Oct 25, 2014

Data Manipulation Using R

Cleaning & Summarizing Datasets

ACM DataScience Camp

Packages Useful for this Presentation
dplyr

Ram Narasimhan
@ramnarasimhan

http://goo.gl/DXe1zs

What will we be covering today?

Basics of Data Manipulation

- **What do we mean by Data Manipulation?**
- 4 Reserved Words in R (NA, NaN, Inf & NULL)
- **Data Quality**: Cleaning up data
  - Missing Values | Duplicate Rows| Formatting Columns
- **Subsetting Data**
- “Factors” in R

Data Manipulation Made Intuitive

- **dplyr**
- **The “pipe” operator %>% (‘and then’)**

A note about Built-in datasets

Note

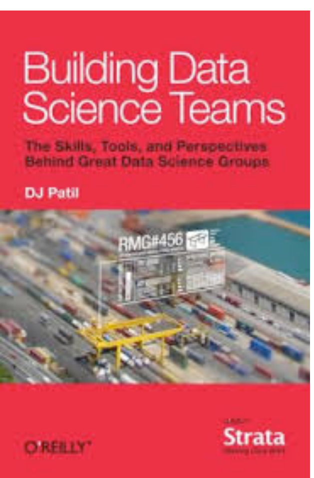
- Many datasets come bundled with R
- Many packages have their own data sets
- To find what you have, type **data()**

```
> data()
#Examples: mtcars, iris, quakes, faithful, airquality,
women
#In ggplot2
> movies; diamonds
```

Important: You won't permanently damage these, so feel free to experiment!

Why Data Carpentry?

Good data scientists understand, in a deep way, that the heavy lifting of cleanup and preparation isn't something that gets in the way of solving the problem – it *is* the problem.



DJ Patil, *Building Data Science Teams*

What are the ways to manipulate data?

- Missing values
- Data Summarization
  - Group By Factors
  - Aggregate
  - Subset / Exclude
  - Bucketing Values
- Rearrange (Shape)
- Merge Datasets

Data Quality

Data Quality

Datasets in real life are never perfect…

How to handle these real-life data quality issues?

- Missing Values
- Duplicate Rows
- Inconsistent Dates
- Impossible values (Negative Sales)
  - Check using if conditions
  - Outlier detection

NA, NULL, Inf & NaN

- `NA     # missing`
- `NULL   # undefined`
- `Inf    # infinite     3/0`
- `NaN    # Not a number   Inf/Inf`

From R Documentation

- **NULL** represents the null object in R: it is a reserved word.
  NULL is often returned by expressions and functions whose values are undefined.
- **NA** is a logical constant of length 1 which contains a missing value indicator.

Dealing with NA's (Unavailable Values)

- To check if any value is NA: **`is.na()`**

Usage:
```
is.na(variable)
is.na(vector)
```

```
> x <- c(3, NA, 4, NA, NA)
> is.na(x[2])
[1] TRUE
> is.na(x)
[1] FALSE  TRUE FALSE  TRUE  TRUE
> !is.na(x)
[1]  TRUE FALSE  TRUE FALSE FALSE
```

Let's use the built-in dataset ***airquality***

```
> is.na(airquality$Ozone)
#TRUE if the value is NA, FALSE otherwise
>!is.na(airquality$Ozone) #note the !(not)
Prints FALSE if any value is NA
```

How to Convert these NA's to 0's?

```
tf <- is.na(airquality$Solar.R) # TRUE FALSE
conditional vector
(TRUE if the values of the Solar.R variable is
NA, FALSE otherwise)
airquality$Solar.R[tf] <- 0
```

Cleaning the data

"iris" is a built-in dataset in R

- Duplicate Rows
  - **`Which rows are duplicated?`**

```
> duplicated(iris)
```

Formatting Columns

- **`as.numeric()`**
- **`as.character()`**

Subsetting
Summarizing
& Aggregation

"Factors" in R

R concept

- Categorical Variables in Statistics
  - Example: "Gender" = {Male, Female}
  - "Meal" = {Breakfast, Lunch, Dinner}
  - Hair Color = {blonde, brown, brunette, red}

