Zooniverse Data Cleaning Tutorial. Teacher Edition

Adopt a Pixel Project

2022-04-22

In this tutorial we will be going through the data obtained from Zooniverse to clean and visualize the data as well a understand the importance in proper file naming conventions and its implications on data science tasks. Follow along with code provided below and see if you can do it yourself!

#Part 1: Setting up R/Rstudio

Before we begin to dive into the data, we need to download the required "packages" in order to make cleaning the data easier. Much like python, R provides packages that have built in codes and functions that make data science tasks easier. Knowing what packages could be used makes your job as budding data scientists and researchers much easier. In this part of the tutorial, we will be using the "tidyverse" and "stringr" packages to import and parse the data to get variables that will help us understand the data.

If this is your first time using R, you must install the packages first before using them. To do so, copy and paste the code below into the console at the bottom left quadrant and hit enter. Do this one at a time to ensure proper installation.

install.packages("tidyverse") install.packages("stringr") install.packages("dplyr")

Now that the packages are installed, we can call on the packages whenever we need the functions provided. You only need to call on any package once throughout a project so lets call on them to begin the cleaning process. To do so, we use the library() function.

Now that we have installed the packages and called on them to use in our cleaning project, we are ready to start working with the Zooniverse data! To start we need to import the data from our Working Directory. Think of the Working Directory as the location R is looking for information in. If you saved this tutorial in the same file that the data set obtained from Zooniverse is in, you should be good to go. If not we need to set the Working Directory to the correct file location. To do this, look at the bottom right quadrant in R Studio. It should show you what file location is selected, if the location is wrong, navigate to the right location, then click on the cogwheel icon and click "Set as Working Directory". If you Zooniverse data file is already showing in the file location, the file location then you are ready to import the data.

#Part 2: Importing data into R/R Studio and Filtering

Follow the code below to import the data from Zooniverse. In this example, I name the dataset "Zooniverse" but you can name it whatever you like, just be sure to reference the data set with the same name going forward.

```r
Zooniverse <- read_csv("landcoverelements.csv")
```

```
## Rows: 3642 Columns: 14
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## chr (8): user_name, user_ip, workflow_name, created_at, expert, metadata, an...
## dbl (5): classification_id, user_id, workflow_id, workflow_version, subject_ids
## lgl (1): gold_standard
```

```
##
## i Use ‘spec()‘ to retrieve the full column specification for this data.
## i Specify the column types or set ‘show_col_types = FALSE‘ to quiet this message.
```

If done correctly, you should see the data set in your environment located at the top left quadrant. Showing is the entire data set, however we do not need the entirety of it, as there are multiple workflows and versions within the same data set. We specifically care about "workflow_id" of 20048, and "workflow_version" 13.9. In order to pull only the entries with these specifications, we can use the filter function to (as you can already guess) filter out the data to what we are looking for.

In the next chunk of code, we use the filter function to create a new data set with the workflow ID and version criteria above. I name this data set as "zoo" but you can name it whatever you like

```
zoo <- filter(Zooniverse, workflow_id == 20048 & workflow_version == 13.9)
```

To check that we have the correct segment from the data set, we can use the view() function to see the data set, or if you want a short preview use the head() function to get the first six entries in the data set.

```
head(zoo)
```

```
## # A tibble: 6 x 14
##   classification_id user_name   user_id user_ip        workflow_id workflow_name
##               <dbl> <chr>         <dbl> <chr>                <dbl> <chr>
## 1         373043461 pedernelson 1805556 4c0253a3589f4~       20048 Sort photos ~
## 2         373043479 pedernelson 1805556 4c0253a3589f4~       20048 Sort photos ~
## 3         373043513 pedernelson 1805556 4c0253a3589f4~       20048 Sort photos ~
## 4         373043546 pedernelson 1805556 4c0253a3589f4~       20048 Sort photos ~
## 5         373043613 pedernelson 1805556 4c0253a3589f4~       20048 Sort photos ~
## 6         373044605 pedernelson 1805556 a8656956265cf~       20048 Sort photos ~
## # ... with 8 more variables: workflow_version <dbl>, created_at <chr>,
## #   gold_standard <lgl>, expert <chr>, metadata <chr>, annotations <chr>,
## #   subject_data <chr>, subject_ids <dbl>
```

Question Time! Now that we have pulled the correct data according to our criteria, we can begin the cleaning process. How many observations and variables are in this filtered data set? Use the view() function to look at the "annotations" and the "subject_data". What do you see in the annotations that look important? You can expand the column by dragging the column wider. Take note of all the different land covers present in the annotations. Does the format look the same? What about subject data? Is the format the same for all the subject data?

