Exploratory Data Analysis

謝舒凱 Shu-Kai Hsieh

January 6, 2015

Contents


```
# clear R's memory
rm(list=ls())
# where R is currently looking
getwd()
# tell R where to look
setwd("~/Coding/R_lab/Linguistics Data Analysis with R/")
```

1 Introduction

Once you've **loaded** your data and **cleaned** and **transformed** it into a suitable state, you get to start asking questions like "what does it all mean?" The two main tools at your disposal are summary statistics - means and medians, variances, and counts - and plots (or graphs of the data). (Modeling comes later, because you need to understand your data before you can model it properly.)

You can spot some problems just by using summary statistics; other problems are easier to find visually.So the main learnings from this units:

- Summary statistics (based on median and its variants, which are robust to outliers)
- Visualization techniques (in stem-and-leaf, letter values, and bagplots): know how to draw standard plots and manipulate those plots in simple ways.
- First regression model in Resistant line and refined methods in smoothing data and median polish.

2 Essential summary statistics

We've already come across many of the functions for calculating summary statistics, so this section is partly a recap.

We usually use `summary()` to first look at the data.

```
custdata <- read.table('custdata2.tsv', header=T, sep='\t')
summary(custdata)
```

```
##      custid        sex     is.employed         income
##  Min.   :   2068   F:386   Mode :logical   Min.   :    30
##  1st Qu.: 346832   M:524   FALSE:62        1st Qu.: 19000
##  Median : 709044           TRUE :593       Median : 38350
##  Mean   : 700502           NA's :255       Mean   : 57684
##  3rd Qu.:1043725                           3rd Qu.: 70162
##  Max.   :1414286                           Max.   :615000
##
##              marital.stat health.ins
##  Divorced/Separated:146   Mode :logical
##  Married           :469   FALSE:119
##  Never Married     :203   TRUE :791
```

```
##  Widowed           : 92   NA's :0
##
##
##
##                       housing.type recent.move      num.vehicles
##  Homeowner free and clear    :151   Mode :logical   Min.   :0.00
##  Homeowner with mortgage/loan:390   FALSE:768       1st Qu.:1.00
##  Occupied with no rent       : 11   TRUE :111       Median :2.00
##  Rented                      :327   NA's :31        Mean   :1.91
##  NA's                        : 31                   3rd Qu.:2.00
##                                                     Max.   :6.00
##                                                     NA's   :31
##       age              state.of.res
##  Min.   :18.00   California  : 92
##  1st Qu.:39.00   Pennsylvania: 67
##  Median :50.00   New York    : 65
##  Mean   :51.85   Texas       : 51
##  3rd Qu.:64.00   Michigan    : 48
##  Max.   :93.00   Ohio        : 48
##                  (Other)     :539
```

Let's walk through the example taken from Cotton (2013).

- the `obama_vs_mccain` dataset contains the fractions of people voting for Obama and McCain in the 2008 US presidential elections, along with some contextual background information on demographics.

```
load("~/Linguistic.Data.Science/Corpus.Processing.Method.I.2014-5.ntu/week.7/obama_vs_mccain.r
obama <- obama_vs_mccain$Obama
# mean(obama)
# median(obama)
# range(obama)
# var(obama)
# sd(obama)
# quantile(obama); quantile(obama, c(0.3, 0.6, 0.99))
# summary(obama)
```

- The `table()` doesn't make sense for the obama variable (or many numeric variables) since each value is unique. Recall that we can use `cut()` to combine it, so that we can see how many values fall into different bins:

```
table(cut(obama,seq.int(0,100,10)))

##
##   (0,10]  (10,20]  (20,30]  (30,40]  (40,50]  (50,60]  (60,70]  (70,80]
##        0        0        0        8       16       16        9        1
##  (80,90] (90,100]
##        0        1

# plot it
```

- The `col()` calculates correlations between numeric vectors, The `cancor()` (short for 'canonical correlation') provides extra details, and the `cov()` function calculates covariances:

```
with(obama_vs_mccain, cor(Obama, McCain)) # almost perfect negative correlation

## [1] -0.9981189
```

