STAT 408 - Week 4: Tidy Data, Data Manipulation, and Processing

February 2, 2018

Tidy Data

Rules for Tidy Data

The concept of tidy data can be attributed to Hadley Wickham and has three principles for organizing data.

1. Each variable must have its own column,
2. Each observation must have its own row, and
3. Each value must have its own cell.

Rules for Tidy Data

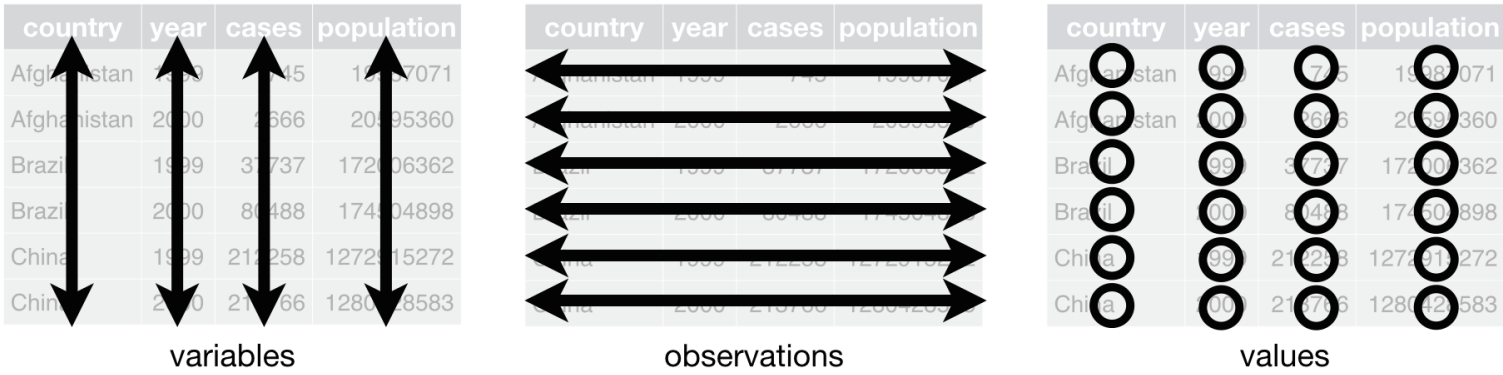

Figure 1: Visual Representation of Tidy Data. Source: R4DS

Why use Tidy Data

Tidy datasets are all alike, but every messy dataset is messy in itw own way. - Hadley Wickham

- Storing data in a consistent way gives familiarity with methods for manipulating data.
- Tidy data structure takes advantage of vectorised operations in R
- Many useful packages `dplyr` `ggplot2` require tidy data.

Baltimore Towing Data

An Overview

The first example focuses on a subset of a dataset that contains information on vehicles towed in Baltimore, MD:

- A larger version of this dataset along with additional descriptions can be found at: `https://data.baltimorecity.gov/Transportation/DOT-Towing/k78j-azhn`.
- The full version of the dataset contains 61,000 rows and 36 columns, where each row corresponds to a vehicle and the columns are information pertaining to the vehicle.
- We will be working with a smaller dataset with approximately 30,000 rows and 5 columns.

The dataset

First read in the data set which is available at: http://www.math.montana.edu/ahoegh/teaching/stat408/datasets/BaltimoreTowing.csv.

```
baltimore.tow <-
  read.csv('http://www.math.montana.edu/ahoegh/teaching/stat408/datasets
           stringsAsFactors = F)
str(baltimore.tow)
```

```
## 'data.frame':    30263 obs. of  5 variables:
##  $ vehicleType       : chr  "Van" "Car" "Car" "Car" ...
##  $ vehicleMake       : chr  "LEXUS" "Mercedes" "Chysler" "Chevrolet" .
##  $ vehicleModel      : chr  "" "" "Cirrus" "Cavalier" ...
##  $ receivingDateTime: chr  "10/24/2010 12:41:00 PM" "04/28/2015 09:27
##  $ totalPaid         : chr  "$322.00" "$130.00" "$280.00" "$1057.00" .
```

