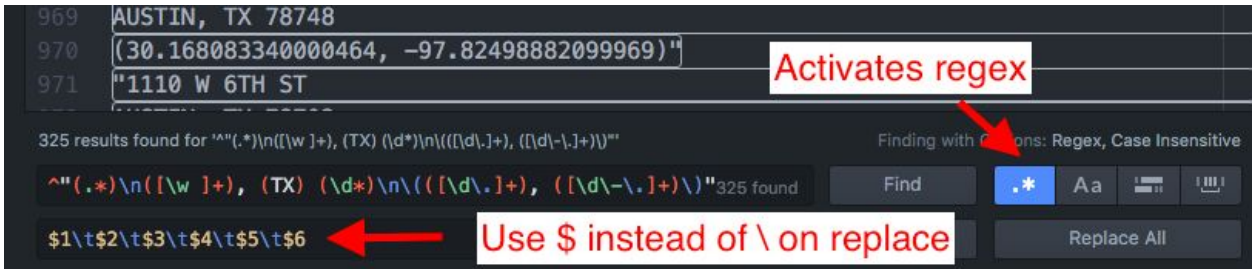
The search and replace functions are a little different in other text editors:

Text Wrangler



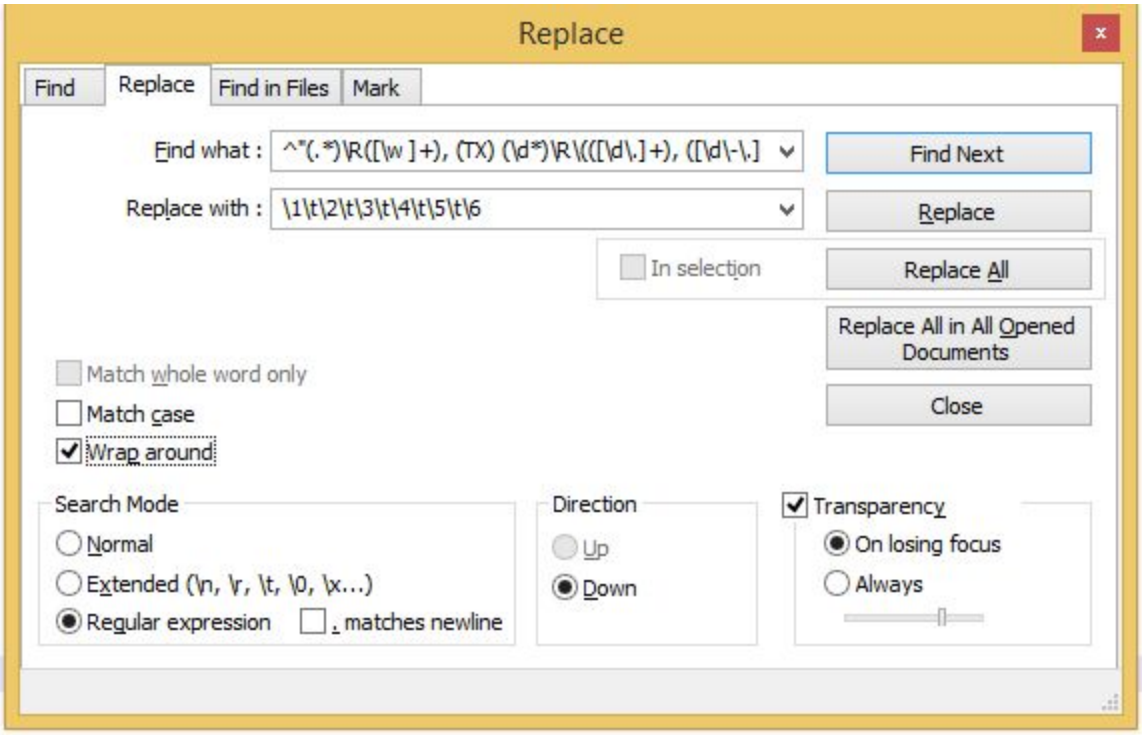
Atom text editor

Atom uses JavaScript notation instead of PCRE, so you have to use `$1` instead of `\1` on the replace strings:



NotePad++

Windows PCs sometimes handle the token for returns a little differently, using `\R` for a return instead of `\n`. Regex101 understands both, but sometimes Notepad++ wants that `\R`. So here is the search window for NotePad++:



More Regex

There are lots of sites and tutorials on regurlar expressions, but http://www.regular-expressions.info/ is one of my favorites.

And remember, if you are stumped by something, chances are you are not the first. Google and Stack Overflow are your friends.