Answer: There are 1644 observations and 15 variables in the filtered data set. In the annotations, there are land cover elements such as Sky, Urban, and Herbaceous that look important. The format of the annotations look kthe same, however the subject data looks different.

#Part 3: Data Cleaning and Variable Creation

Lets start with the subject data. As you can see, the subject data is not uniform throughout the data set. This is a problem because if they are not structured the same, we cannot get the information necessary to create a clean data set. Open the subject data key image found with the R Markdown file. If the subject data follows this same format then it is the new and clean structure. We want all entries that follow this structure. From the key, we can see that the "GOLC" starts all new iterations of subject data, so we want all entries that have this indictator in the subject_data. In the following code, we will create a new column in the data set that indicates if the subject data follows the same format. From there we can easily filter (like before!) the data that follows this structure.

```
zoo$structured <- ifelse(grepl("GOLC", zoo$subject_data),1,0)
```

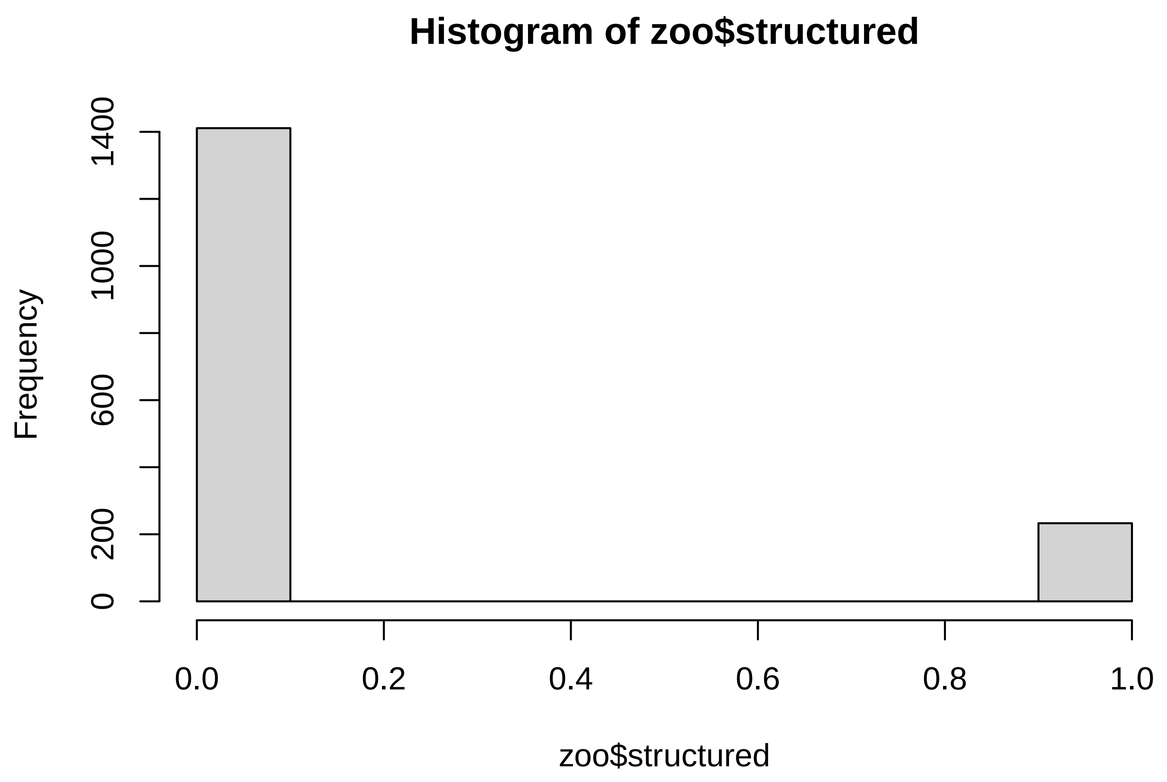
The code above shows us creating a new column using an if else function. To break it down, on the left of the arrow we are calling our data set and use the "$" sign to reference a specific variable. Since the "structured" variable does not exist in the data set, we are adding it a new one. This explains why we went from 14 to 15 variables! To the right of the arrow, we created an "ifelse" function in combination with the "grepl" function. The "grepl" function will search for a pattern, which in this case it is searching for the pattern "GOLC" within the subject_data of the zoo data frame. This is the condition set within the "ifelse" function. If the pattern is found, then a value of 1 is assigned to the new structured column we created. If the pattern is not found then a 0 is assigned.