  Note: There is no intrinsic ordering to the categories

- In R, Categorical variables are called "Factors"
  - The limited set of values they can take on are called "Levels"

```
class(iris$Species)
iris$Species[1:5] #notice that all Levels are listed
str(mtcars)
#Let's make the "gear" column into a factor
mtcars$gear <- as.factor(mtcars$gear)
str(mtcars$gear)
```

The `subset()` function

Usage: `subset(dataframe, condition)`

- Very easy to use syntax
- One of the most useful commands

```
small_iris <- subset(iris, Sepal.Length > 7)
subset(movies, mpaa=='R')
```

Things to keep in mind

- Note that we don’t need to say **`df$column_name`**
- Note that equals condition is written as **`==`**
- Usually a good idea to verify the number of rows in the smaller data frame (using nrow())

Aggregating using table()

Table counts the #Observations in each level of a factor

```
table(vector)
```

```
table(iris$Species)
table(mtcars$gear)
table(mtcars$cyl)
#put it together to create a summary table
table(mtcars$gear, mtcars$cyl)
```

These resulting tables are sometimes referred to as "frequency tables"

```
#Using "with": note that we don't need to use $
with(movies, table(year))
with(movies, table(length))
with(movies, table(length>200))
```

Data Manipulation - Key Takeaways

1. **Data Quality:** is.na(), na.rm(), is.nan(), is.null()
2. **Table() to get frequencies**
3. **Subset(df, var==value)**

dplyr

Why Use dplyr?

- Very intuitive, once you understand the basics
- Very fast
  - Created with execution times in mind
- Easy for those migrating from the SQL world
- When written well, your code reads like a ‘recipe’
- “Code the way you think”

