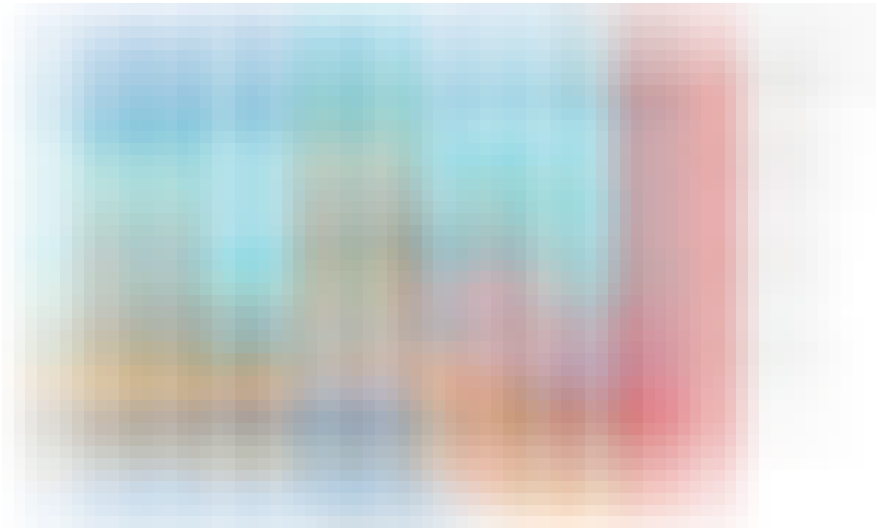
Kan Nishida Follow

Mar 28, 2016 · 10 min read



Restaurant types per State / Province

When Yelp published their data to the public for academic research (update link) (how many years ago?), I was kind of dissapointed because the data format was JSON and at that time I didn't really know how to work with such data format.

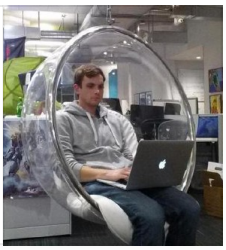
When we work on projects using Github as the source code repository, we manage all the issues (bugs), tasks or enhancement features, etc, as 'issues' at Github. And of course I want to do some analysis on that data. I mean, it is pretty basic stuff like, "how many issues do we have for each release or milestone ?", "how many issues are assigned to who ?", "What is the trend of the incoming and outgoing issues ?". Luckily, Github lets us extract these data, but the data comes in JSON format.

It has been a bit tricky to work with JSON data in general, not just with R, because of the nested and hierarchical nature of the data, until I met this amazing package called 'jsonlite', which helps us work with JSON data a lot easier and faster in R.

jeroenooms/jsonlite

jsonlite - A Robust, High Performance JSON Parser and Generator for R

github.com



Combined with dplyr, not only can we work with JSON data easier, but also we can take advantage of the nested nature of JSON data format to do something that we couldn't have done effectively with typical tabular data like CSV or relational database tables.

In this post, I'm going to use Yelp data and do a quick analysis to demonstrate how easy to work with JSON data thanks to the packages like 'jsonlite', 'dplyr', 'tibble', 'stringr', etc.

Get Yelp JSON data

You can download Yelp data sets from the following web site.

Yelp Dataset Challenge | Yelp

Yelp is a fun and easy way to find, recommend and talk about what's great - and not so great - in Sa...

www.yelp.com

I have one of the data called 'Yelp Academic Dataset Business', which contains information about the businesses listed on Yelp for selected states and provinces in US and Europe, hosted here if you would like to download quickly. This is the data I'm going to use for our data analysis.

And here is a snippet of the data.

```
{
  "business_id": "PK6aSizckHFWk8i0oxt5DA",
  "full_address": "400 Waterfront Dr E\nHomestead\nHomestead, PA 15120",
  "hours": {},
  "open": true,
  "categories": [
    "Burgers",
    "Fast Food",
    "Restaurants"
  ],
  "city": "Homestead",
  "review_count": 5,
  "name": "McDonald's",
  "neighborhoods": [
    "Homestead"
  ],
  "longitude": -79.910032,
  "state": "PA",
  "stars": 2,
  "latitude": 40.412086,
  "attributes": {
    "Take-out": true,
    "Wi-Fi": "free",
    "Drive-Thru": true,
    "Good For": {
      "dessert": false,
      "latenight": false,
      "lunch": false,
      "dinner": false,
      "breakfast": false,
      "brunch": false
    },
    "Caters": false,
    "Noise Level": "average",
    "Takes Reservations": false,
    "Delivery": false
```