TIP! When parsing for a pattern, make sure that the pattern you are looking for matches EXACTLY as what is found in the data. That includes capitalization, puncuation marks, etc.

DATA FACT: A variable that is consisted of only 1 or 0 is called a binary variable! Just like the computer language! Binary variables answer a yes no question by assigning a value to yes's and no's. These can be however you want to structure so as data scientists we want to ensure that the variables we create or are given are easily interpretable.

Lets see how many structured and unstructured entries there are in the data set. We can do so by creating a simple histogram giving us the count of structured and unstructured data.

```
hist(zoo$structured)
```



As we can see, there are almost 5 times as much unstructured data as there is as structured data. That is okay! We are learning why assigning structure to files are important.

Let's filter out the structured data into a new data set to work with. We can use code similar to the one we used to filter out the workflow ID and version. I am going to call this new data set "zooclean".

```
zooclean <- filter(zoo, structured == 1)
```

We should now have a data set with much fewer observations, but all the subject_data matches the same structure. We can now begin to parse the data set for values to create a full data set. Let's start with the annotations. In the annotation column, you should have found all the land cover elements that we want to extract out of the annotations. We can use the same code that we used to create the ifelse() function on the "GOLC" pattern, but this time change the parts around to find the land cover elements! The elements we are looking for are Sky, Urban, Barren, Cultivated, Trees, Herbaceous, Shrubs, Open Water, and Can't label. The first one is done for you!

HINT: For "Open Water" call the column Water, and for "Can't Label" call the column None.

```
zooclean$Sky <- ifelse(grepl("Sky", zooclean$annotations), 1, 0)  #Follow this structure to parse the r
zooclean$Urban <- ifelse(grepl("Urban", zooclean$annotations), 1, 0)
zooclean$Barren <- ifelse(grepl("Barren", zooclean$annotations), 1, 0)
zooclean$Cultivated <- ifelse(grepl("Cultivated", zooclean$annotations), 1, 0)
zooclean$Trees <- ifelse(grepl("Trees", zooclean$annotations), 1, 0)
zooclean$Herbaceous <- ifelse(grepl("Herbaceous", zooclean$annotations), 1, 0)
zooclean$Shrubs <- ifelse(grepl("Shrubs", zooclean$annotations), 1, 0)
zooclean$Water <- ifelse(grepl("Open Water", zooclean$annotations), 1, 0)
zooclean$None <- ifelse(grepl("Can't label", zooclean$annotations), 1, 0)
```

Let's use the head() function to see the inclusion of these land cover elements are variables!

```
head(zooclean)
```

```
## # A tibble: 6 x 24
##   classification_id user_name   user_id user_ip        workflow_id workflow_name
##               <dbl> <chr>         <dbl> <chr>                <dbl> <chr>
## 1         373043461 pedernelson 1805556 4c0253a3589f4~       20048 Sort photos ~
## 2         373043479 pedernelson 1805556 4c0253a3589f4~       20048 Sort photos ~
## 3         373043513 pedernelson 1805556 4c0253a3589f4~       20048 Sort photos ~
## 4         373043546 pedernelson 1805556 4c0253a3589f4~       20048 Sort photos ~
## 5         373043613 pedernelson 1805556 4c0253a3589f4~       20048 Sort photos ~
## 6         373044605 pedernelson 1805556 a8656956265cf~       20048 Sort photos ~
## # ... with 18 more variables: workflow_version <dbl>, created_at <chr>,
## #   gold_standard <lgl>, expert <chr>, metadata <chr>, annotations <chr>,
## #   subject_data <chr>, subject_ids <dbl>, structured <dbl>, Sky <dbl>,
## #   Urban <dbl>, Barren <dbl>, Cultivated <dbl>, Trees <dbl>, Herbaceous <dbl>,
## #   Shrubs <dbl>, Water <dbl>, None <dbl>
```