Information for a few vehicles


```
kable(head(baltimore.tow, 20))
```

vehicleType	vehicleMake	vehicleModel	receivingDateTime	totalPaid
Van	LEXUS		10/24/2010 12:41:00 PM	$322.00
Car	Mercedes		04/28/2015 09:27:00 AM	$130.00
Car	Chysler	Cirrus	07/23/2015 07:55:00 AM	$280.00
Car	Chevrolet	Cavalier	10/23/2010 11:35:00 AM	$1057.00
Car	Hyundai	Tiburon	10/25/2010 02:49:00 PM	$469.00
SUV	Toyota	RAV4	10/25/2010 11:12:00 AM	$305.00
Car	Bmw	325	10/23/2012 07:50:00 PM	$220.00
Car	Honda	Accord	10/25/2010 02:53:00 PM	$327.00
Car	Ford	Taurus	12/23/2010 04:09:00 AM	$290.00
SUV	Ford	Expo	12/23/2010 02:51:00 PM	$230.00
SUV	Lincoln	Mkx	12/23/2010 01:40:00 PM	$230.00
Car	Geo	Prizm	12/23/2010 06:45:00 AM	$570.00
Car	Kia	Spectra	12/23/2010 03:57:00 AM	$280.00
Car	Nissan		12/23/2010 05:08:00 AM	$280.00
Pick-up Truck	Dodge	Dakota	12/23/2010 02:05:00 PM	$275.00
Motor Cycle (Street Bike)	Honda	CBR600	12/22/2010 11:09:00 PM	$140.00
Car	Chrysler	Sebring	12/23/2010 11:45:00 AM	$220.00
Car	Cadillac	Deville	12/22/2010 09:39:00 PM	$230.00
Car	Nissan	Maxima	12/22/2010 12:41:00 PM	$275.00
Van	Dodge	Caravan	12/23/2010 12:21:00 AM	$140.00

dplyr package

dplyr package

We have a set of tools to complete the analysis, but we are going to use a packaged called `dplyr` to do the data transformation and aggregation for this problem.

There is a nice cheat sheet on the functionality of `dplyr` which a link can be accessed on the course webpage.

PIPING %>%

The pipe operator is an important part of the dplyr toolkit.

- the pipe operator can be read as "THEN"
- the pipe operator allows linking of other dplyr functionality

FILTER()

The `filter()` command is another way to carry out subsetting.

```
baltimore.tow %>% filter(vehicleType == 'Van') %>% head
```

```
##   vehicleType vehicleMake vehicleModel      receivingDateTime totalPa
## 1         Van       LEXUS                10/24/2010 12:41:00 PM   $322.
## 2         Van       Dodge      Caravan 12/23/2010 12:21:00 AM   $140.
## 3         Van     Mercury     Villager 12/22/2010 12:45:00 PM   $275.
## 4         Van    Plymouth      Voyager 05/04/2011 07:57:00 AM   $332.
## 5         Van       Dodge      Caravan 06/20/2012 11:36:00 AM   $220.
## 6         Van    Plymouth      Voyager 06/08/2012 07:00:00 PM   $230.
```

summarize() or summarise()

Summarize is a way to apply functions in a vectorized manner. Below we calculate the average cost of towing vans.

```
baltimore.tow$totalNumeric <-
  as.numeric(substr(baltimore.tow$totalPaid,
                    start = 2, stop=nchar(baltimore.tow)))
baltimore.tow %>% filter(vehicleType == 'Van') %>%
  summarize(mean.cost = mean(totalNumeric))
```

```
##   mean.cost
## 1  294.9024
```

Exercise: group_by()

Now also use the group by procedure to compute the average towing cost for all vehicle types.