As you can see, it's a JSON data and it's nested and hierarchical. For example, when you look at 'categories' there are three values of "Burgers", "Fast Food", and "Restaurants" at the same level. This is called 'Array', and it's useful to have multiple values assigned to one entity like "business" in this case. When you look at "attributes" it has several key-value pairs like '"Take-out": true' and one of them "Good For" has even its own child level information.

Let's import this data into R first.

Import JSON data

We can use 'fromJSON()' function from 'jsonlite' package to import most of the typical JSON files below.

```
library(jsonlite)

yelp <- fromJSON("yelp_academic_dataset_business.json")
```

However, when we run the above command we would actully get an error like below.

```
> yelp_test <- fromJSON("yelp_academic_dataset_business.json", flatten = TRUE)
Error in feed_push_parser(readBin(con, raw(), n), reset = TRUE) :
  parse error: trailing garbage
          : true}, "type": "business"} {"business_id": "UsFtqoBl7naz8A
                     (right here) ------^
```

This is because this JSON file turned out to be something called ‘NDJSON (Newline delimited JSON)’, which means there are multiple JSON values inside this file and each of the JSON values is considered as an independent object. In this particular case, each business information makes up one single JSON value therefore there are many JSON values inside of this JSON file. This could be used often in data streaming situations where each JSON data can be separated from other parts of the file so that each JSON data can be processed without waiting for the whole document to load.

Anyway, ‘jsonlite’ actually has a function to deal with this ‘NDJSON’ file type with ‘stream_in()’ function, so we can use it instead like below.

```
yelp <-
stream_in(file("yelp_academic_dataset_business.json"))
```

```
> head(yelp, 10)
               business_id                                      full_address hours.Tuesday.close
1  vcNAWiLM4dR7D2nwwJ7nCA 4840 E Indian School Rd\nSte 101\nPhoenix, AZ 85018               17:00
2  UsFtqoBl7naz8AVUBZMjQQ               202 McClure St\nDravosburg, PA 15034                <NA>
3  cE27W9VPgO88Qxe4ol6y_g            1530 Hamilton Rd\nBethel Park, PA 15234                <NA>
4  HZdLhv6COCleJMo7nPl-RA              301 S Hills Vlg\nPittsburgh, PA 15241               21:00
5  mVHrayjG3uZ_RLHkLj-AMg               414 Hawkins Ave\nBraddock, PA 15104               19:00
6  KayYbHCt-RkbGcPdGOThNg    141 Hawthorne St\nGreentree\nCarnegie, PA 15106                <NA>
7  b12U9TFESStdy7CsTtcOeg             718 Hope Hollow Rd\nCarnegie, PA 15106                <NA>
8  Sktj1eHQFuVa-M4bgnEh8g      920 Forsythe Rd\nCarnegie\nCarnegie, PA 15106                <NA>
9  3ZVKmuK2l7uXPE6lXY4Dbg            8 Logan St\nCarnegie\nCarnegie, PA 15106                <NA>
10 wJr6kSA5dchdgOdwH6dZ2w           2100 Washington Pike\nCarnegie, PA 15106               02:00
   hours.Tuesday.open hours.Friday.close hours.Friday.open hours.Monday.close hours.Monday.open
1               08:00              17:00              08:00              17:00              08:00
2                <NA>               <NA>               <NA>               <NA>               <NA>
3                <NA>               <NA>               <NA>               <NA>               <NA>
4               10:00              21:00              10:00              21:00              10:00
5               10:00              20:00              10:00               <NA>               <NA>
6                <NA>               <NA>               <NA>               <NA>               <NA>
7                <NA>               <NA>               <NA>               <NA>               <NA>
8                <NA>               <NA>               <NA>               <NA>               <NA>
9                <NA>               <NA>               <NA>               <NA>               <NA>
10              08:00              02:00              08:00              02:00              08:00
```

Note that you need to use ‘file()’ function to create a ‘connection’ for accessing to the files on your disk when you use ‘stream_in()’ function.