Now that the annotations are out of the way, let's look at the subject_data. Open the Subject Data Key file and use this to see what information we need to extract from the subject data! (You do not need to open this in R, just open it from your normal file explorer to see the Key.) The reason we need this key is to know what information is in the subject_data as well as the positions they are located. The subject data contains different information for every observation, however since it is structured the same we can parse by the position of the information instead of looking for a pattern! Remember we can only do this because the structures are the same, which is why naming files in a structured format is important as a Data Scientist!

To parse by position we will be using the substr() function. This function will search extract whatever is in a string by the position of the characters. Normally to find the position you would need to count to find where the character of the information to be extracted starts and where it ends. To save time, the positions are already found in the key!

```
zooclean$MUC <- substr(zooclean$subject_data, 116, 118) #The first  code is done for you. See how in th
zooclean$IGBP <- substr(zooclean$subject_data, 120, 125)
zooclean$direction <- substr(zooclean$subject_data, 104, 108)
zooclean$longitude <- substr(zooclean$subject_data, 46, 54)
zooclean$latitude <- substr(zooclean$subject_data, 56, 65)
```

#Part 4: Advanced Cleaning

With that we have extracted all the information from the subject_data! However, we can take it a step further to look nicer. (clean data is the best data!) We can clean up the longitude and latitude by changing the place holder characters (n,s,e,w,p) for what they actually represent! To do so, we will be using the str_remove() function and the str_replace_all() function to either remove or replace specific characters. The code chunk below cleans up the Longitude column. Try to clean up the latitude using the code below as an example!

```
zooclean$longitude <- str_remove(zooclean$longitude, "n") #Removes the "n" character completely.
zooclean$longitude <-str_replace_all(zooclean$longitude, "s", "-") #Replaces the "s" character with a "
zooclean$longitude <- str_replace_all(zooclean$longitude, "p", ".") #Replaces the "p" character with th
```

```
zooclean$latitude <- str_remove(zooclean$latitude, "e")
zooclean$latitude <- str_replace_all(zooclean$latitude, "w", "-")
zooclean$latitude <- str_replace_all(zooclean$latitude, "p", ".")
```

There are also some IGBP values that have not parsed correctly. There are instances where "_IGBP1" is showing instead of "IGBP12". The code below will replace all the incorrect IGBP values with the correct version.

```
zooclean$IGBP <- str_replace_all(zooclean$IGBP, "_IGBP1", "IGBP12")
```

This part is optional, but we can also remove the placeholder zeros in our direction field by using the str_remove_all() function.

```
zooclean$direction <- str_remove_all(zooclean$direction,"0")
```

#Part 5: Merging Look Up Tables

With that our data set is now clean! This can now be used for modeling and visualization purposes, however lets add one more thing to make sense of the MUC and IGBP values. We also have the MUC Classification and IGBP Classification look up tables. Lets merge these two look up tables into our clean data set so we can see exactly what land type each location is! Lets start by importing both the MUC Classification and IGBP Classification look up tables into R.

```
IGBPtable <- read_csv("IGBP Classifications.csv")
```

```
## Rows: 18 Columns: 3
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## chr (3): IGBP Classification, IGBP, Description
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```
MUCtable <- read_csv("MUC Classifications.csv")
```

```
## Rows: 35 Columns: 2
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## chr (2): MUC, MUC Classification
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

Now that both look up tables are imported, we can begin merging by matching the information in each data set to each other. To do this we will be using the left_join() function.

```
zooclean <- left_join(zooclean, MUCtable, by = "MUC") #Since MUC is present in both data sets, we can u
```

Try to join the IGBP Classifications table to the main data set use the code above as an example!