Solution: group_by()

Now also use the group by procedure to compute the average towing cost for all vehicle types.

```
baltimore.tow %>%
  group_by(vehicleType) %>%
  summarize(mean.cost = mean(totalNumeric))
```

```
## # A tibble: 22 x 2
##                    vehicleType mean.cost
##                          <chr>     <dbl>
##  1 All terrain - 4 wheel bike  238.0882
##  2                       Boat  208.3333
##  3                        Car  354.0881
##  4           Commercial Truck  618.0000
##  5     Construction Equipment  288.2500
##  6                 Convertible  385.3878
##  7                   Dirt Bike  229.9419
##  8                   Golf Cart  156.2500
##  9                   Mini-Bike  266.5000
## 10  Motor Cycle (Street Bike)  315.4192
## # ... with 12 more rows
```

TIBBLES

- tibbles are a modern rethinking of data.frame() in R.
- tibbles can be coerced into a data.frame with `as.data.frame()`
- tibbles only print first 10 rows when printing and also includes column report type

Data Wrangling

Data Wrangling

As a statistician or more generally a data scientist the ability to manipulate, process, clean, and merge datasets is an essential skill.

- These skills are generally referred to as data wrangling or munging.
- In a data analysis or visualization setting, they will undoubtedly require a majority of your time.
- Wrangling data can be a painful process.
- This lecture will provide some tools and example of organizing data.

Goal 1: Vehicles Towed by Year

The first goal is to determine how many vehicles were towed for each year in the data set.

- Given that the we don't have a column for year and the first observation for receiving date is "10/24/2010 12:41:00 PM".
- Describe the process for obtaining this information.
- What R functions are you familiar with that might be useful here?

substr() function

Consider adding a column for year to the data set. This can be done using `substr()`.

Usage: substr(x, start, stop)

Arguments:

- x, text a character vector.
- start, first integer. The first element to be extracted
- stop, last integer. The last element to be extracted

Exercise: Using the substr() function

Use the substr() function to extract year and create a new variable in R.

```
# baltimore.tow$Year <-
```

Solution: Using the substr() function


```
baltimore.tow$Year <- substr(baltimore.tow$receivingDateTime,7,10)
head(baltimore.tow$Year)
```

```
## [1] "2010" "2015" "2015" "2010" "2010" "2010"
```

```
baltimore.tow$YearNumeric <- as.numeric(baltimore.tow$Year)
head(baltimore.tow$YearNumeric)
```

```
## [1] 2010 2015 2015 2010 2010 2010
```

STRSPLIT() FUNCTION

In many situations, the year could be in a different position so the `substr()` might not work. For example month the date could be coded `4/1/2015` rather than `04/01/2015` So consider, using `strsplit()` instead.

Usage: strsplit(x, split)

Arguments:

- x: character vector, each element of which is to be split. Other inputs, including a factor, will give an error.
- split: character vector (or object which can be coerced to such) containing regular expression(s) (unless fixed = TRUE) to use for splitting.

STRSPLIT() FUNCTION

So, what do we need to split on to isolate year, given that the value follows the form "10/24/2010 12:41:00 PM"?

It is also useful to know that:

- `strsplit` requires a character vector and a factor will return an error and
- the output of this function is a list, where each string in the vector is an element in the list and each element contains a vector of the pieces that have been split.

STRSPLIT() FUNCTION

First split on '/':

```
pieces <- strsplit(as.character(
  baltimore.tow$receivingDateTime[1]), '/')
pieces
```

```
## [[1]]
## [1] "10"                "24"                "2010 12:41:00 PM"
```

Notice the brackets, that means this returns a list

STRSPLIT FUNCTION


```
split <- function(string.in, split.char, position){
  # function to split and retain a portion of output
  # ARGS: string - string so split
  #       split - value to split on
  #       position - position in string to retain
  # RETURNS: character string
  return(strsplit(string.in, split = split.char)[[1]][[position]]
}

temp.date <-
  sapply(baltimore.tow$receivingDateTime,
         split, split.char = '/', position = 3)
as.character(temp.date[1])
```

```
## [1] "2010 12:41:00 PM"
```