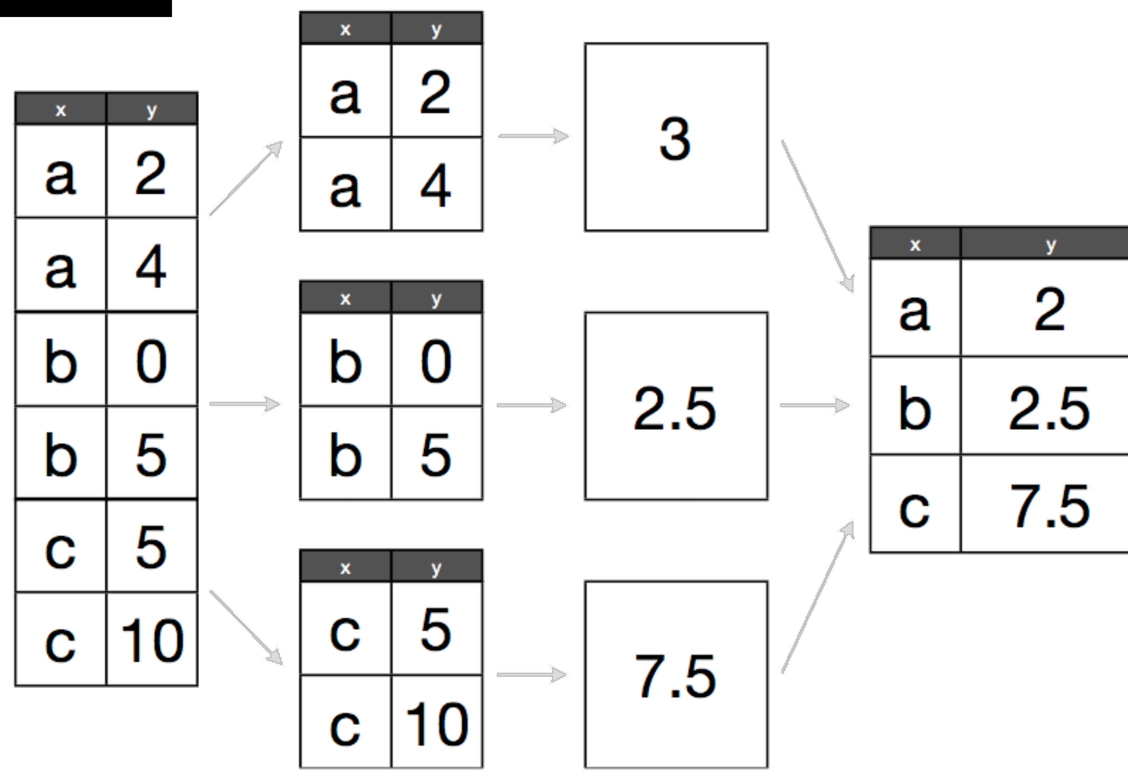
SAC – Split-Apply-Combine

- Let’s understand the SAC idiom

Split up a big dataset

Apply a function to each piece

Combine all the pieces back together



tbl_df() and glimpse()

tbl_df is a 'wrapper' that prettifies a data frame

```
> library(ggplot2)
> glimpse(movies)
> pretty_movies <- tbl_df(movies)
> movies
> pretty_movies
```

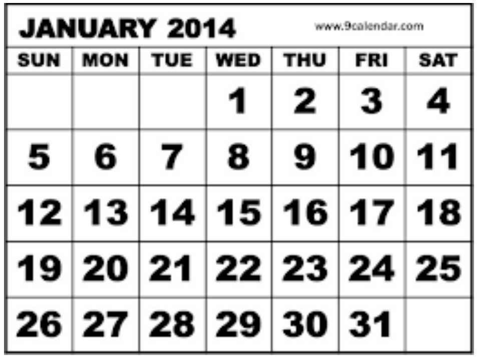
```
> glimpse(movies)
Variables:
$ title       (chr) "$", "$1000 a Touchdown", "$21 a Day Once a Month", "$...
$ year        (int) 1971, 1939, 1941, 1996, 1975, 2000, 2002, 2002, 1987, ...
$ length      (int) 121, 71, 7, 70, 71, 91, 93, 25, 97, 61, 99, 96, 10, 10...
$ budget      (int) NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA, NA...
$ rating      (dbl) 6.4, 6.0, 8.2, 8.2, 3.4, 4.3, 5.3, 6.7, 6.6, 6.0, 5.4,...
$ votes       (int) 348, 20, 5, 6, 17, 45, 200, 24, 18, 51, 23, 53, 44, 11...
$ r1          (dbl) 4.5, 0.0, 0.0, 14.5, 24.5, 4.5, 4.5, 4.5, 4.5, 4.5, 4....
$ r2          (dbl) 4.5, 14.5, 0.0, 0.0, 4.5, 4.5, 0.0, 4.5, 4.5, 0.0, 0.0...
$ r3          (dbl) 4.5, 4.5, 0.0, 0.0, 0.0, 4.5, 4.5, 4.5, 4.5, 4.5, 4.5,...
$ r4          (dbl) 4.5, 24.5, 0.0, 0.0, 14.5, 14.5, 4.5, 4.5, 0.0, 4.5, 1...
$ r5          (dbl) 14.5, 14.5, 0.0, 0.0, 14.5, 14.5, 24.5, 4.5, 0.0, 4.5,...
$ r6          (dbl) 24.5, 14.5, 24.5, 0.0, 4.5, 14.5, 24.5, 14.5, 0.0, 44....
$ r7          (dbl) 24.5, 14.5, 0.0, 0.0, 0.0, 4.5, 14.5, 14.5, 34.5, 14.5...
$ r8          (dbl) 14.5, 4.5, 44.5, 0.0, 0.0, 4.5, 4.5, 14.5, 14.5, 4.5, ...
$ r9          (dbl) 4.5, 4.5, 24.5, 34.5, 0.0, 14.5, 4.5, 4.5, 4.5, 4.5, 1...
$ r10         (dbl) 4.5, 14.5, 24.5, 45.5, 24.5, 14.5, 14.5, 14.5, 24.5, 4...
$ mpaa        (fctr) , , , , , , R, , , , , , , , PG-13, PG-13, , , , , , ...
$ Action      (int) 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, ...
$ Animation   (int) 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...
$ Comedy      (int) 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 0, ...
$ Drama       (int) 1, 0, 0, 0, 0, 1, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, ...
$ Documentary (int) 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, ...
$ Romance     (int) 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...
$ Short       (int) 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1, ...
```

```
> pretty_movies
Source: local data frame [58,788 x 24]

                       title year length budget rating votes   r1   r2  r3   r4
1                          $ 1971    121     NA    6.4   348  4.5  4.5 4.5  4.5
2        $1000 a Touchdown 1939     71     NA    6.0    20  0.0 14.5 4.5 24.5
3   $21 a Day Once a Month 1941      7     NA    8.2     5  0.0  0.0 0.0  0.0
4                  $40,000 1996     70     NA    8.2     6 14.5  0.0 0.0  0.0
5  $50,000 Climax Show, The 1975     71     NA    3.4    17 24.5  4.5 0.0 14.5
6                    $pent 2000     91     NA    4.3    45  4.5  4.5 4.5 14.5
7                  $windle 2002     93     NA    5.3   200  4.5  0.0 4.5  4.5
8                     '15' 2002     25     NA    6.7    24  4.5  4.5 4.5  4.5
9                      '38 1987     97     NA    6.6    18  4.5  4.5 4.5  0.0
10                 '49-'17 1917     61     NA    6.0    51  4.5  0.0 4.5  4.5
..                     ...  ...    ...    ...    ...   ...  ...  ... ...  ...
Variables not shown: r5 (dbl), r6 (dbl), r7 (dbl), r8 (dbl), r9 (dbl), r10
  (dbl), mpaa (fctr), Action (int), Animation (int), Comedy (int), Drama
  (int), Documentary (int), Romance (int), Short (int)
>
```