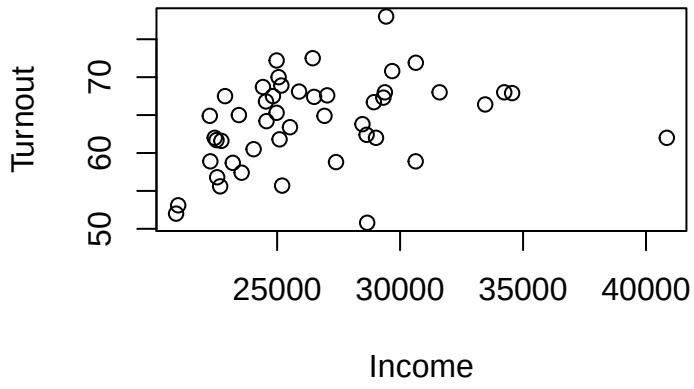
3 Plotting

- Recall that the graph systems in R: `base, lattice, ggplot2`

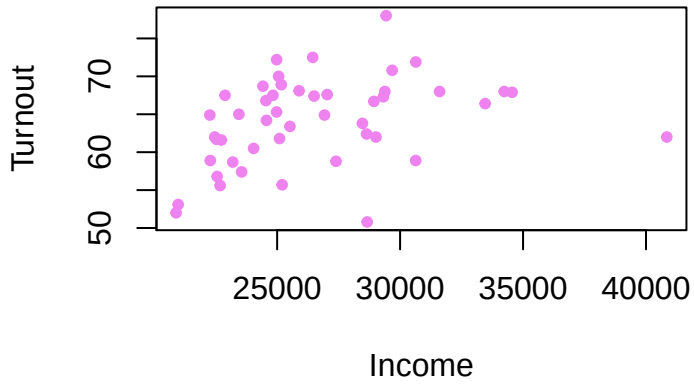
Question: [1] Does voter income affect turnout at the polls? (scatterplot)

Take 1: base Graphics

```
# Although plot will simply ignore missing values,
# for tidiness let's remove the rows with missing Turnout values:

obama_vs_mccain <- obama_vs_mccain[!is.na(obama_vs_mccain$Turnout), ]
# then create a simple scatterplot
with(obama_vs_mccain, plot(Income, Turnout))
```

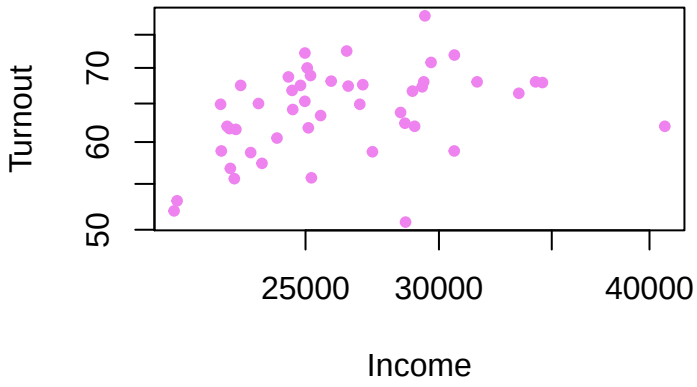



```
#### what's the difference with the following?
#obama_vs_mccain_2 <- na.omit(obama_vs_mccain)
#with(obama_vs_mccain_2, plot(Income, Turnout))

# customize the plot

with(obama_vs_mccain, plot(Income, Turnout, col = "violet", pch = 20))
```



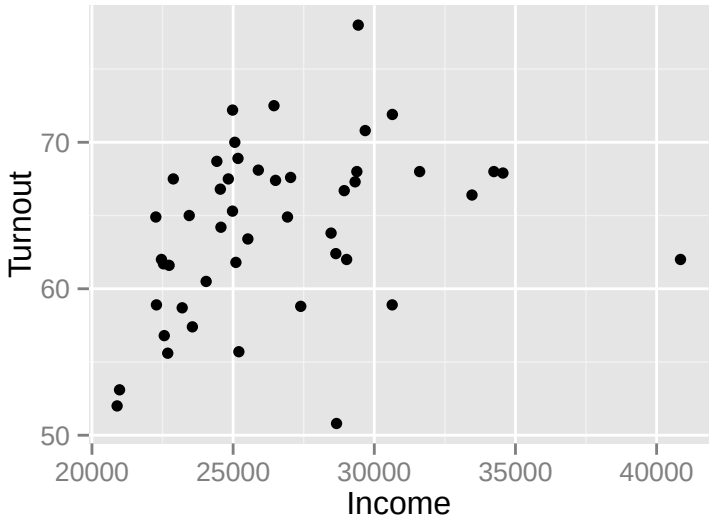

```
# Log scales are possible by setting the log argument.
# log = "x" means use a logarithmic x-scale,
```

```
# log = "y" means use a logarithmic y-scale, and log = "xy" makes both scales logarithmic.

with(obama_vs_mccain, plot(Income, Turnout, col = "violet", pch = 20, log = "xy"))
```

