# Cleaning Data

## Cleaning Data In R

GitHub Repository with this demonstration: https://github.com/unmrds/R-data-cleaning

Zipfile Download: https://github.com/unmrds/R-data-cleaning/archive/master.zip

PDF Version of this Notebook: https://github.com/unmrds/R-data-cleaning/raw/master/code/cleaning_data.pdf

Shared RStudioCloud version that you can copy into your account to work with: https://rstudio.cloud/project/3081275

Check out the **Preflight Check** to see if you have the needed R libraries installed run the `package.check.R` script in the top directory of this R project. This script will check to see if the packages are installed, and if they are not will install them. The script finishes with a listing of the currently installed packages so you can verify that the packages we need are installed. Check out the README.md file in the workshop repository for full setup instructions.

When planning a data analysis the first step, and often most time consuming, is the acquisition and processing of the data into a form that can be used in the analytic procedures you intend to use. Today we are going to focus on a sequence of steps that generally follow the workflow that you will find yourself going through when bringing data into R to perform an analysis.

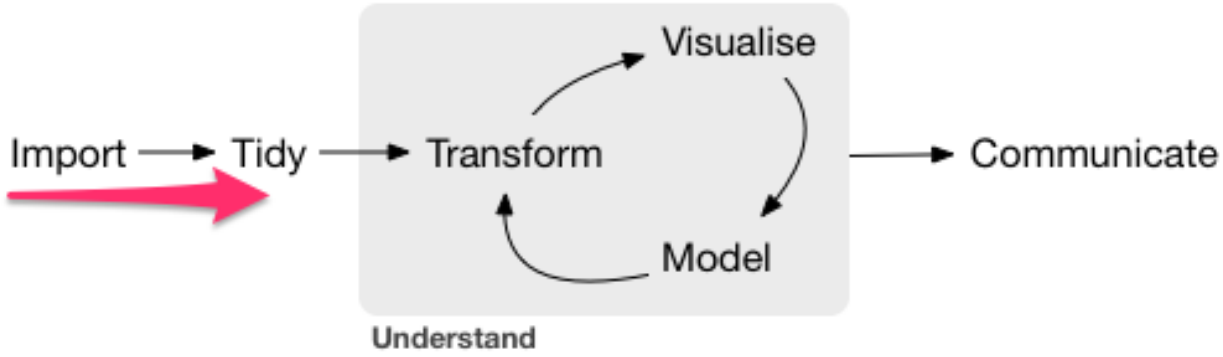



Figure 1: Portion of *R for Data Science*[1] workflow

### Deal with issues that may come up when importing data files

1. Identify and correct structural issues in the source data that prevent clean import into R data structures
2. Check and handle data type errors
3. Check and handle missing data values

### Tuning up the structure of the data to facilitate analysis

4. Split up fields that contain multiple values in a single field

[1] Hadley Wickham & Garrett Grolemund. 2017. *R for Data Science*. O'Reilly. https://r4ds.had.co.nz

5. Check for anomalous values and otherwise explore the data to become familiar with its content and structure.

*Beyond what we will cover today* - continued structural changes and the rest of the exploration, analysis, and communication process.

### Data for today's demonstration

The data for this demonstration are based upon the idigbio_rodents.csv dataset. The data are described as follows in the repository where they are shared:

> The idigbio_rodents.csv dataset is just over 10k records and contains data from natural history collections specimen records representing 5 genera from 4 major US collections, limited to US records. All records are from the Order Rodentia. All the data are mapped to the biodiversity data standard Darwin Core (http://rs.tdwg.org/dwc/terms/).

Collins, Matthew; Paul, Deborah (2017): idigbio_rodents.csv. figshare. Dataset. https://doi.org/10.6084/m9.figshare.5535724.v1

The original data have been modified for use in this demonstration by:

1. Generating new data columns (`latDMS` and `lonDMS`) for latitude and longitude that have sample coordinates presented in Degrees-Minutes-Seconds instead of the originally provided decimal degrees.
2. Generating a column of mixed numeric and text values - `textLatDD`.

The `breaking_data.Rmd` R notebook file programmatically performs the above changes and automatically generated the `learning.csv`.

This is the `../data/learning.csv` file. These newly created columns in addition to some of the originally provided ones will be used to demonstrate a variety of data cleaning steps in R.

An additional file was developed that only includes the first 10 rows of the file (including headers) but introduces a structural error. This file is the `../data/learning_struct.csv` file.

### R libraries used in the demonstration

For this demonstration a combination of R packages that are part of the *Tidyverse* and additional specialized data evaluation packages will be used. The tidyverse collection of packages provide (as described on the project's homepage):

> an opinionated collection of R packages designed for data science. All packages share an underlying design philosophy, grammar, and data structures.

In addition to `tidyverse`, this workshop also uses the `mice`, `VIM`, and `assertr` packages.

There are currently over 14,000 R packages in the Comprehensive R Archive Network (CRAN). While the tidyverse packages provide a useful degree of consistency and internal interoperability it is strongly encouraged to examine the broad collection of R packages when working on a particular analysis problem.

If you need to install any of the needed packages in your environment you can execute the `install.packages("<package name>")` command in the R console. The `package.check.R` script mentioned above will both check to see if the packages are installed, and if needed, install them.

now onto the data processing ...

## 1. Identify and correct structural issues in the source data that prevent clean import into R data structures

R can import a wide variety of *rectangular* data structures: comma-delimited, tab-delimited, excel spreadsheets, fixed-width among the many options. If there are errors in the structure of these files, R import commands

may not be able to parse the lines in the data file preventing import. In these cases the returned error messages *may* provide some clues to where the errors may be found.

**One strategy for identifying potential structural issues in the source file is to try to import the dataset and review any errors that are returned**

Let's try it first with a small file ...

```
########## Working with a CSV file with structural problems ###################

library(tidyverse)
```

```
## -- Attaching packages --------------------------------------- tidyverse 1.3.1 --
```

```
## v ggplot2 3.3.5     v purrr   0.3.4
## v tibble  3.1.5     v dplyr   1.0.7
## v tidyr   1.1.4     v stringr 1.4.0
## v readr   2.0.2     v forcats 0.5.1
```

```
## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

```
# Import the source CSV file that contains a structural flaw
# ? what does that path string in the read_csf function mean?
rawDataStruct <- read_csv("../data/learning_struct.csv",
                    progress = FALSE)
```

```
## New names:
## * `` -> ...1
```

```
## Rows: 9 Columns: 24
```

```
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## chr (20): uuid, institutionCode, collectionCode, recordedBy, countryCode, st...
## dbl  (4): ...1, catalogNumber, year, day
```

```
##
## 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.
```

```
# Display the column definitions for the imported dataset
spec(rawDataStruct)
```

```
## cols(
##   ...1 = col_double(),
##   uuid = col_character(),
##   institutionCode = col_character(),
##   collectionCode = col_character(),
##   catalogNumber = col_double(),
##   recordedBy = col_character(),
##   countryCode = col_character(),
##   stateProvince = col_character(),
##   county = col_character(),
##   decimalLatitude = col_character(),
##   decimalLongitude = col_character(),
##   eventDate = col_character(),
##   year = col_double(),
##   month = col_character(),
```

```
##   day = col_double(),
##   genus = col_character(),
##   specificEpithet = col_character(),
##   scientificName = col_character(),
##   weight = col_character(),
##   length = col_character(),
##   sex = col_character(),
##   latDMS = col_character(),
##   lonDMS = col_character(),
##   textLatDD = col_character()
## )
```

```
# Report the problems that were encountered when the data were imported.
problems(rawDataStruct)
```

```
## Warning: One or more parsing issues, see `problems()` for details
```

```
## # A tibble: 1 x 5
##     row   col expected   actual     file
##   <int> <int> <chr>      <chr>      <chr>
## 1     8    26 24 columns 26 columns /cloud/project/data/learning_struct.csv
```