Flatten Yelp data frame

Let's find out how the data has been imported by quickly running 'str()' function.

```
str(yelp)
```

```
> str(yelp)
'data.frame':   61184 obs. of  15 variables:
 $ business_id  : chr  "vcNAWiLM4dR7D2nwwJ7nCA" "UsFtqoBl7naz8AVUBZMjQQ" "cE27W9VPgO88Qxe4ol6y_g" "HZdLhv6COCleJMo7
nPl-RA" ...
 $ full_address : chr  "4840 E Indian School Rd\nSte 101\nPhoenix, AZ 85018" "202 McClure St\nDravosburg, PA 15034"
"1530 Hamilton Rd\nBethel Park, PA 15234" "301 S Hills Vlg\nPittsburgh, PA 15241" ...
 $ hours        :'data.frame':  61184 obs. of  7 variables:
  ..$ Tuesday  :'data.frame':   61184 obs. of  2 variables:
  .. ..$ close: chr  "17:00" NA NA "21:00" ...
  .. ..$ open : chr  "08:00" NA NA "10:00" ...
  ..$ Friday   :'data.frame':   61184 obs. of  2 variables:
  .. ..$ close: chr  "17:00" NA NA "21:00" ...
  .. ..$ open : chr  "08:00" NA NA "10:00" ...
  ..$ Monday   :'data.frame':   61184 obs. of  2 variables:
  .. ..$ close: chr  "17:00" NA NA "21:00" ...
  .. ..$ open : chr  "08:00" NA NA "10:00" ...
  ..$ Wednesday:'data.frame':   61184 obs. of  2 variables:
  .. ..$ close: chr  "17:00" NA NA "21:00" ...
  .. ..$ open : chr  "08:00" NA NA "10:00" ...
  ..$ Thursday :'data.frame':   61184 obs. of  2 variables:
  .. ..$ close: chr  "17:00" NA NA "21:00" ...
  .. ..$ open : chr  "08:00" NA NA "10:00" ...
  ..$ Sunday   :'data.frame':   61184 obs. of  2 variables:
  .. ..$ close: chr  NA NA NA "18:00" ...
```

As you can see 'hours' variable is actually a data frame that contains 7 data frames each of which is for a weekday like 'Tuesday'. And the weekday variables themselves are data frames and each contains two 'character' variables of 'open' and 'close'.

This is reflecting the original JSON data structure, but it is a bit confusing for analyzing data in R. We can use 'flatten()' function from 'jsonlite' package to make the nested hiearchical data structure into a flatten manner by assigning each of the nested variable as its own column as much as possible.

```
yelp_flat <- flatten(yelp)
```

```
str(yelp_flat)
```

```
$ hours.Tuesday.close                          : chr  "17:00" NA NA "21:00" ...
$ hours.Tuesday.open                           : chr  "08:00" NA NA "10:00" ...
$ hours.Friday.close                           : chr  "17:00" NA NA "21:00" ...
$ hours.Friday.open                            : chr  "08:00" NA NA "10:00" ...
$ hours.Monday.close                           : chr  "17:00" NA NA "21:00" ...
$ hours.Monday.open                            : chr  "08:00" NA NA "10:00" ...
$ hours.Wednesday.close                        : chr  "17:00" NA NA "21:00" ...
$ hours.Wednesday.open                         : chr  "08:00" NA NA "10:00" ...
$ hours.Thursday.close                         : chr  "17:00" NA NA "21:00" ...
$ hours.Thursday.open                          : chr  "08:00" NA NA "10:00" ...
$ hours.Sunday.close                           : chr  NA NA NA "18:00" ...
$ hours.Sunday.open                            : chr  NA NA NA "11:00" ...
$ hours.Saturday.close                         : chr  NA NA NA "21:00" ...
$ hours.Saturday.open                          : chr  NA NA NA "10:00" ...
$ attributes.By Appointment Only               : logi  TRUE NA NA NA NA NA ...
$ attributes.Happy Hour                        : logi  NA TRUE NA NA FALSE TRUE ...
```

Now the data structure looks a lot easier to grasp and even the data looks easier to see.

Before printing out the data I'm going to use 'as_data_frame()' function from the new package called 'tibble', which was released just last week from Hadley Wickham and the team to make it easier to see the data frame data in R console UI.