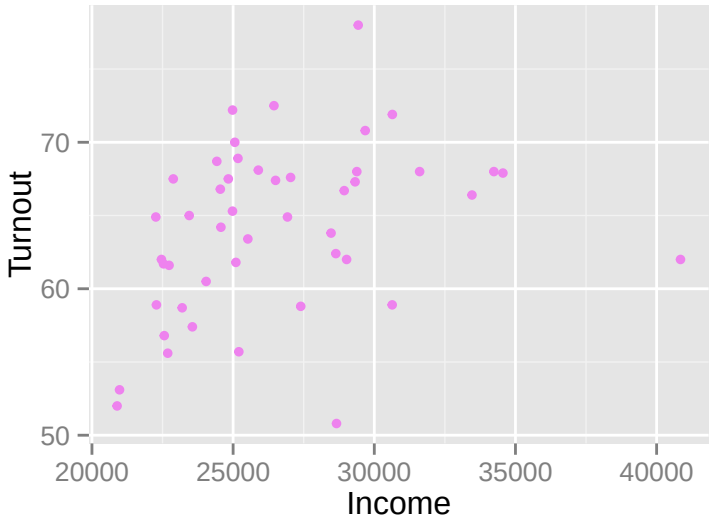


Take 2: ggplot2

```
library(ggplot2)

# A simple scatterplot
ggplot(obama_vs_mccain, aes(Income, Turnout)) +
  geom_point()
```

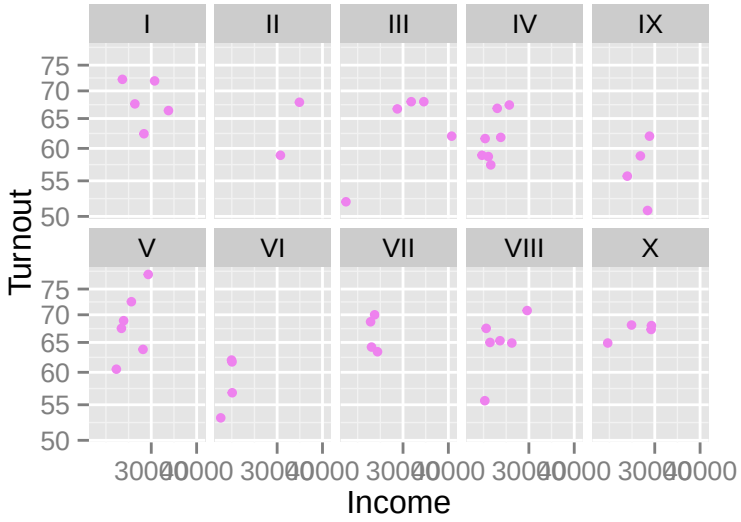



```
ggplot(obama_vs_mccain, aes(Income, Turnout)) +
  geom_point(color = "violet", shape = 20, log ="xy")
```



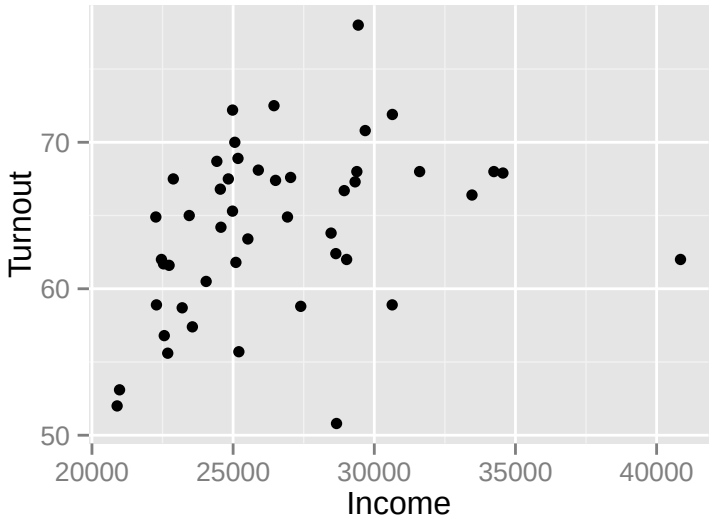

```
# ggplot(obama_vs_mccain, aes(Income, Turnout)) +
#   geom_point(color = "violet", shape = 20) +
#   scale_x_log10(breaks = seq(2e4, 4e4, 1e4)) +
#   scale_y_log10(breaks = seq(50, 75, 5))

# To split the plot into individual panels, we add a facet.
```

```
ggplot(obama_vs_mccain, aes(Income, Turnout)) +
  geom_point(color = "violet", shape = 20) +
  scale_x_log10(breaks = seq(2e4, 4e4, 1e4)) +
  scale_y_log10(breaks = seq(50, 75, 5)) +
  facet_wrap(~ Region, ncol = 5)
```