Exercise: strsplit function

Now we can extract year from this chunk of code contained in pieces.mat.

```
#baltimore.tow$Year <-
```

Solution: strsplit function

Now we can extract year from this chunk of code contained in pieces.mat.

```
baltimore.tow$Year <- sapply(temp.date, split,
                     split.char = ' ', position = 1)
```

Goal 2. Type of Vehicles Towed by Month

Next we wish to compute how many vehicles were towed in the AM and PM for each type of vehicle.

However, we want to take a close look at the vehicle types in the data set and perhaps create more useful groups.

Unique function - how to group vehicles

First examine the unique types of vehicles in this data set.

```
unique(baltimore.tow$vehicleType)
```

```
##  [1] "Van"                          "Car"
##  [3] "SUV"                          "Pick-up Truck"
##  [5] "Motor Cycle (Street Bike)"    "Dirt Bike"
##  [7] "Commercial Truck"             "Trailer"
##  [9] "Station Wagon"                "Truck"
## [11] "Taxi"                         "Pickup Truck"
## [13] "Convertible"                  "Tractor Trailer"
## [15] "Tow Truck"                    "All terrain - 4 wheel bike"
## [17] "Mini-Bike"                    "Golf Cart"
## [19] "Boat"                         "Tractor"
## [21] "Construction Equipment"       "Sport Utility Vehicle"
```

Grouping

First consider reasonable groups for vehicle types.

1. Cars - (Car, convertible)
2. Large Cars - (SUV, Station Wagon, Sport Utility Vehicle, Van, Taxi)
3. Trucks - (Pick-up Truck, Pickup Truck)
4. Large Trucks - (Truck, Tractor Trailer, Tow Truck, Tractor, Construction Equipment, Commercial Truck)
5. Bikes - (Motor Cycle (Street Bike), Dirt Bike, All terrain - 4 wheel bike, Mini-Bike)
6. Misc (delete) - (Boat, Golf Cart, Trailer)

Messy Data: Grouping

Next examine values in some of these groups, we will just look at the vehicle type of 'Truck'.

```
unique(baltimore.tow$vehicleMake[baltimore.tow$vehicleType == "Truck"])
```

```
##  [1] "GMC"           "Ford"           "Dodge"          "Freightliner
##  [5] "Chevrolet"     "Izuzu"          "Toyota"         "Chevy"
##  [9] "Peterbilt"     "International"  "Kenworth"       "Nissan"
## [13] "Mercedes"      "Isuzu"          "Frightliner"    "Mack"
## [17] "Sterling"      "Internantional" "Peterbelt"      "Pete"
## [21] "Hummer"        "Hino"
```

Note that there are several spelling errors in this data set. How do we combine them?

Messy Data: Data Cleaning

Spelling errors can be addressed, by reassigning vehicles to the correct spelling.

```
baltimore.tow$vehicleMake[baltimore.tow$vehicleMake ==
                            'Peterbelt'] <- 'Peterbilt'
baltimore.tow$vehicleMake[baltimore.tow$vehicleMake ==
                            'Izuzu'] <- 'Isuzu'
baltimore.tow$vehicleMake[baltimore.tow$vehicleMake ==
                            'Frightliner'] <- 'Freightliner'
baltimore.tow$vehicleMake[baltimore.tow$vehicleMake ==
                            'Internantional'] <- 'International'
```

Also note that many of the groupings have mis-classified vehicles, but we will not focus on that yet.

Exercise: Delete Misc. Type Vehicles

First we will delete golf carts, boats, and trailers. There are several ways to do this, consider making a new data frame called balt.tow.small that does not include golf carts, boats, and trailers.

```
balt.tow.small <-
```

Solution: Delete Misc. Type Vehicles

First we will delete golf carts, boats, and trailers.

```
balt.tow.small <- subset(baltimore.tow,!vehicleType %in%
    c('Golf Cart','Trailer','Boat'))
```

Exercise: Create Additional Groups

Now we need to create a variable for the additional groups below.