We can use 'as_data_frame()' function to convert our data frame to be 'tbl_df', which is an extended version of the data frame. Once it's in 'tbl_df' type, it automatically shows only the first 10 variables in the console output by simply typing the data frame name so you don't need to call 'head()' function separately. Also, it shows a data type for each variable in the output.

```
library(tibble)

yelp_tbl <- as_data_frame(yelp_flat)

yelp_tbl
```

```
> yelp_tbl
Source: local data frame [61,184 x 105]

            business_id                                    full_address  open categories        city
                  <chr>                                           <chr> <lgl>     <list>       <chr>
1  vcNAWiLM4dR7D2nwwJ7nCA 4840 E Indian School Rd\nSte 101\nPhoenix, AZ 85018  TRUE  <chr [2]>     Phoenix
2  UsFtqoBl7naz8AVUBZMjQQ               202 McClure St\nDravosburg, PA 15034  TRUE  <chr [1]>  Dravosburg
3  cE27W9VPgO88Qxe4ol6y_g            1530 Hamilton Rd\nBethel Park, PA 15234 FALSE  <chr [3]> Bethel Park
4  HZdLhv6COCleJMo7nPl-RA              301 S Hills Vlg\nPittsburgh, PA 15241  TRUE  <chr [6]>  Pittsburgh
5  mVHrayjG3uZ_RLHkLj-AMg               414 Hawkins Ave\nBraddock, PA 15104  TRUE  <chr [5]>    Braddock
6  KayYbHCt-RkbGcPdGOThNg    141 Hawthorne St\nGreentree\nCarnegie, PA 15106  TRUE  <chr [4]>    Carnegie
7  b12U9TFESStdy7CsTtcOeg             718 Hope Hollow Rd\nCarnegie, PA 15106  TRUE  <chr [2]>    Carnegie
8  Sktj1eHQFuVa-M4bgnEh8g      920 Forsythe Rd\nCarnegie\nCarnegie, PA 15106  TRUE  <chr [2]>    Carnegie
9  3ZVKmuK2l7uXPE6lXY4Dbg           8 Logan St\nCarnegie\nCarnegie, PA 15106  TRUE  <chr [2]>    Carnegie
10 wJr6kSA5dchdgOdwH6dZ2w           2100 Washington Pike\nCarnegie, PA 15106  TRUE  <chr [4]>    Carnegie
..                   ...                                             ...   ...         ...          ...
Variables not shown: review_count <int>, name <chr>, neighborhoods <list>, longitude <dbl>, state <chr>, stars
  <dbl>, latitude <dbl>, type <chr>, hours.Tuesday.close <chr>, hours.Tuesday.open <chr>, hours.Friday.close
  <chr>, hours.Friday.open <chr>, hours.Monday.close <chr>, hours.Monday.open <chr>, hours.Wednesday.close <chr>,
  hours.Wednesday.open <chr>, hours.Thursday.close <chr>, hours.Thursday.open <chr>, hours.Sunday.close <chr>,
  hours.Sunday.open <chr>, hours.Saturday.close <chr>, hours.Saturday.open <chr>, attributes.By Appointment Only
  <lgl>, attributes.Happy Hour <lgl>, attributes.Accepts Credit Cards <list>, attributes.Good For Groups <lgl>,
  attributes.Outdoor Seating <lgl>, attributes.Price Range <int>, attributes.Good for Kids <lgl>,
  attributes.Alcohol <chr>, attributes.Noise Level <chr>, attributes.Has TV <lgl>, attributes.Attire <chr>,
```

As you look closer you would notice ‘categories’ variable is ‘list’ data type and it is ‘nested’. This is because ‘categories’ includes an array data in the original JSON data.


```
{
  "business_id": "PK6aSizckHFWk8i0oxt5DA",
  "full_address": "400 Waterfront Dr E\nHomestead\nHomestead, PA 15120",
  "hours": {},
  "open": true,
  "categories": [
    "Burgers",
    "Fast Food",
    "Restaurants"
  ],
  "city": "Homestead",
  "review_count": 5,
  "name": "McDonald's",
  "neighborhoods": [
    "Homestead"
  ],
  "longitude": -79.910032,
  "state": "PA",
  "stars": 2,
```