```
zooclean <- left_join(zooclean, IGBPtable, by = "IGBP")
```

#Part 6: Exporting Cleaned Data from R/RStudio

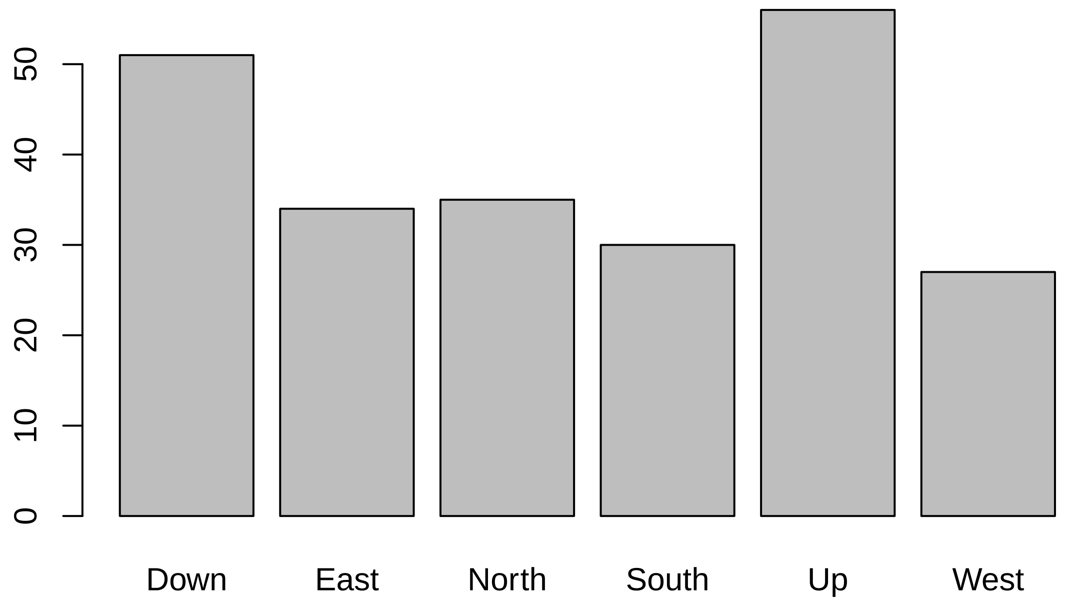
With that our data cleaning is done! We can now extract the clean data set and save it for future use. The code below will write the clean data set into a usable csv file!

```
write.csv(zooclean, "Zooniverse_clean.csv")
```

#Part 7: Priliminary Analysis

Since we got this far, lets check out some simple visualizations using the cleaned data set to see if we can understand what is going on in Zooniverse. We can start by creating a barplot to see how many photographs in each direction are present in our clean data. To do so, we create a small data table called X that contains all the directions present. We then can use the barplot() function to graph the frequency of each direction.