1. Cars - (Car, convertible)
2. Large Cars - (SUV, Station Wagon, Sport Utility Vehicle, Van, Taxi)
3. Trucks - (Pick-up Truck, Pickup Truck)
4. Large Trucks - (Truck, Tractor Trailer, Tow Truck, Tractor, Construction Equipment, Commercial Truck)
5. Bikes - (Motor Cycle (Street Bike), Dirt Bike, All terrain - 4 wheel bike, Mini-Bike)

Solution: Create Additional Groups

One way to create groups is by creating a new variable

```
balt.tow.small$Group <- ''

balt.tow.small$Group[balt.tow.small$vehicleType %in%
                       c('Car','Convertible')] <- 'Cars'

balt.tow.small$Group[balt.tow.small$vehicleType %in% c('SUV',
 'Station Wagon','Sport Utility Vehicle','Van','Taxi')] <- 'Large Cars'

balt.tow.small$Group[balt.tow.small$vehicleType %in%
                       c('Pick-up Truck','Pickup Truck')] <- 'Trucks'

balt.tow.small$Group[balt.tow.small$vehicleType %in%
    c('Truck','Tractor Trailer','Tow Truck','Tractor',
      'Construction Equipment','Commercial Truck')] <- 'Large Trucks'

balt.tow.small$Group[balt.tow.small$vehicleType %in%
    c('Motor Cycle (Street Bike)','Dirt Bike','Mini-Bike',
      'All terrain - 4 wheel bike')] <- 'Bikes'
```

Ready for Calculations?

First we need to extract the AM/PM tag from the time-date character string. Recall the general structure of this variable is 10/24/2010 12:41:00 PM.

As the tag that we are looking for falls at the end of the string, we can use `nchar()` to find the length of the string.

```
unique(nchar(as.character(balt.tow.small$receivingDateTime)))
```

```
## [1] 22
```

```
balt.tow.small$Time <-
  substr(balt.tow.small$receivingDateTime,21,22)
```

Solution: Aggregate

We could use aggregate, as such:

```
towed.counts <- aggregate(rep(1,nrow(balt.tow.small)),
      by=list(balt.tow.small$Group,balt.tow.small$Time),sum)
colnames(towed.counts)
```

```
## [1] "Group.1" "Group.2" "x"
```

```
colnames(towed.counts) <- c('VehicleGroup','Time','VehicleCount')
```

Solution: Aggregate Results

In this case, `sort()` is not the function we are looking for, rather we use order. Order returns integers for the 'ordering' based on the variables.

```
towed.counts[order(towed.counts$VehicleGroup,towed.counts$Time),]
```

```
##    VehicleGroup Time VehicleCount
## 1         Bikes   AM           87
## 6         Bikes   PM          296
## 2          Cars   AM         8864
## 7          Cars   PM        10811
## 3    Large Cars   AM         3723
## 8    Large Cars   PM         4852
## 4  Large Trucks   AM           82
## 9  Large Trucks   PM          129
## 5        Trucks   AM          606
## 10       Trucks   PM          772
```

Solution: dplyr

We could use aggregate, as such:

```
balt.tow.small %>% group_by(Time, Group) %>% tally()
```

```
## # A tibble: 10 x 3
## # Groups:   Time [?]
##      Time         Group     n
##     <chr>         <chr> <int>
##  1    AM         Bikes    87
##  2    AM          Cars  8864
##  3    AM   Large Cars  3723
##  4    AM Large Trucks    82
##  5    AM        Trucks   606
##  6    PM         Bikes   296
##  7    PM          Cars 10811
##  8    PM   Large Cars  4852
##  9    PM Large Trucks   129
## 10    PM        Trucks   772
```