```
# Display the imported table
rawDataStruct
```

```
## # A tibble: 9 x 24
##    ...1 uuid       institutionCode collectionCode catalogNumber recordedBy
##   <dbl> <chr>      <chr>           <chr>                 <dbl> <chr>
## 1     1 060380ea-~ mvz             mammal specim~       219088 collector(s): a~
## 2     2 0fb17a79-~ mvz             mammal specim~       233524 collector(s): w~
## 3     3 1a69c8ad-~ mvz             mammal specim~       234346 collector(s): w~
## 4     4 1a9932b4-~ mvz             mammal specim~       233951 collector(s): w~
## 5     5 1f3b8aea-~ mvz             mammal specim~       235290 collector(s): w~
## 6     6 203f0531-~ uam             mammal specim~        85106 collector(s): t~
## 7     7 21ceb6f0-~ omnh            mammals               50048 caldwell
## 8     8 23d3f0d9-~ mvz             mammal specim~       216309 collector(s): j~
## 9     9 244bbe9f-~ msb             mammal specim~       294933 collector(s): t~
## # ... with 18 more variables: countryCode <chr>, stateProvince <chr>,
## #   county <chr>, decimalLatitude <chr>, decimalLongitude <chr>,
## #   eventDate <chr>, year <dbl>, month <chr>, day <dbl>, genus <chr>,
## #   specificEpithet <chr>, scientificName <chr>, weight <chr>, length <chr>,
## #   sex <chr>, latDMS <chr>, lonDMS <chr>, textLatDD <chr>
```

The output of the `read_csv` command and of the `problems` function indicate that there was a problem of some sort around row 7 during the import of the CSV:

1 parsing failure. row col expected actual file 7 – 24 columns 26 columns '../data/learning_struct.csv'

Let's take a look at the source data file (in your favorite text editor or within the RStudio interface) and see if we can find the problem...

**What did we find?**

**After the structural problem has been resolved load the full dataset for use in the rest of the workshop**

```
########## Load the full CSV file without the structural error ################
```

```
library(tidyverse)

# Import the source CSV file that does not contain the structural problem highlighted above
rawData <- read_csv("../data/learning.csv",
                    progress = FALSE)
```

```
## New names:
## * `` -> ...1
```

```
## Rows: 10767 Columns: 24
```

```
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## chr  (14): uuid, institutionCode, collectionCode, recordedBy, countryCode, s...
## dbl   (9): ...1, catalogNumber, decimalLatitude, decimalLongitude, year, mon...
## dttm  (1): eventDate
```

```
##
## 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.
```

```
# Display the column definitions for the imported dataset
spec(rawData)
```

```
## cols(
##   ...1 = col_double(),
##   uuid = col_character(),
##   institutionCode = col_character(),
##   collectionCode = col_character(),
##   catalogNumber = col_double(),
##   recordedBy = col_character(),
##   countryCode = col_character(),
##   stateProvince = col_character(),
##   county = col_character(),
##   decimalLatitude = col_double(),
##   decimalLongitude = col_double(),
##   eventDate = col_datetime(format = ""),
##   year = col_double(),
##   month = col_double(),
##   day = col_double(),
##   genus = col_character(),
##   specificEpithet = col_character(),
##   scientificName = col_character(),
##   weight = col_double(),
##   length = col_double(),
##   sex = col_character(),
##   latDMS = col_character(),
##   lonDMS = col_character(),
##   textLatDD = col_character()
## )
```

```
# Report the problems that were encountered when the data were imported.
problems(rawData)
```

```
## # A tibble: 0 x 5
## # ... with 5 variables: row <int>, col <int>, expected <chr>, actual <chr>,
## #   file <chr>
```

```r
# Display the first few rows of the imported table
head(rawData, n=20)
```

```
## # A tibble: 20 x 24
##     ...1 uuid      institutionCode collectionCode catalogNumber recordedBy
##    <dbl> <chr>     <chr>           <chr>                 <dbl> <chr>
##  1     1 060380ea~ mvz             mammal specim~       219088 collector(s): a~
##  2     2 0fb17a79~ mvz             mammal specim~       233524 collector(s): w~
##  3     3 1a69c8ad~ mvz             mammal specim~       234346 collector(s): w~
##  4     4 1a9932b4~ mvz             mammal specim~       233951 collector(s): w~
##  5     5 1f3b8aea~ mvz             mammal specim~       235290 collector(s): w~
##  6     6 203f0531~ uam             mammal specim~        85106 collector(s): t~
##  7     7 21ceb6f0~ omnh            mammals               50048 caldwell, j. p.~
##  8     8 23d3f0d9~ mvz             mammal specim~       216309 collector(s): j~
##  9     9 244bbe9f~ msb             mammal specim~       294933 collector(s): t~
## 10    10 2682aa08~ uam             mammal specim~        50255 collector(s): k~
## 11    11 2e42bc0c~ msb             mammal specim~       268066 collector(s): b~
## 12    12 2e819a65~ msb             mammal specim~       128484 collector(s): c~
## 13    13 304ebf81~ uam             mammal specim~        24581 collector(s): d~
## 14    14 3640f52d~ mvz             mammal specim~       233924 collector(s): w~
## 15    15 37488038~ mvz             mammal specim~       217113 collector(s): j~
## 16    16 4f5079e5~ msb             mammal specim~       294927 collector(s): t~
## 17    17 5b04820b~ mvz             mammal specim~       234317 collector(s): w~
## 18    18 5fd85073~ mvz             mammal specim~       225307 collector(s): j~
## 19    19 6821f9e5~ omnh            mammals                9619 best, t. l.
## 20    20 68662fb9~ msb             mammal specim~       263666 collector(s): m~
## # ... with 18 more variables: countryCode <chr>, stateProvince <chr>,
## #   county <chr>, decimalLatitude <dbl>, decimalLongitude <dbl>,
## #   eventDate <dttm>, year <dbl>, month <dbl>, day <dbl>, genus <chr>,
## #   specificEpithet <chr>, scientificName <chr>, weight <dbl>, length <dbl>,
## #   sex <chr>, latDMS <chr>, lonDMS <chr>, textLatDD <chr>
```

Some questions:

1. How do the data types for the columns from this import process differ from those in the previous subset (at least before we fixed it)? Why do you think this is the case?
2. Where there any errors identified during the import? If no, does this mean that there are no potential problems or issues with the imported data? **Let's take a look**
3. How would you explain the values in the `eventDate` column when compared to the `year`, `month`, and `day` columns? *Hint - what do we know about timezones in this dataset?*
4. What were the different ways in which missing data values were handled?

## 2. Checking and handling data type errors

Depending on the types of data that are encountered in each column of the imported dataset different R import functions will automatically "type" the column (or in R terminology set the "mode" of the column) based on some sample of rows from the source file. In the case of `readr`, the data reading package used by tidyverse, the first 1000 lines of data will be read to determine the data type that should be used for each column. The core R data types are:

- character
- numeric (real or decimal)
- integer
- logical
- complex

These core data types can then be used as the foundation for more complex data types such as *dates*, *times* and *datetimes*.

The core data structures that can be used to organize collections of these data types include:

- vector - a sequence of data values of the same type
- list - a sequence of data values of the same or different types, and structures
- matrix - a vector for which one or more *dimensions* are defined
- data frame (and the "tibble" in the tidyverse) - a structured collection of vectors of the same length
- factors - a factor vector is a set of integer values that are associated with a collection of categorical character values

Let's focus on the `catalogNumber` and `textLatDD` columns in our sample datasets.