```
x <- table(zooclean$direction) #We are assigning "x" the table of the directional values.
barplot(x)
```



Question Time! Each location site should have a photo of each of the six directions. In our cleaned data set, which direction shows up the most? the least? Why could this be an issue when interpreting site data? Any opinion on how we can fix this in the future?

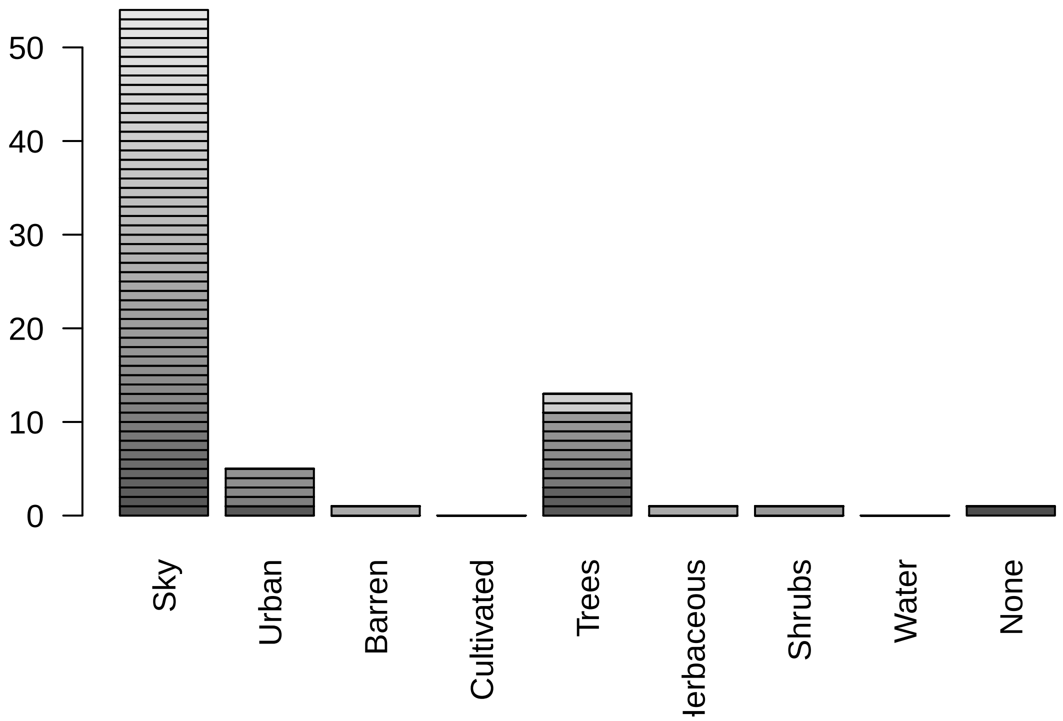
Answer: If each location site should have a photo for each of the 6 directions, we should expect that the bar plot will show even bars. The Up and Down direction showed up the most, while West had the least frequency. This could be an issue when interpreting site data as we are missing photos for different sites which could be valuable when figuring out what land covers should be present. We could possibly fix this by collecting more site data and to get a higher frequency of photos.

Lets look at the "Up" direction specifically. We can use the same filter() function from before to specify the direction we want.

```
Up <- filter(zooclean, direction == "Up")
```

Lets create a plot that shows what land cover elements are shown in the up direction and how many are present.

```
Y <- data.frame(Up[,16:24]) #Here we are creating a data frame called Y with columns 16 through 24 of t
barplot(as.matrix(Y), las = 2) #We use the as.matrix() function in the bar plot to get the multiple var
```



Question Time! When we think of the Up direction of a photo, what do we expect to see? Is it reflected in the graph above? Are there any variables that "might" be a cause for concern?

Answer: When we think of photos in the "Up" direction, we expect to see sky, urban if there are buildings, and Trees that are tall enough. These variables were the most displayed in the graph above with Sky have the highest frequency. Barren, Herbacous, and shrubs might be causes for concern as Barren and Herbaceous refer to dry or grassy land. It is strange to find them in the Up direction. Lastly, Shrubs could be confusing but there might just be very all bushes.

Lets try looking at a specific location! Using the filter function lets find an entry where the longitude = 53.42660, and the latitude = -000.62950.

```
filter(zooclean, longitude == "53.42660" & latitude  == "-000.62950")
```

```
## # A tibble: 1 x 32
##   classification_id user_name   user_id user_ip        workflow_id workflow_name
##               <dbl> <chr>         <dbl> <chr>                <dbl> <chr>
## 1         378243063 pedernelson 1805556 5a012128dd507~       20048 Sort photos ~
## # ... with 26 more variables: workflow_version <dbl>, created_at <chr>,
## #   gold_standard <lgl>, expert <chr>, metadata <chr>, annotations <chr>,
## #   subject_data <chr>, subject_ids <dbl>, structured <dbl>, Sky <dbl>,
## #   Urban <dbl>, Barren <dbl>, Cultivated <dbl>, Trees <dbl>, Herbaceous <dbl>,
## #   Shrubs <dbl>, Water <dbl>, None <dbl>, MUC <chr>, IGBP <chr>,
## #   direction <chr>, longitude <chr>, latitude <chr>,
## #   ‘MUC Classification‘ <chr>, ‘IGBP Classification‘ <chr>, ...
```

Question Time! What land elements are present at this location? What does the MUC Classification and

IGBP Classification say? Do they match? Do you think the classifications match what land elements are present?

Answer: Urban and Herbaceous are present in this entry. The MUC Classification depicts this locations as agriculture and the IGBP Classifications depicts its at Croplands. These classifications match each other, as well as the land cover elements present.

#Part 8: Cleaning Unstructured Data

Now that we have a solid understanding of what clean data should look like lets try the cleaning process from earlier with the unstructured data.