So even after the ‘flatten()’ operation this type of variables are registered as ‘list’ data type and it has a list of the values in each row of the data frame. You can easily show the values inside of this type of variable by using as.character() function like below.

```
yelp_tbl %>% mutate(categories = as.character(categories))
%>% select(categories)
```

```
Source: local data frame [61,184 x 1]


categories

<chr>
1                                                                                  c("Doctors", "Health & Medica
l")
2                                                                                                       Nightl
ife
3                                                                                c("Active Life", "Mini Golf", "Gol
f")
4  c("Shopping", "Home Services", "Internet Service Providers", "Mobile Phones", "Professional Services", "Electron
ics
5                                                 c("Bars", "American (New)", "Nightlife", "Lounges", "Restaurant
s")
6                                                   c("Bars", "American (Traditional)", "Nightlife", "Restaurant
s")
7                                                                                  c("Auto Repair", "Automotiv
e")
8                                                                                    c("Active Life", "Mini Gol
f")
9                                                                                c("Home Services", "Contractor
```

Remove unnecessary variables

Now when you look at the data again, looks there are bunch of variables whose name starts with either ‘hours’ or ‘attributes’.

```
Source: local data frame [61,184 x 105]

             business_id                                 full_address  open categories        city
                   <chr>                                       <chr> <lgl>     <list>       <chr>
1  vcNAWiLM4dR7D2nwwJ7nCA 4840 E Indian School Rd\nSte 101\nPhoenix, AZ 85018  TRUE  <chr [2]>     Phoenix
2  UsFtqoBl7naz8AVUBZMjQQ               202 McClure St\nDravosburg, PA 15034  TRUE  <chr [1]>  Dravosburg
3  cE27W9VPgO88Qxe4ol6y_g           1530 Hamilton Rd\nBethel Park, PA 15234 FALSE  <chr [3]> Bethel Park
4  HZdLhv6COCleJMo7nPl-RA             301 S Hills Vlg\nPittsburgh, PA 15241  TRUE  <chr [6]>  Pittsburgh
5  mVHrayjG3uZ_RLHkLj-AMg               414 Hawkins Ave\nBraddock, PA 15104  TRUE  <chr [5]>    Braddock
6  KayYbHCt-RkbGcPdGOThNg     141 Hawthorne St\nGreentree\nCarnegie, PA 15106  TRUE  <chr [4]>    Carnegie
7  b12U9TFESStdy7CsTtcOeg           718 Hope Hollow Rd\nCarnegie, PA 15106  TRUE  <chr [2]>    Carnegie
8  Sktj1eHQFuVa-M4bgnEh8g       920 Forsythe Rd\nCarnegie\nCarnegie, PA 15106  TRUE  <chr [2]>    Carnegie
9  3ZVKmuK2l7uXPE6lXY4Dbg            8 Logan St\nCarnegie\nCarnegie, PA 15106  TRUE  <chr [2]>    Carnegie
10 wJr6kSA5dchdgOdwH6dZZw           2100 Washington Pike\nCarnegie, PA 15106  TRUE  <chr [4]>    Carnegie
..                   ...                                         ...   ...        ...          ...
Variables not shown: review_count <int>, name <chr>, neighborhoods <list>, longitude <dbl>, state <chr>, stars
  <dbl>, latitude <dbl>, type <chr>, hours.Tuesday.close <chr>, hours.Tuesday.open <chr>, hours.Friday.close
  <chr>, hours.Friday.open <chr>, hours.Monday.close <chr>, hours.Monday.open <chr>, hours.Wednesday.close <chr>,
  hours.Wednesday.open <chr>, hours.Thursday.close <chr>, hours.Thursday.open <chr>, hours.Sunday.close <chr>,
  hours.Sunday.open <chr>, hours.Saturday.close <chr>, hours.Saturday.open <chr>, attributes.By Appointment Only
  <lgl>, attributes.Happy Hour <lgl>, attributes.Accepts Credit Cards <list>, attributes.Good For Groups <lgl>,
  attributes.Outdoor Seating <lgl>, attributes.Price Range <int>, attributes.Good for Kids <lgl>,
  attributes.Alcohol <chr>, attributes.Noise Level <chr>, attributes.Has TV <lgl>, attributes.Attire <chr>,
  attributes.Good For Dancing <lgl>, attributes.Delivery <lgl>, attributes.Coat Check <lgl>, attributes.Smoking
  <chr>, attributes.Take-out <lgl>, attributes.Takes Reservations <lgl>, attributes.Waiter Service <lgl>,
  attributes.Wi-Fi <chr>, attributes.Caters <lgl>, attributes.Drive-Thru <lgl>, attributes.Wheelchair Accessible
  <lgl>, attributes.BYOB <lgl>, attributes.Corkage <lgl>, attributes.BYOB/Corkage <chr>, attributes.Order at
  Counter <lgl>, attributes.Good For Kids <lgl>, attributes.Dogs Allowed <lgl>, attributes.Open 24 Hours <lgl>,
  attributes.Accepts Insurance <lgl>, attributes.Ages Allowed <chr>, attributes.Ambience.romantic <lgl>,
  attributes.Ambience.intimate <lgl>, attributes.Ambience.classy <lgl>, attributes.Ambience.hipster <lgl>,
  attributes.Ambience.divey <lgl>, attributes.Ambience.touristy <lgl>, attributes.Ambience.trendy <lgl>,
  attributes.Ambience.upscale <lgl>, attributes.Ambience.casual <lgl>, attributes.Good For.dessert <lgl>,
  attributes.Good For.latenight <lgl>, attributes.Good For.lunch <lgl>, attributes.Good For.dinner <lgl>,
  attributes.Good For.breakfast <lgl>, attributes.Good For.brunch <lgl>, attributes.Parking.garage <lgl>,
  attributes.Parking.street <lgl>, attributes.Parking.validated <lgl>, attributes.Parking.lot <lgl>,
```