Understanding the Pipe Operator

- On January first of 2014, a new R package was launched on github
  - maggritr
- A "magic" operator called the PIPE was introduced



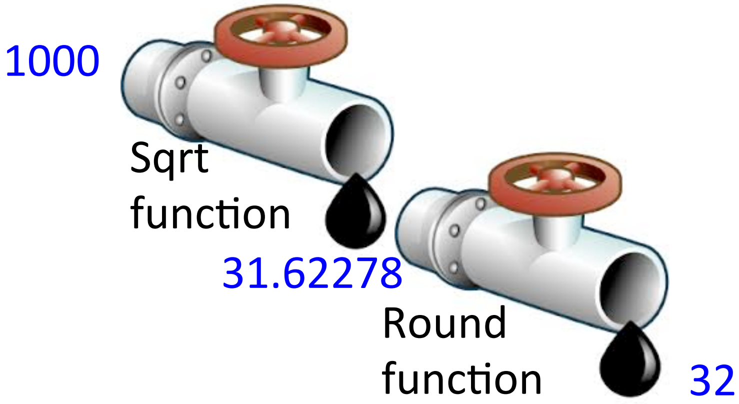
%>%

(Read aloud as: THEN, "AND THEN", "PIPE TO")

```
round(sqrt(1000), 3)

library(magrittr)
1000 %>% sqrt %>% round()
1000 %>% sqrt %>% round(.,3)
```

Take 1000, and then its sqrt
And then round it



dplyr takes advantage of Pipe

- Dplyr takes the %>% operator and uses it to great effect for manipulating data frames
- Works ONLY with Data Frames

A belief that 90% of data manipulation
can be accomplished with
5 basic “verbs”

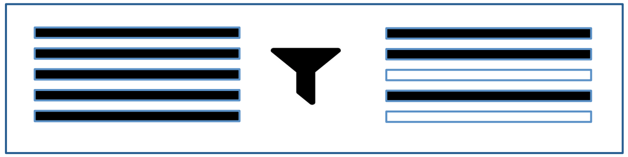
dplyr Package

- The five Basic “Verbs”