```
########## Handling data type errors on import ################################

## Take a look at the types and content of a couple of columns
spec(rawData)
```

```
## cols(
##   ...1 = col_double(),
##   uuid = col_character(),
##   institutionCode = col_character(),
##   collectionCode = col_character(),
##   catalogNumber = col_double(),
##   recordedBy = col_character(),
##   countryCode = col_character(),
##   stateProvince = col_character(),
##   county = col_character(),
##   decimalLatitude = col_double(),
##   decimalLongitude = col_double(),
##   eventDate = col_datetime(format = ""),
##   year = col_double(),
##   month = col_double(),
##   day = col_double(),
##   genus = col_character(),
##   specificEpithet = col_character(),
##   scientificName = col_character(),
##   weight = col_double(),
##   length = col_double(),
##   sex = col_character(),
##   latDMS = col_character(),
##   lonDMS = col_character(),
##   textLatDD = col_character()
## )
```

```
rawData %>%
  select(catalogNumber,textLatDD) %>%       # let's just look at the "catalogNumber" and "textlatDD" co
  slice_head(n=20)                          # let's just look at the first 20 rows
```

```
## # A tibble: 20 x 2
##    catalogNumber textLatDD
##            <dbl> <chr>
##  1        219088 37.7609527778
##  2        233524 37.8999569
##  3        234346 37.8999569
##  4        233951 37.8999569
```

```
##  5        235290 37.8999569
##  6         85106 43.2751187
##  7         50048 34.53903
##  8        216309 36.97099
##  9        294933 36.584948
## 10         50255 44.2611111111
## 11        268066 missing
## 12        128484 39.993572
## 13         24581 45.54
## 14        233924 37.8999569
## 15        217113 37.50348
## 16        294927 32.5075
## 17        234317 37.8999569
## 18        225307 36.4775
## 19          9619 36.967
## 20        263666 33.02048198
```

```
## Test the creation of a numLatDD column as a numeric column and
## see what rows were converted to NA
rawData %>%
  mutate(numLatDD = as.numeric(rawData$textLatDD)) %>%      # convert all of the values to numbers if p
  filter(is.na(numLatDD)) %>%                               # select all of the rows in the new numLat
  select(textLatDD, numLatDD) %>%                           # include only the "textLatDD" and "numLatD
  print() %>%                                               # print the resulting rows and columns
  group_by(textLatDD) %>%                                   # aggregate rows by the content of the "tex
  summarize(count = n())                                    # count up the number of values in each gro
```

```
## Warning in mask$eval_all_mutate(quo): NAs introduced by coercion
```

```
## # A tibble: 1,222 x 2
##    textLatDD numLatDD
##    <chr>        <dbl>
##  1 missing         NA
##  2 missing         NA
##  3 missing         NA
##  4 missing         NA
##  5 missing         NA
##  6 missing         NA
##  7 missing         NA
##  8 missing         NA
##  9 missing         NA
## 10 missing         NA
## # ... with 1,212 more rows
```

```
## # A tibble: 1 x 2
##   textLatDD count
##   <chr>     <int>
## 1 missing    1222
```

```
## create a numeric column based on the previously tested conversion of
## the textLatDD column
rawData$numLatDD <- as.numeric(rawData$textLatDD)
```

```
## Warning: NAs introduced by coercion
```

```
head(rawData, n = 20)
```

```
## # A tibble: 20 x 25
##     ...1 uuid      institutionCode collectionCode catalogNumber recordedBy
##    <dbl> <chr>     <chr>           <chr>                 <dbl> <chr>
##  1     1 060380ea~ mvz             mammal specim~       219088 collector(s): a~
##  2     2 0fb17a79~ mvz             mammal specim~       233524 collector(s): w~
##  3     3 1a69c8ad~ mvz             mammal specim~       234346 collector(s): w~
##  4     4 1a9932b4~ mvz             mammal specim~       233951 collector(s): w~
##  5     5 1f3b8aea~ mvz             mammal specim~       235290 collector(s): w~
##  6     6 203f0531~ uam             mammal specim~        85106 collector(s): t~
##  7     7 21ceb6f0~ omnh            mammals               50048 caldwell, j. p.~
##  8     8 23d3f0d9~ mvz             mammal specim~       216309 collector(s): j~
##  9     9 244bbe9f~ msb             mammal specim~       294933 collector(s): t~
## 10    10 2682aa08~ uam             mammal specim~        50255 collector(s): k~
## 11    11 2e42bc0c~ msb             mammal specim~       268066 collector(s): b~
## 12    12 2e819a65~ msb             mammal specim~       128484 collector(s): c~
## 13    13 304ebf81~ uam             mammal specim~        24581 collector(s): d~
## 14    14 3640f52d~ mvz             mammal specim~       233924 collector(s): w~
## 15    15 37488038~ mvz             mammal specim~       217113 collector(s): j~
## 16    16 4f5079e5~ msb             mammal specim~       294927 collector(s): t~
## 17    17 5b04820b~ mvz             mammal specim~       234317 collector(s): w~
## 18    18 5fd85073~ mvz             mammal specim~       225307 collector(s): j~
## 19    19 6821f9e5~ omnh            mammals                9619 best, t. l.
## 20    20 68662fb9~ msb             mammal specim~       263666 collector(s): m~
## # ... with 19 more variables: countryCode <chr>, stateProvince <chr>,
## #   county <chr>, decimalLatitude <dbl>, decimalLongitude <dbl>,
## #   eventDate <dttm>, year <dbl>, month <dbl>, day <dbl>, genus <chr>,
## #   specificEpithet <chr>, scientificName <chr>, weight <dbl>, length <dbl>,
## #   sex <chr>, latDMS <chr>, lonDMS <chr>, textLatDD <chr>, numLatDD <dbl>
```

We can also accomplish a similar outcome by specifying the column type that should be created as part of the import process.

```
## Specify the column data type when importing
rawData2 <- read_csv("../data/learning.csv",
                     col_types = cols(
                        textLatDD = col_double()
                        ),
                     progress = FALSE)
```

```
## New names:
## * `` -> ...1
```

```
# Display the column definitions for the imported dataset
spec(rawData2)
```

```
## cols(
##   ...1 = col_double(),
##   uuid = col_character(),
##   institutionCode = col_character(),
##   collectionCode = col_character(),
##   catalogNumber = col_double(),
##   recordedBy = col_character(),
##   countryCode = col_character(),
##   stateProvince = col_character(),
##   county = col_character(),
##   decimalLatitude = col_double(),
```

```
##   decimalLongitude = col_double(),
##   eventDate = col_datetime(format = ""),
##   year = col_double(),
##   month = col_double(),
##   day = col_double(),
##   genus = col_character(),
##   specificEpithet = col_character(),
##   scientificName = col_character(),
##   weight = col_double(),
##   length = col_double(),
##   sex = col_character(),
##   latDMS = col_character(),
##   lonDMS = col_character(),
##   textLatDD = col_double()
## )
```

```
# Report the problems that were encountered when the data were imported.
problems(rawData2)
```

```
## Warning: One or more parsing issues, see `problems()` for details
```

```
## # A tibble: 1,222 x 5
##      row   col expected actual  file
##    <int> <int> <chr>    <chr>   <chr>
##  1    12    24 a double missing /cloud/project/data/learning.csv
##  2    22    24 a double missing /cloud/project/data/learning.csv
##  3    26    24 a double missing /cloud/project/data/learning.csv
##  4    33    24 a double missing /cloud/project/data/learning.csv
##  5    40    24 a double missing /cloud/project/data/learning.csv
##  6    45    24 a double missing /cloud/project/data/learning.csv
##  7    55    24 a double missing /cloud/project/data/learning.csv
##  8    56    24 a double missing /cloud/project/data/learning.csv
##  9    60    24 a double missing /cloud/project/data/learning.csv
## 10    64    24 a double missing /cloud/project/data/learning.csv
## # ... with 1,212 more rows
```