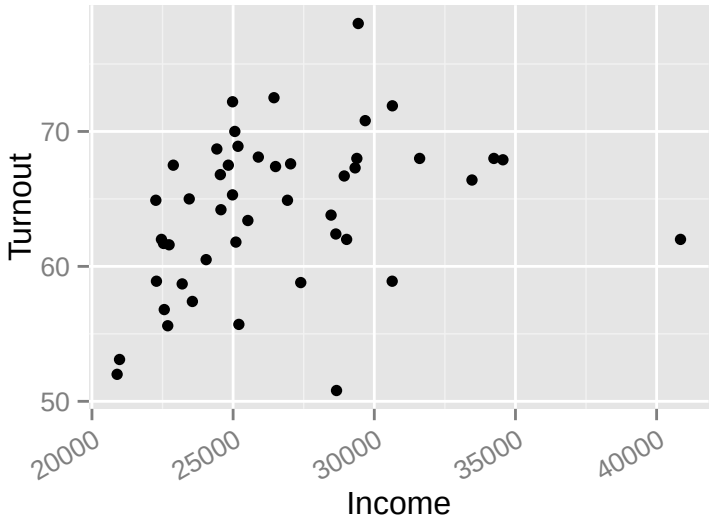


- ggplots can be stored in variables and added to sequentially.

```
#As usual, wrapping the expression in parentheses makes it auto-print:


(gg1 <- ggplot(obama_vs_mccain, aes(Income, Turnout)) +
  geom_point()
)
```

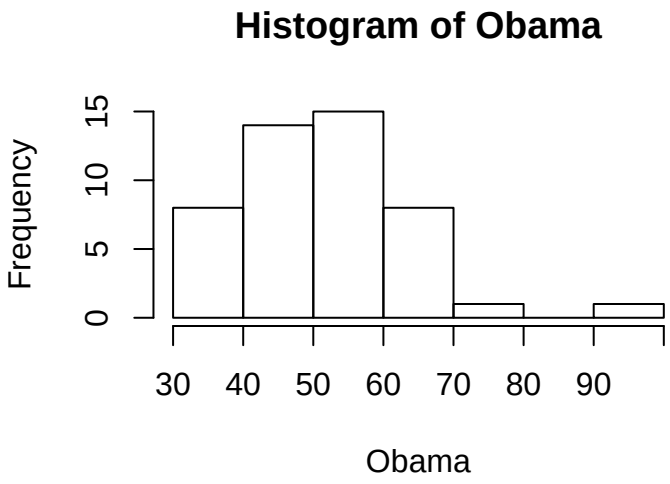



```
(gg2 <- gg1 +
  theme(axis.text.x = element_text(angle = 30, hjust = 1))
)
```

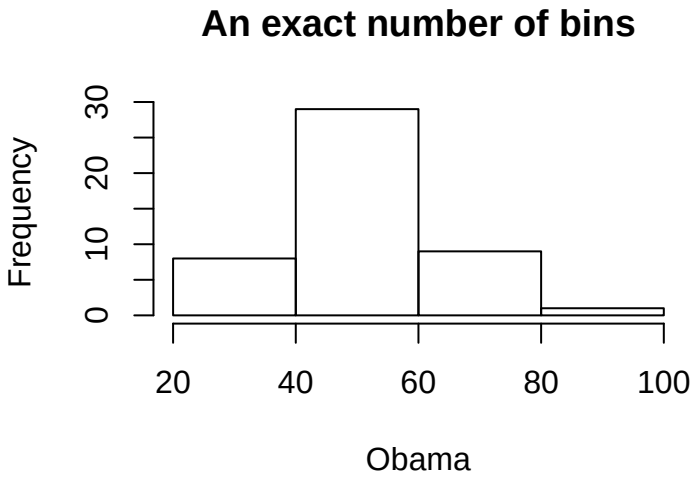


Histogram

```
#the distribution of the percentage of votes for Obama
#(calculated by default by Sturges's algorithM)
with(obama_vs_mccain, hist(Obama))
```