For now, we are not really interested in those data so let’s remove them with ‘select()’ command. If you want to know more detail on ‘select()’ operation check out this post.

```
yelp_tbl %>%
 select(-starts_with("hours"), -starts_with("attribute"))
```

This would actually drop almost 90 variables, and the result looks much less. But these are enough information for us to see what type of businesses are in this data.

Count how many restaurants there are

Now, let's find out how many 'Restaurant' business in the data. We can use 'str_detect()' function from 'stringr' package to find the businesses whose 'categories' variable values contain 'Restaurant' text. If you want to know more detail about this function check out this post.

```
library(stringr)
```

```
yelp_tbl %>% select(-starts_with("hours"), -
starts_with("attribute")) %>%
 filter(str_detect(categories, "Restaurant"))
```


The cool thing about this function is that even when we have a 'list' data type variable it can go inside the list values and do the text matching. Pretty awesome.

Now, number of the rows is 21,892, as opposed to the original of 61,184 rows. That means there are 21,892 'Restaurant' related businesses in this data out of 61,184 businesses. To confirm if this is really the case, let's look at the categories column with 'as.character()' function again.

```
yelp_tbl %>% select(-starts_with("hours"), -
starts_with("attribute")) %>%
 filter(str_detect(categories, "Restaurant")) %>%
 mutate(categories = as.character(categories)) %>%
select(categories)
```


Looks all the rows (well, all the first 10 rows anyway) look like they have "Restaurants' as one of the categories.

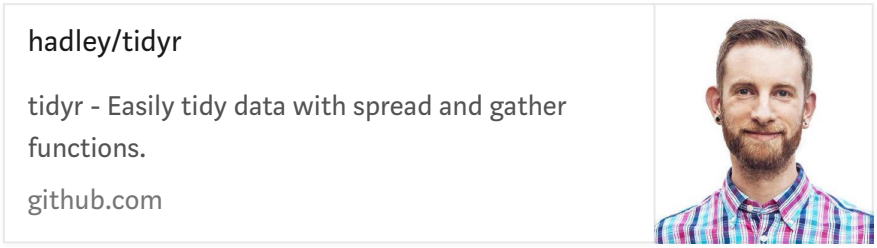
And this is interesting because people on Yelp are tagging different genres for each "Restaurant" business. For example, some restaurants could be considered as "Bars", "American (New)", "Pub", etc. So one of the interesting question would be, "what type of the restaurants are more common in this data set?"

The easiest way to answer this question is to break out this 'categories' list data type variable and assign each value inside the list into each row. So essentially we'll have a variable called 'categories' and this will have just one category value for each row. This means, some businesses might be repeated across rows many times, but that's ok because our concern for now is to count each restaurant category type.