```
# Display the imported table
head(rawData2, n = 20)
```

```
## # A tibble: 20 x 24
##     ...1 uuid      institutionCode collectionCode catalogNumber recordedBy
##    <dbl> <chr>     <chr>           <chr>                 <dbl> <chr>
##  1     1 060380ea~ mvz             mammal specim~       219088 collector(s): a~
##  2     2 0fb17a79~ mvz             mammal specim~       233524 collector(s): w~
##  3     3 1a69c8ad~ mvz             mammal specim~       234346 collector(s): w~
##  4     4 1a9932b4~ mvz             mammal specim~       233951 collector(s): w~
##  5     5 1f3b8aea~ mvz             mammal specim~       235290 collector(s): w~
##  6     6 203f0531~ uam             mammal specim~        85106 collector(s): t~
##  7     7 21ceb6f0~ omnh            mammals               50048 caldwell, j. p.~
##  8     8 23d3f0d9~ mvz             mammal specim~       216309 collector(s): j~
##  9     9 244bbe9f~ msb             mammal specim~       294933 collector(s): t~
## 10    10 2682aa08~ uam             mammal specim~        50255 collector(s): k~
## 11    11 2e42bc0c~ msb             mammal specim~       268066 collector(s): b~
## 12    12 2e819a65~ msb             mammal specim~       128484 collector(s): c~
## 13    13 304ebf81~ uam             mammal specim~        24581 collector(s): d~
## 14    14 3640f52d~ mvz             mammal specim~       233924 collector(s): w~
```

```
## 15    15 37488038~ mvz            mammal specim~        217113 collector(s): j~
## 16    16 4f5079e5~ msb            mammal specim~        294927 collector(s): t~
## 17    17 5b04820b~ mvz            mammal specim~        234317 collector(s): w~
## 18    18 5fd85073~ mvz            mammal specim~        225307 collector(s): j~
## 19    19 6821f9e5~ omnh           mammals                 9619 best, t. l.
## 20    20 68662fb9~ msb            mammal specim~        263666 collector(s): m~
## # ... with 18 more variables: countryCode <chr>, stateProvince <chr>,
## #   county <chr>, decimalLatitude <dbl>, decimalLongitude <dbl>,
## #   eventDate <dttm>, year <dbl>, month <dbl>, day <dbl>, genus <chr>,
## #   specificEpithet <chr>, scientificName <chr>, weight <dbl>, length <dbl>,
## #   sex <chr>, latDMS <chr>, lonDMS <chr>, textLatDD <dbl>
```

```
## Convert the catalogNumberTxt column to a character column and see what
## the result is
rawData %>%
  mutate(catalogNumberTxt = as.character(catalogNumber)) %>%
  filter(is.na(catalogNumberTxt))
```

```
## # A tibble: 0 x 26
## # ... with 26 variables: ...1 <dbl>, uuid <chr>, institutionCode <chr>,
## #   collectionCode <chr>, catalogNumber <dbl>, recordedBy <chr>,
## #   countryCode <chr>, stateProvince <chr>, county <chr>,
## #   decimalLatitude <dbl>, decimalLongitude <dbl>, eventDate <dttm>,
## #   year <dbl>, month <dbl>, day <dbl>, genus <chr>, specificEpithet <chr>,
## #   scientificName <chr>, weight <dbl>, length <dbl>, sex <chr>, latDMS <chr>,
## #   lonDMS <chr>, textLatDD <chr>, numLatDD <dbl>, catalogNumberTxt <chr>
```

What do you think would be more forgiving (i.e. less error prone), conversion from character to numeric values or the reverse?

## 3. Check and handle missing values

It is important to understand the potential impact that missing data will have on your analysis. As we've already seen the import process may automatically produce missing data values in your analysis dataframe (or *tibble* in the context of tidyverse based processes). Some functions enable you to efficiently visualize the patterns of missing values in your dataset - allowing for the analysis of large datasets that otherwise would not be feasible to review manually.

```
########## Check and handle missing values ####################################

## Manually by printing out sum (FALSE = 0, TRUE = 1) of is.na test results
# remember that the paste() function just lets you concatenate pieces of text (or values that can be co
paste("decimalLatitude: number of NA values",
      sum(is.na(rawData$decimalLatitude)),
      sep = " ")
```

```
## [1] "decimalLatitude: number of NA values 1222"
```

```
paste("decimalLongitude: number of NA values",
      sum(is.na(rawData$decimalLongitude)),
      sep = " ")
```

```
## [1] "decimalLongitude: number of NA values 1222"
```

```
paste("weight: number of NA values",
      sum(is.na(rawData$weight)),
      sep = " ")
```

```
## [1] "weight: number of NA values 0"
```

```
paste("length: number of NA values",
      sum(is.na(rawData$length)),
      sep = " ")
```

```
## [1] "length: number of NA values 0"
```

```
paste("sex: number of NA values",
      sum(is.na(rawData$sex)),
      sep = " ")
```

```
## [1] "sex: number of NA values 1446"
```

```
paste("latDMS: number of NA values",
      sum(is.na(rawData$latDMS)),
      sep = " ")
```

```
## [1] "latDMS: number of NA values 0"
```

```
paste("lonDMS: number of NA values",
      sum(is.na(rawData$lonDMS)),
      sep = " ")
```

```
## [1] "lonDMS: number of NA values 0"
```

```
paste("textLatDD: number of NA values",
      sum(is.na(rawData$textLatDD)),
      sep = " ")
```

```
## [1] "textLatDD: number of NA values 0"
```

```
paste("numLatDD: number of NA values",
      sum(is.na(rawData$numLatDD)),
      sep = " ")
```

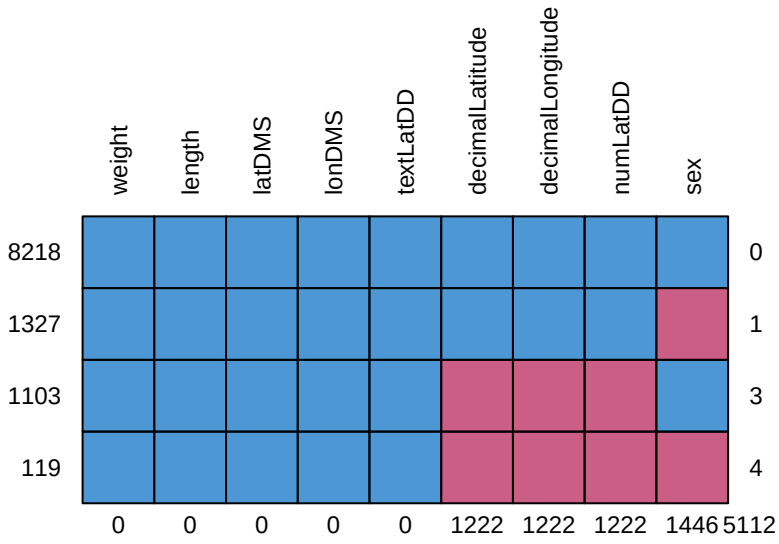
```
## [1] "numLatDD: number of NA values 1222"
```

In this analysis we will be using the `md.pattern` function that is part of the `mice` package, and the `aggr` function which is part of the `VIM` package. If you haven't already installed the `mice` and `VIM` packages you can do so by executing the `install.packages("mice")` and `install.packages("VIM")` commands.