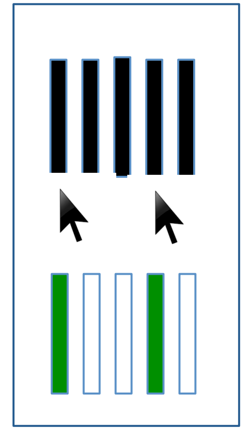
Verbs	What does it do?
`filter()`	Select a subset of ROWS by conditions
`arrange()`	Reorders ROWS in a data frame
`select()`	Select the COLUMNS of interest
`mutate()`	Create new columns based on existing columns (mutations!)
`summarise()`	Aggregate values for each group, reduces to single value

Remember these Verbs (Mnemonics)

- FILTE**R**OWS



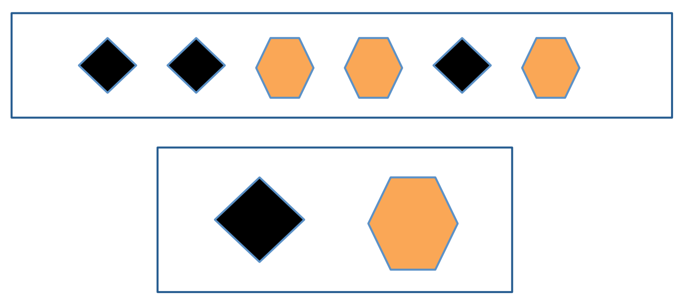
- SELE**CT** Column Types



- Ar**R**ange Rows (SORT)

- Mutate (into something new)

- **Summarize** by Groups



filter()

- Usage: `filter(data, condition)`
  - Returns a subset of rows
  - Multiple conditions can be supplied.
  - They are combined with an AND

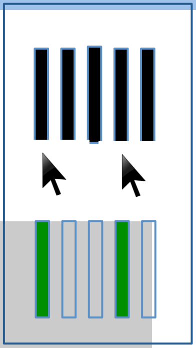
```
movies_with_budgets <- filter(movies_df, !is.na(budget))
filter(movies, Documentary==1)
filter(movies, Documentary==1) %>% nrow()
good_comedies <- filter(movies, rating > 9, Comedy==1)
dim(good_comedies) #171 movies

#' Let us  say we only want highly rated comdies, which a lot
of people have watched, made after year 2000.
movies %>%
  filter(rating >8, Comedy==1, votes > 100, year > 2000)
```

Select()

- Usage:

```
select(data, columns)
```



```
movies_df <- tbl_df(movies)
select(movies_df, title, year, rating) #Just the columns we want to see
select(movies_df, -c(r1:r10)) #we don't want certain columns

#You can also select a range of columns from start:end
select(movies_df, title:votes)   # All the columns from title to votes
select(movies_df, -c(budget, r1:r10, Animation, Documentary, Short, Romance))

select(movies_df, contains("r"))  # Any column that contains 'r' in its name
select(movies_df, ends_with("t")) # All vars ending with "t"

select(movies_df, starts_with("r"))  # Gets all vars staring with "r"
#The above is not quite what we want. We don't want the Romance column
select(movies_df, matches("r[0-9]"))  # Columns that match a regex.
```

arrange()

Usage: `arrange(data, column_to_sort_by)`

- Returns a reordered set of rows
- Multiple inputs are arranged from left-to-right

```
movies_df <- tbl_df(movies)
arrange(movies_df, rating) #but this is not what we want
arrange(movies_df, desc(rating))
#Show the highest ratings first and the latest year…
#Sort by Decreasing Rating and Year
arrange(movies_df, desc(rating), desc(year))
```

What's the difference between these two?

```
arrange(movies_df, desc(rating), desc(year))
arrange(movies_df, desc(year), desc(rating))
```

mutate()

- Usage:

```
mutate(data, new_col = func(oldcolumns)
```

- Creates new columns, that are functions of existing variables

```
mutate(iris, aspect_ratio = Petal.Width/Petal.Length)

movies_with_budgets <- filter(movies_df, !is.na(budget))
mutate(movies_with_budgets, costPerMinute = budget/length) %>%
  select(title, costPerMinute)
```

group_by() & summarize()

```
group_by(data, column_to_group) %>%
summarize(function_of_variable)
```