Unnest a list variable

To break out 'categories' variable and create one row for each value, we can use 'unnest()' function from tidyr package, which is another great

package from ‘Hadleyverse’ to help making raw data into a ‘tidy’ format.



I will talk about more on ‘tidy’ format with ‘tidyr’ package in a separate post, but for now I am going to simply use ‘unnest()’ function to “unnest” the ‘categories’ variable like below.

```
library(tidyr)
```

```
yelp_tbl %>% select(-starts_with("hours"), -
starts_with("attribute")) %>%
 filter(str_detect(categories, "Restaurant")) %>%
 unnest(categories) %>%
 select(name, categories)
```


As you can see, ‘Emil’s Lounge’ is now repeated 5 times, for example. This is because it has those 5 different categories assigned to this business. This will allow us to do a quick summarization with ‘count()’ function like below.

```
yelp_tbl %>% select(-starts_with("hours"), -
starts_with("attribute")) %>%
 filter(str_detect(categories, "Restaurant")) %>%
 unnest(categories) %>%
```

```
 select(name, categories) %>%
 count(categories)
```


And if you want to see the top categories you can simply use ‘arrange()’ function to sort.

```
yelp_tbl %>% select(-starts_with("hours"), -
starts_with("attribute")) %>%
 filter(str_detect(categories, "Restaurant")) %>%
 unnest(categories) %>%
 select(name, categories) %>%
 count(categories) %>%
 arrange(desc(n))
```


Let’s get rid of the rows with ‘Restaurants’ category because we know every single rows in this data set has something to do with ‘Restaurant’ now.

```
yelp_tbl %>% select(-starts_with("hours"), -
starts_with("attribute")) %>%
 filter(str_detect(categories, "Restaurant")) %>%
 unnest(categories) %>%
 filter(categories != "Restaurants") %>%
```

```
  count(categories) %>%
  arrange(desc(n))
```

When you run the above command you will get something like below. You can see ‘Fast Food’ is the number one, and ‘Pizza’ and ‘Mexican’ come after.

What are the most common restaurant types per state / province?

In this data set, there is a variable called ‘state’ that contains state names or province names of US and some European countries. So, what if we want to know what restaurant categories are more frequently showing up for each state ?

We can simply add ‘state’ variable into ‘count()’ function like below.

```
yelp_tbl %>% select(-starts_with("hours"), -
starts_with("attribute")) %>%
 filter(str_detect(categories, "Restaurant")) %>%
 unnest(categories) %>%
 filter(categories != "Restaurants") %>%
 count(state, categories) %>%
 arrange(desc(n))
```

Now, let's take a look at the top restaurant category for each state. We can use '**group_by**()' function to group the data by state and use '**top_n**()' function to keep only the top category. Both functions are from 'dplyr' package.

```
yelp_tbl %>% select(-starts_with("hours"), -
starts_with("attribute")) %>%
 filter(str_detect(categories, "Restaurant")) %>%
 unnest(categories) %>%
 filter(categories != "Restaurants") %>%
 count(state, categories) %>%
 group_by(state) %>%
 top_n(1, n)
```


As you can see there are 5 entries for "FIF" province, this is because they are all tie. But the values are just '1' and that's a really small number compared to the others. So let's filter out those small numbers from the data before the **group_by** and **top_n** steps.

```
yelp_tbl %>% select(-starts_with("hours"), -
starts_with("attribute")) %>%
 filter(str_detect(categories, "Restaurant")) %>%
 unnest(categories) %>%
 filter(categories != "Restaurants") %>%
 count(state, categories) %>%
```

```
  filter(n > 10) %>%
  group_by(state) %>%
  top_n(1, n)
```


The result is a list of the top restaurant category for each of the 12 state or province.

. . .

This is a very simple analysis on Yelp business data. But given where we started with Yelp business data in the raw JSON format, hope this has demonstrated how quickly, incrementally, and iteratively we can get some interesting information out of such un-traditional (not tabular) data format relatively easily and quickly.

> *If you liked this post, please click the green heart below to share with the world!*

. . .

hadley/dplyr

dplyr - Plyr specialised for data frames: faster & with remote datastores

github.com

. . .