```
##
## Attaching package: 'mice'
```

```
## The following object is masked from 'package:stats':
##
##     filter
```

```
## The following objects are masked from 'package:base':
##
##     cbind, rbind
```




```
##      weight length latDMS lonDMS textLatDD decimalLatitude decimalLongitude numLatDD  sex
## 8218      1      1      1      1          1               1                1        1    1    0
## 1327      1      1      1      1          1               1                1        1    0    1
## 1103      1      1      1      1          1               0                0        0    1    3
## 119       1      1      1      1          1               0                0        0    0    4
##           0      0      0      0          0            1222             1222     1222 1446 5112
```

Another method of viewing similar information

```
## View the combinations of NA values across multiple columns with the
## aggr function from the VIM package
library(VIM)
```

```
## Loading required package: colorspace
```

```
## Loading required package: grid
```

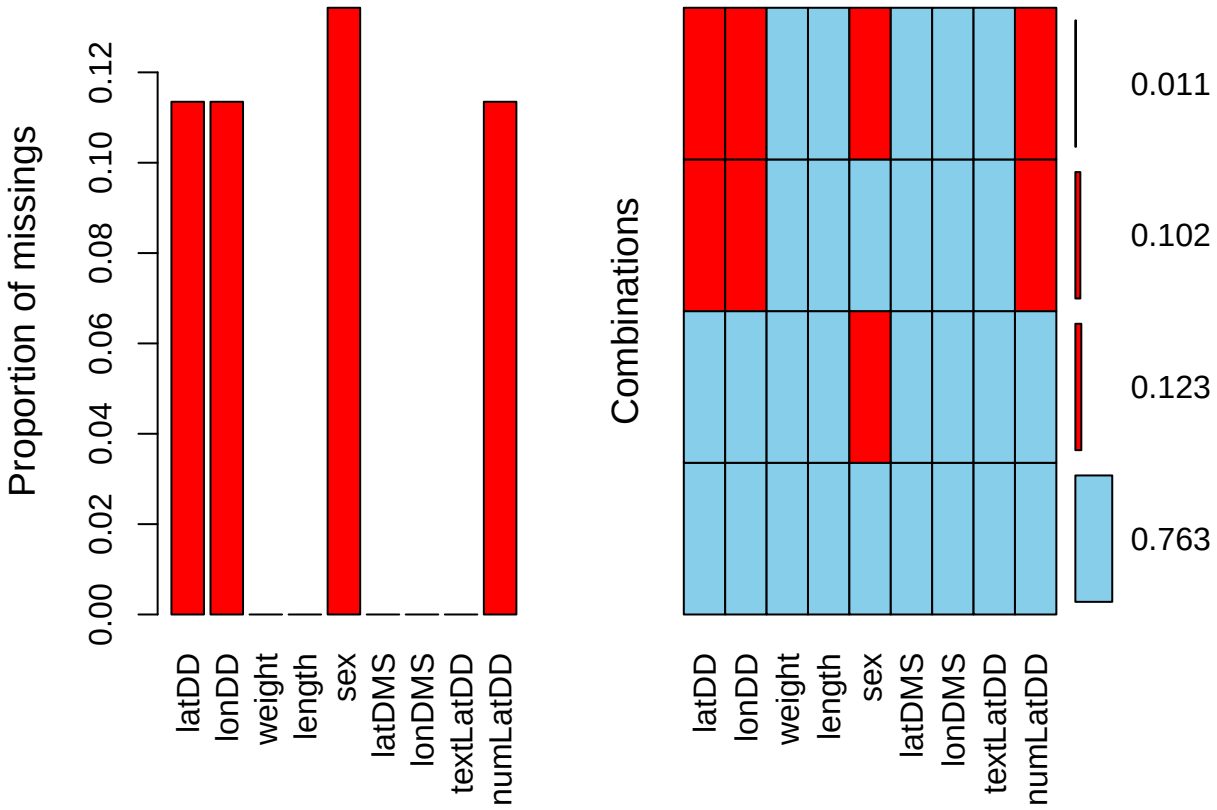
```
## VIM is ready to use.
```

```
## Suggestions and bug-reports can be submitted at: https://github.com/statistikat/VIM/issues
```

```
##
## Attaching package: 'VIM'
```

```
## The following object is masked from 'package:datasets':
##
##     sleep
```

```
rawData %>%
  select(decimalLatitude,
         decimalLongitude,
         weight,
         length,
         sex,
         latDMS,
         lonDMS,
         textLatDD,
         numLatDD) %>%
  rename(latDD = decimalLatitude, lonDD = decimalLongitude) %>%
  aggr(numbers=TRUE)
```




## 4. Multi-value columns

A core principle of having well structured data that is read for analysis is that:

In the context of "Tidy" data that underlie the tools developed as part of the tidyverse packageˆ[Hadley Wickham & Garrett Grolemund. 2017. *R for Data Science*. O'Reilly. - section on Tidy Data https://r4ds.had.co.nz/tidy-data.html

> There are three interrelated rules which make a dataset tidy:
>
> 1. Each variable must have its own column.
> 2. Each observation must have its own row.
> 3. **Each value must have its own cell.**

This issue also relates to the idea of *atomicity* in Codd's definition of *First Normal Form* when *normalizing* a relational database[2]. While we're not going to get into relational data modeling in R here, well structured data allow for the use of relational data models in your analysis independent of a separate database server.

In this example we are going to focus on three columns: `recordedBy`, `latDMS`, and `lonDMS`.

```
########## Handling multi-value columns ########################################

## Take a look at the recordedBy, latDMS, and lonDMS columns
rawData %>%
  select(recordedBy, latDMS, lonDMS)
```

```
## # A tibble: 10,767 x 3
##    recordedBy                                                  latDMS    lonDMS
##    <chr>                                                       <chr>     <chr>
##  1 collector(s): ana lilia trujano álvarez, eric ghilarducci "37°45'3~ "-122° 6~
```

[2]https://en.wikipedia.org/wiki/First_normal_form

```
##  2 collector(s): william z. lidicker jr.                    "37°53'5~ "-123°38~
##  3 collector(s): william z. lidicker jr.                    "37°53'5~ "-123°38~
##  4 collector(s): william z. lidicker jr.                    "37°53'5~ "-123°38~
##  5 collector(s): william z. lidicker jr.                    "37°53'5~ "-123°38~
##  6 collector(s): tom manning; preparator(s): amber baxter   "43°16'3~ "-123°12~
##  7 caldwell, j. p. and vitt, l. j.                          "34°32'2~ "-95° 6'~
##  8 collector(s): james l. patton                            "36°58'1~ "-119°54~
##  9 collector(s): troy l. best; preparator(s): troy l. best  "36°35' ~ "-108° 0~
## 10 collector(s): karl j. martin; preparator(s): paul ollig  "44°15'4~ "-124°25~
## # ... with 10,757 more rows
```

Breaking apart the `recordedBy` column using the `str_match` function from the `stringr` package. This uses *regular expressions* for defining the text patterns that should be found and processed in the process of breaking the column apart into new columns. Regular expressions are an art to themselves and there are many resources for learning their effective use - ranging from one-page cheat-sheets to full length books. Some great resources include:

- Jeffrey E.F. Friedl (2006) *Mastering Regular Expressions*. 3rd Ed. O'Reilly. http://shop.oreilly.com/product/9780596528126.do and for UNM affiliates through our Safari Online Learning subscription: https://learning.oreilly.com/library/view/mastering-regular-expressions/0596528124/ - this is a comprehensive, in-depth treatment of regular expressions from the most simple to most complex - including disucssions of implementations of regular expression support in different programming languages.
- Steven Levithan, Jan Goyvaerts (2012). *Regular Expressions Cookbook*. 2nd Ed. O'Reilly. http://shop.oreilly.com/product/0636920023630.do and for UNM affiliates https://learning.oreilly.com/library/view/regular-expressions-cookbook/9781449327453/ - a good set of regular expression common problems, and their solutions in multiple language implementations.
- Ian Kopacka (2016). *Basic Regular Expressions in R - Cheat Sheet*. Online resource: https://github.com/rstudio/cheatsheets/raw/master/regex.pdf](https://github.com/rstudio/cheatsheets/raw/master/regex.pdf)

The following figure describes the structure of the regular expressions used in the sample code:

```
## Define and use some R regular expressions for extracting text from the
## recordedBy column
collectorExtract <- "^collector\\(s\\):\\s(.*;|.*$)"
preparatorExtract <- "preparator\\(s\\):\\s(.*;|.*$)"

collector_string <- str_match(rawData$recordedBy, collectorExtract)
preparator_string <- str_match(rawData$recordedBy, preparatorExtract)

print(head(collector_string))
```

```
##      [,1]
## [1,] "collector(s): ana lilia trujano álvarez, eric ghilarducci"
## [2,] "collector(s): william z. lidicker jr."
## [3,] "collector(s): william z. lidicker jr."
## [4,] "collector(s): william z. lidicker jr."
## [5,] "collector(s): william z. lidicker jr."
## [6,] "collector(s): tom manning;"
##      [,2]
## [1,] "ana lilia trujano álvarez, eric ghilarducci"
## [2,] "william z. lidicker jr."
## [3,] "william z. lidicker jr."
## [4,] "william z. lidicker jr."
## [5,] "william z. lidicker jr."
```

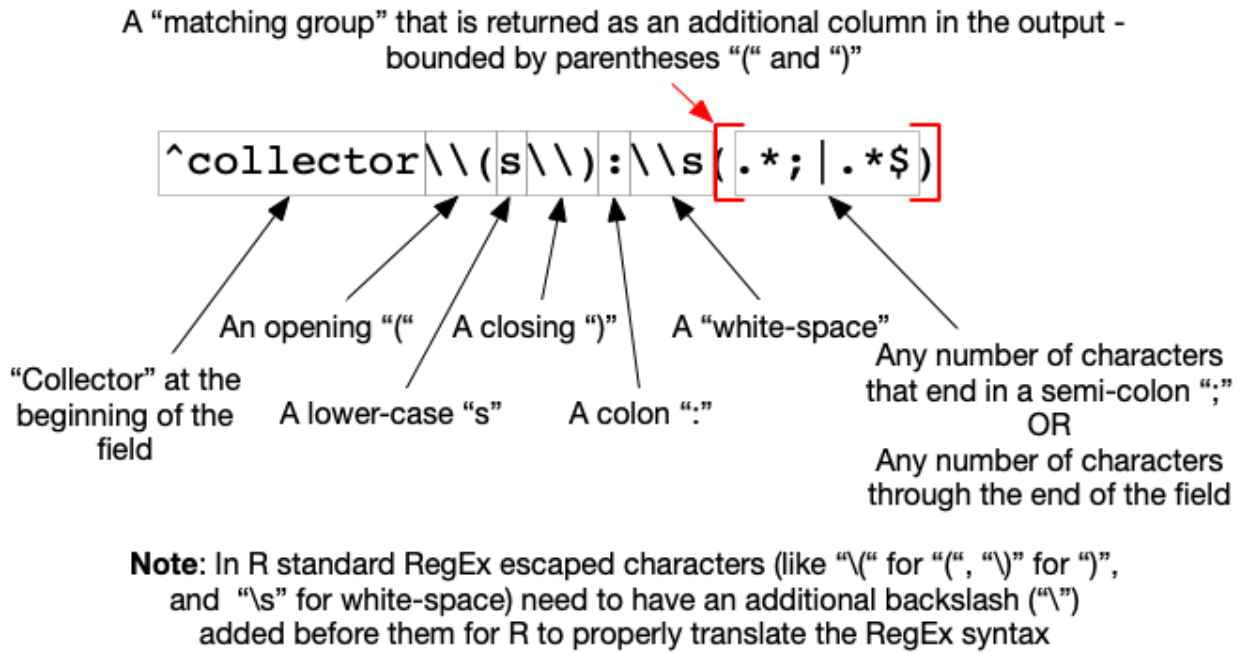



Figure 2: Description of the regular expression used to extract targeted substrings from the combined “recordedBy” field

```
## [6,] "tom manning;"
```

```
print(head(preparator_string))
```

```
##      [,1]                          [,2]
## [1,] NA                            NA
## [2,] NA                            NA
## [3,] NA                            NA
## [4,] NA                            NA
## [5,] NA                            NA
## [6,] "preparator(s): amber baxter" "amber baxter"
```

```
rawData$collectors <- collector_string[,2]
rawData$preparators <- preparator_string[,2]

# check the first ten rows to see what the output looks like
head(rawData, n=10) %>%
  select(recordedBy, collectors, preparators)
```

```
## # A tibble: 10 x 3
##    recordedBy                          collectors                  preparators
##    <chr>                               <chr>                       <chr>
##  1 collector(s): ana lilia trujano álv~ ana lilia trujano álvarez, ~ <NA>
##  2 collector(s): william z. lidicker j~ william z. lidicker jr.      <NA>
##  3 collector(s): william z. lidicker j~ william z. lidicker jr.      <NA>
##  4 collector(s): william z. lidicker j~ william z. lidicker jr.      <NA>
##  5 collector(s): william z. lidicker j~ william z. lidicker jr.      <NA>
##  6 collector(s): tom manning; preparat~ tom manning;                amber baxt~
##  7 caldwell, j. p. and vitt, l. j.      <NA>                        <NA>
##  8 collector(s): james l. patton        james l. patton             <NA>
```

```
##  9 collector(s): troy l. best; prepara~ troy l. best;                troy l. be~
## 10 collector(s): karl j. martin; prepa~ karl j. martin;              paul ollig
```

*What would the next logical step in the process be for cleaning up the* ***recordedBy*** *or newly generated columns?*

Breaking apart the `latDMS` and `lonDMS` columns into their constituent parts

```
## Define a regular expression and use it to extract pieces from a DMS string
dmsExtract <- "\\s*(-*[:digit:]+)°\\s*([:digit:]+)\\'\\s*([:digit:]+\\.*[:digit:]*)"

latSubstrings <- str_match(rawData$latDMS, dmsExtract)
print(head(latSubstrings))
```

```
##      [,1]           [,2] [,3] [,4]
## [1,] "37°45'39.430" "37" "45" "39.430"
## [2,] "37°53'59.845" "37" "53" "59.845"
## [3,] "37°53'59.845" "37" "53" "59.845"
## [4,] "37°53'59.845" "37" "53" "59.845"
## [5,] "37°53'59.845" "37" "53" "59.845"
## [6,] "43°16'30.427" "43" "16" "30.427"
```

```
rawData$latD <- as.numeric(latSubstrings[,2])
rawData$latM <- as.numeric(latSubstrings[,3])
rawData$latS <- as.numeric(latSubstrings[,4])

lonSubstrings <- str_match(rawData$lonDMS, dmsExtract)
print(head(lonSubstrings))
```

```
##      [,1]            [,2]   [,3] [,4]
## [1,] "-122° 6'48.370" "-122" "6"  "48.370"
## [2,] "-123°38'18.039" "-123" "38" "18.039"
## [3,] "-123°38'18.039" "-123" "38" "18.039"
## [4,] "-123°38'18.039" "-123" "38" "18.039"
## [5,] "-123°38'18.039" "-123" "38" "18.039"
## [6,] "-123°12'32.023" "-123" "12" "32.023"
```

```
rawData$lonD <- as.numeric(lonSubstrings[,2])
rawData$lonM <- as.numeric(lonSubstrings[,3])
rawData$lonS <- as.numeric(lonSubstrings[,4])


head(rawData, n=10) %>%
  select(latDMS, latD, latM, latS,
         lonDMS, lonD, lonM, lonS)
```

```
## # A tibble: 10 x 8
##    latDMS             latD  latM  latS lonDMS              lonD  lonM  lonS
##    <chr>             <dbl> <dbl> <dbl> <chr>              <dbl> <dbl> <dbl>
##  1 "37°45'39.430\"N"    37    45 39.4  "-122° 6'48.370\"E"  -122     6  48.4
##  2 "37°53'59.845\"N"    37    53 59.8  "-123°38'18.039\"E"  -123    38  18.0
##  3 "37°53'59.845\"N"    37    53 59.8  "-123°38'18.039\"E"  -123    38  18.0
##  4 "37°53'59.845\"N"    37    53 59.8  "-123°38'18.039\"E"  -123    38  18.0
##  5 "37°53'59.845\"N"    37    53 59.8  "-123°38'18.039\"E"  -123    38  18.0
##  6 "43°16'30.427\"N"    43    16 30.4  "-123°12'32.023\"E"  -123    12  32.0
##  7 "34°32'20.508\"N"    34    32 20.5  "-95° 6'42.444\"E"    -95     6  42.4
##  8 "36°58'15.564\"N"    36    58 15.6  "-119°54'16.740\"E"  -119    54  16.7
##  9 "36°35' 5.813\"N"    36    35  5.81 "-108° 0'55.757\"E"  -108     0  55.8
```