- Group_by creates groups of data
- Summarize aggregates the data for each group

```
by_rating <- group_by(movies_df, rating)

by_rating %>% summarize(n())

avg_rating_by_year <-
    group_by(movies_df, year) %>%
    summarize(avg_rating = mean(rating))
```

Chaining the verbs together

- Let's put it all together in a logical fashion
- Use a sequence of steps to find the most expensive movie per minute of eventual footage

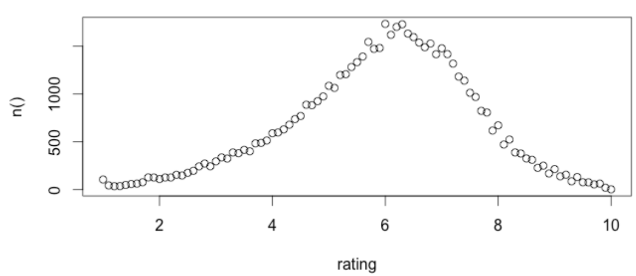
```
producers_nightmare <-
  filter(movies_df, !is.na(budget)) %>%
  mutate(costPerMinute = budget/length) %>%
  arrange(desc(costPerMinute)) %>%
  select(title, costPerMinute)
```

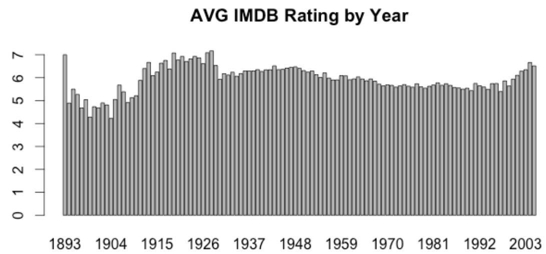
Bonus: Pipe into Plot

- The output of a series of “pipes” can also be fed to a “plot” command

```
movies %>%
  group_by(rating) %>%
  summarize(n()) %>%
  plot() # plots the histogram of movies by Each value of rating

movies %>%
  group_by(year) %>%
  summarise(y=mean(rating)) %>%
  with(barplot(y, names.arg=year, main="AVG IMDB Rating by Year"))
```





References

- Dplyr vignettes: http://cran.rstudio.com/web/packages/dplyr/vignettes/introduction.html
- Kevin Markham's dplyr tutorial
  - http://rpubs.com/justmarkham/dplyr-tutorial
  - His YouTube video (38-minutes)
  - https://www.youtube.com/watch?feature=player_embedded&v=jWjqLW-u3hc
- http://patilv.com/dplyr/
  - Use arrows to move forward and back

Aggregating Data Using "Cut"

What does "cut" do?

- – Bucketing
- – Cuts a continuous variable into groups

- Extremely useful for grouping values

Take the **airquality** Temperature Data and group into buckets

```
range(airquality$Temp)
#First let's cut this vector into 5 groups:
cut(airquality$Temp, 5)
cut(airquality$Temp, 5, labels=FALSE)
#How many data points fall in each of the 5 intervals?
table(cut(airquality$Temp, 5))

Tempbreaks=seq(50,100, by=10)
TempBuckets <- cut(airquality$Temp, breaks=Tempbreaks)
summary(TempBuckets)
```

aggregate()

How many of each species do we have?

Usage: `aggregate(y ~ x, data, FUN)`

```
aggregate(numeric_variable ~ grouping variable, data)
```

How to read this?

“Split the <numeric_variable> by the <grouping variable>”

Split y into groups of x, and apply the function to each group

```
aggregate(Sepal.Length ~ Species, data=iris, FUN='mean')
```

Note the Crucial Difference between the two lines:

```
aggregate(Sepal.Length~Species, data=iris,
FUN='length')
aggregate(Species ~ Sepal.Length, data=iris,
FUN='length') # caution!
```

Note: If you are doing lots of summarizing, the “doBy” package is worth looking into