```
zoodirty <- filter(zoo, structured == 0) #First step is done for you.
```

Lets get the annotations cleaned up!

```
zoodirty$Sky <- ifelse(grepl("Sky", zoodirty$annotations), 1, 0)  #Follow this structure to parse the r
zoodirty$Urban <- ifelse(grepl("Urban", zoodirty$annotations), 1, 0)
zoodirty$Barren <- ifelse(grepl("Barren", zoodirty$annotations), 1, 0)
zoodirty$Cultivated <- ifelse(grepl("Cultivated", zoodirty$annotations), 1, 0)
zoodirty$Trees <- ifelse(grepl("Trees", zoodirty$annotations), 1, 0)
zoodirty$Herbaceous <- ifelse(grepl("Herbaceous", zoodirty$annotations), 1, 0)
zoodirty$Shrubs <- ifelse(grepl("Shrubs", zoodirty$annotations), 1, 0)
zoodirty$Water <- ifelse(grepl("Open Water", zoodirty$annotations), 1, 0)
zoodirty$None <- ifelse(grepl("Can't label", zoodirty$annotations), 1, 0)
```

How about the subject data?

```
zoodirty$MUC <- substr(zoodirty$subject_data, 116, 118) #The first  code is done for you. See how in th
zoodirty$IGBP <- substr(zoodirty$subject_data, 120, 125)
zoodirty$direction <- substr(zoodirty$subject_data, 104, 108)
zoodirty$longitude <- substr(zoodirty$subject_data, 46, 54)
zoodirty$latitude <- substr(zoodirty$subject_data, 56, 65)
```

Lets stop here and use the head() function to get a snapshot of the data set.

```
head(zoodirty)
```

```
## # A tibble: 6 x 29
##   classification_id user_name   user_id user_ip        workflow_id workflow_name
##               <dbl> <chr>         <dbl> <chr>                <dbl> <chr>
## 1         373055746 emmarosenau 2383324 a47a524d356d1~       20048 Sort photos ~
## 2         373099550 pedernelson 1805556 5f2d9a87fe4e6~       20048 Sort photos ~
## 3         373099595 pedernelson 1805556 5f2d9a87fe4e6~       20048 Sort photos ~
## 4         373099621 pedernelson 1805556 5f2d9a87fe4e6~       20048 Sort photos ~
## 5         373099630 pedernelson 1805556 5f2d9a87fe4e6~       20048 Sort photos ~
## 6         373099650 pedernelson 1805556 5f2d9a87fe4e6~       20048 Sort photos ~
## # ... with 23 more variables: workflow_version <dbl>, created_at <chr>,
## #   gold_standard <lgl>, expert <chr>, metadata <chr>, annotations <chr>,
## #   subject_data <chr>, subject_ids <dbl>, structured <dbl>, Sky <dbl>,
## #   Urban <dbl>, Barren <dbl>, Cultivated <dbl>, Trees <dbl>, Herbaceous <dbl>,
## #   Shrubs <dbl>, Water <dbl>, None <dbl>, MUC <chr>, IGBP <chr>,
## #   direction <chr>, longitude <chr>, latitude <chr>
```

Question Time! What codes worked and what didn't? If any part did not work, why do you think that is?

Answer: The codes for the annotations worked because they are structured the same. We can see in the data table snapshot above that the land cover elements parsed correctly. The subject data codes did not work. Because the subject data is not structured the same, we got random values for the variables we are parsing for.

They say data is messy. Why do you think structured naming conventions are important? How can it make our jobs as data scientists easier?

Answer: Structured naming conventions are important because it allows us as data scientists to better organize and extract information from the data. This makes our jobs easier because we can spend less time cleaning and extract information that can be useful for the future projects.