```
## 10 "44°15'40.000\"N"    44    15 40    "-124°25' 1.000\"E"  -124    25    1
```

## 5. Check value ranges and explore data

As part of the examiniation of these columns we will use the `assertr`. If you need to install the `assertr` package in your enviornment you can execute the `install.packages("assertr")` command.

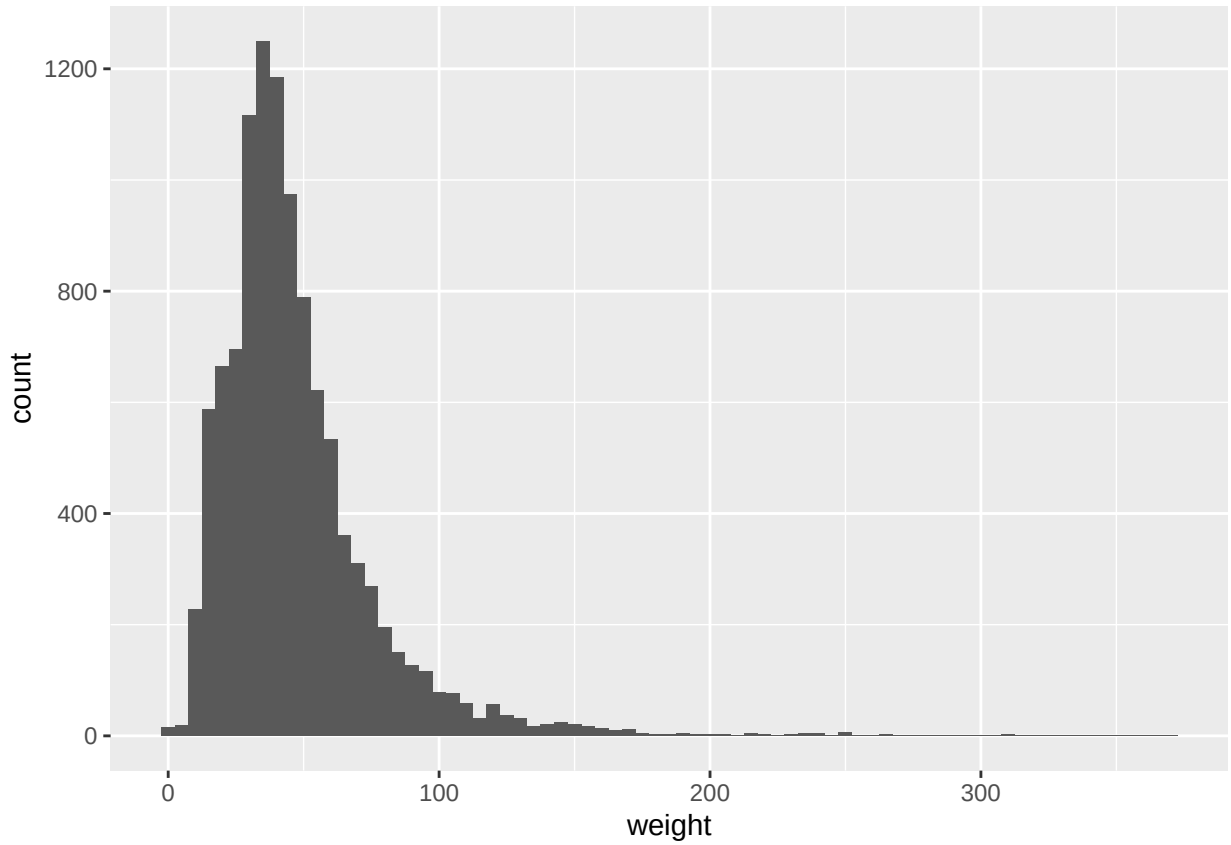
Checking the `weight` and `length` columns.

```
########## Check value ranges and explore data #################################

## assert GitHub repository with instructions and examples:
##      https://github.com/ropensci/assertr
library(assertr)

## you can just run this if you want execution of your workflow to stop if
## any of your tests fail

# rawData %>%
#   chain_start %>%
#   assert(within_bounds(1,Inf), weight) %>% # assert checks individual values
#   assert(within_bounds(1,Inf), length) %>%
#   insist(within_n_sds(3), weight) %>% # insist checks against calculated vals
#   insist(within_n_sds(3), length) %>%
#   chain_end

## you can run this if you want to prevent the errors that are generated
## by your tests from halting execution of your workflow
tryCatch({rawData %>%
  slice_sample(n = 1000) %>%                          # limit the example to 1000 randomly selected row
  chain_start %>%
  assert(within_bounds(1,Inf), weight) %>%            # assert checks individual values
  assert(within_bounds(1,Inf), length) %>%
  insist(within_n_sds(3), weight) %>%                 # insist checks against calculated vals
  insist(within_n_sds(3), length) %>%
  chain_end
}, warning = function(w) {
    paste("A warning was generated: ", w, sep = "")
}, error = function(e) {
    print(e)
}, finally = {
    print("this is the end of the validation check ...")
  }
)
```

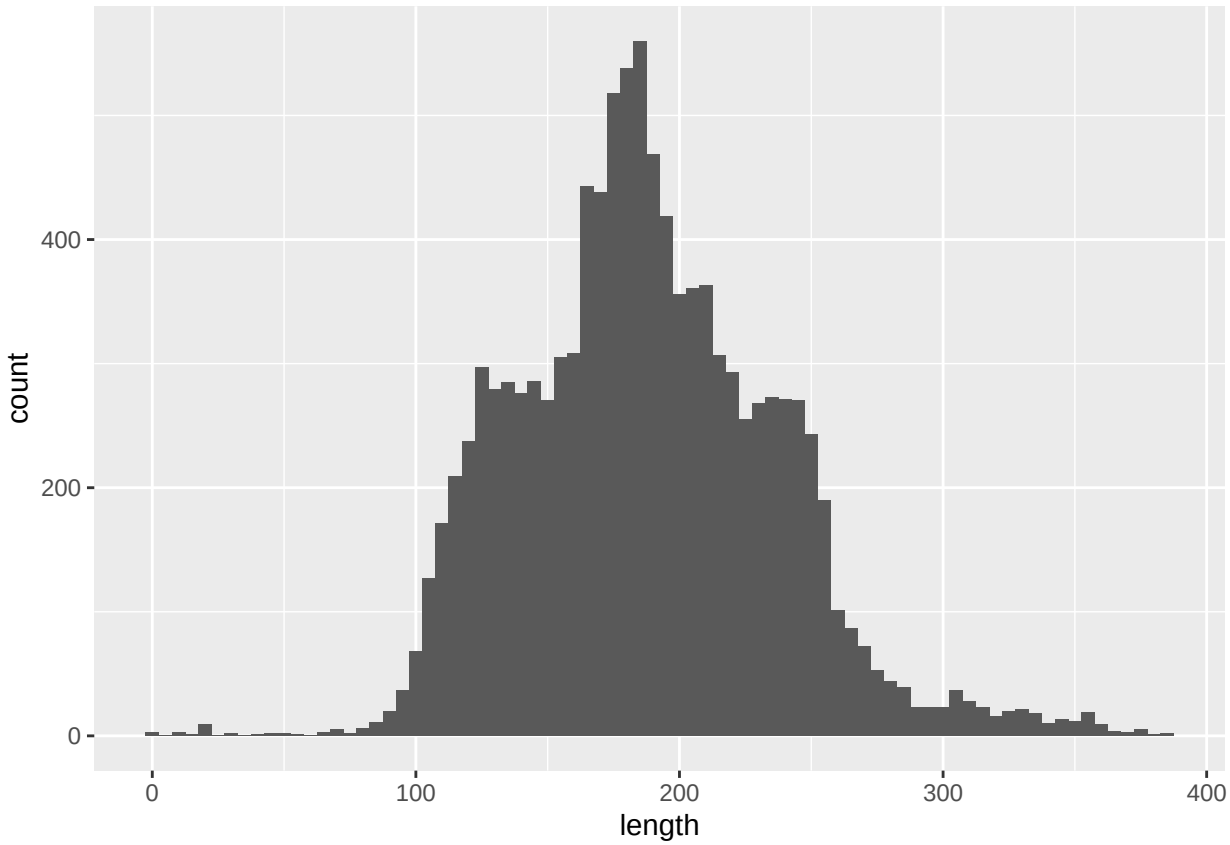
```
## There are 37 errors across 3 verbs:
## -
##      verb redux_fn             predicate column index value
## 1  assert       NA within_bounds(1, Inf) weight   168   0.0
## 2  assert       NA within_bounds(1, Inf) weight   559   0.0
## 3  insist       NA       within_n_sds(3) weight     4 195.0
## 4  insist       NA       within_n_sds(3) weight    97 211.0
## 5  insist       NA       within_n_sds(3) weight   101 264.9
## 6  insist       NA       within_n_sds(3) weight   144 252.0
## 7  insist       NA       within_n_sds(3) weight   187 149.1
## 8  insist       NA       within_n_sds(3) weight   204 156.2
```

```
## 9  insist       NA       within_n_sds(3) weight   297 269.0
## 10 insist       NA       within_n_sds(3) weight   303 310.2
## 11 insist       NA       within_n_sds(3) weight   358 156.0
## 12 insist       NA       within_n_sds(3) weight   402 160.0
## 13 insist       NA       within_n_sds(3) weight   463 172.0
## 14 insist       NA       within_n_sds(3) weight   521 155.0
## 15 insist       NA       within_n_sds(3) weight   612 155.0
## 16 insist       NA       within_n_sds(3) weight   624 157.0
## 17 insist       NA       within_n_sds(3) weight   625 165.0
## 18 insist       NA       within_n_sds(3) weight   662 230.6
## 19 insist       NA       within_n_sds(3) weight   666 190.0
## 20 insist       NA       within_n_sds(3) weight   776 149.0
## 21 insist       NA       within_n_sds(3) weight   834 150.0
## 22 insist       NA       within_n_sds(3) weight   924 146.0
## 23 insist       NA       within_n_sds(3) length    98 335.0
## 24 insist       NA       within_n_sds(3) length   219 373.0
## 25 insist       NA       within_n_sds(3) length   224 335.0
## 26 insist       NA       within_n_sds(3) length   358 358.0
## 27 insist       NA       within_n_sds(3) length   382 345.0
## 28 insist       NA       within_n_sds(3) length   420 337.0
## 29 insist       NA       within_n_sds(3) length   453 334.0
## 30 insist       NA       within_n_sds(3) length   521 334.0
## 31 insist       NA       within_n_sds(3) length   529  28.0
## 32 insist       NA       within_n_sds(3) length   545 334.0
## 33 insist       NA       within_n_sds(3) length   656 361.0
## 34 insist       NA       within_n_sds(3) length   693 333.0
## 35 insist       NA       within_n_sds(3) length   730  45.0
## 36 insist       NA       within_n_sds(3) length   861 362.0
## 37 insist       NA       within_n_sds(3) length   899 332.0
##
## <simpleError: assertr stopped execution>
## [1] "this is the end of the validation check ..."
```

```
## Visual assessment of your data
library(ggplot2)

ggplot(rawData, aes(x=weight)) +
  geom_histogram(binwidth = 5)
```




```
ggplot(rawData, aes(x=length)) +
  geom_histogram(binwidth = 5)
```




## 6. Bringing it all together to produce our analysis dataset

```
##############################################################################
########## Bringing it all together into a single series of commands ##########

# import the data with explicit column definitions for the textLatDD and catalogNumber columns
analysisData <- read_csv("../data/learning.csv",
                     col_types = cols(
                        textLatDD = col_double(),
                        catalogNumber = col_character()
                        ),
                     progress = FALSE)
```

```
## New names:
## * `` -> ...1
```

```
# split up the recordedBy column
collectorExtract <- "^collector\\(s\\):\\s(.*;|.*$)"
preparatorExtract <- "preparator\\(s\\):\\s(.*;|.*$)"

collector_string <- str_match(rawData$recordedBy, collectorExtract)
preparator_string <- str_match(rawData$recordedBy, preparatorExtract)

rawData$collectors <- collector_string[,2]
rawData$preparators <- preparator_string[,2]

# split up the latDMS and lonDMS columns
```

```
dmsExtract <- "\\s*(-*[:digit:]+)°\\s*([:digit:]+)\\'\\s*([:digit:]+\\.*[:digit:]*)"

latSubstrings <- str_match(rawData$latDMS, dmsExtract)

rawData$latD <- as.numeric(latSubstrings[,2])
rawData$latM <- as.numeric(latSubstrings[,3])
rawData$latS <- as.numeric(latSubstrings[,4])

lonSubstrings <- str_match(rawData$lonDMS, dmsExtract)

rawData$lonD <- as.numeric(lonSubstrings[,2])
rawData$lonM <- as.numeric(lonSubstrings[,3])
rawData$lonS <- as.numeric(lonSubstrings[,4])

glimpse(analysisData)
```

```
## Rows: 10,767
## Columns: 24
```

```
## Warning: One or more parsing issues, see `problems()` for details
```

```
## $ ...1             <dbl> 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16~
## $ uuid             <chr> "060380ea-7b06-474e-8d2e-b6e4a8c21e1a", "0fb17a79-a8c~
## $ institutionCode  <chr> "mvz", "mvz", "mvz", "mvz", "mvz", "uam", "omnh", "mv~
## $ collectionCode   <chr> "mammal specimens", "mammal specimens", "mammal speci~
## $ catalogNumber    <chr> "219088", "233524", "234346", "233951", "235290", "85~
## $ recordedBy       <chr> "collector(s): ana lilia trujano álvarez, eric ghilar~
## $ countryCode      <chr> "usa", "usa", "usa", "usa", "usa", "usa", "usa", "usa~
## $ stateProvince    <chr> "california", "california", "california", "california~
## $ county           <chr> "contra costa county", "contra costa county", "contra~
## $ decimalLatitude  <dbl> 37.76095, 37.89996, 37.89996, 37.89996, 37.89996, 43.~
## $ decimalLongitude <dbl> -121.88656, -122.36166, -122.36166, -122.36166, -122.~
## $ eventDate        <dttm> 2005-11-23, 1959-06-21, 1962-11-22, 1960-07-31, 1964~
## $ year             <dbl> 2005, 1959, 1962, 1960, 1964, 1996, 2011, 2005, 1989,~
## $ month            <dbl> 11, 6, 11, 7, 7, 10, 1, 8, 6, 5, 8, 8, 11, 6, 7, 6, 4~
## $ day              <dbl> 22, 20, 21, 30, 3, 22, 17, 6, 4, 19, 10, 12, 30, 17, ~
## $ genus            <chr> "microtus", "microtus", "microtus", "microtus", "micr~
## $ specificEpithet  <chr> "californicus", "californicus", "californicus", "cali~
## $ scientificName   <chr> "microtus californicus californicus", "microtus calif~
## $ weight           <dbl> 30.5, 22.0, 49.0, 33.0, 29.0, 23.5, 24.0, 27.0, 125.0~
## $ length           <dbl> 165, 143, 187, 169, 159, 141, 121, 176, 294, 110, 210~
## $ sex              <chr> "male", "female", "male", "female", "female", "female~
## $ latDMS           <chr> "37°45'39.430\"N", "37°53'59.845\"N", "37°53'59.845\"~
## $ lonDMS           <chr> "-122° 6'48.370\"E", "-123°38'18.039\"E", "-123°38'18~
## $ textLatDD        <dbl> 37.76095, 37.89996, 37.89996, 37.89996, 37.89996, 43.~
```

Export the code from the workshop both as a documented and undocumented R script

```
#library(knitr)
#purl("code/cleaning_data.Rmd", "code/cleaning_data_nodocs.R", documentation = 0)
```