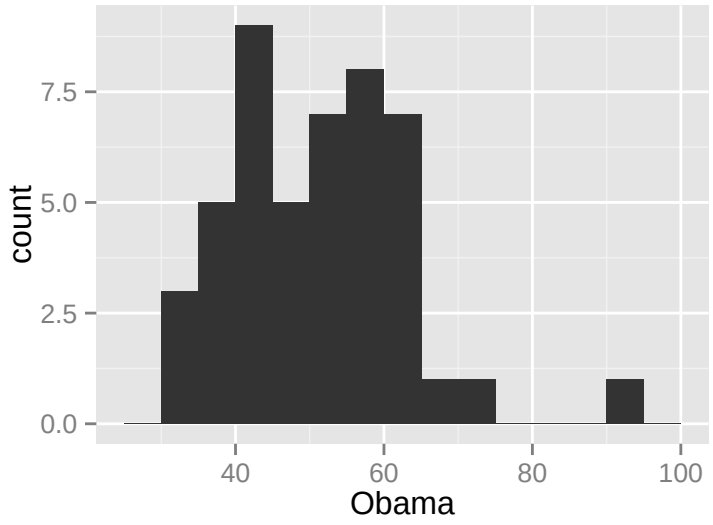



```
# experiment with the width of bins in order to get
# a more complete understanding of the distribution.
with(obama_vs_mccain,
  hist(Obama, 4, main = "An exact number of bins")
)
```

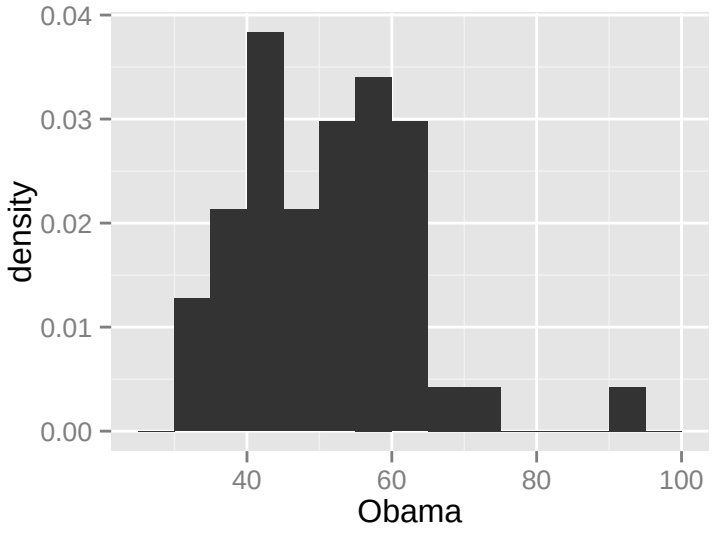



```
# with(obama_vs_mccain,
#  hist(Obama, seq.int(0, 100, 5), main = "A vector of bin edges")
#)
```

```
ggplot(obama_vs_mccain, aes(Obama)) +
  geom_histogram(binwidth = 5)
```




```
# You can choose between counts and densities by
# passing the special names ..count.. or ..density..
# to the y-aesthetic.
ggplot(obama_vs_mccain, aes(Obama, ..density..)) +
  geom_histogram(binwidth = 5)
```

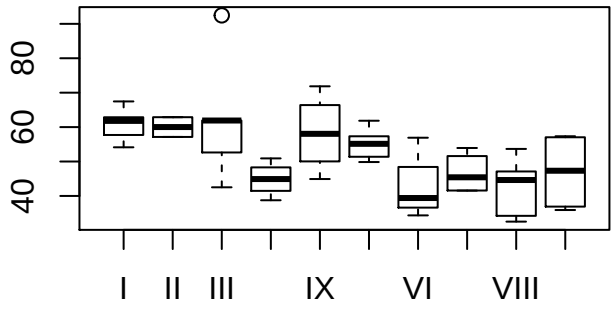


Boxplot

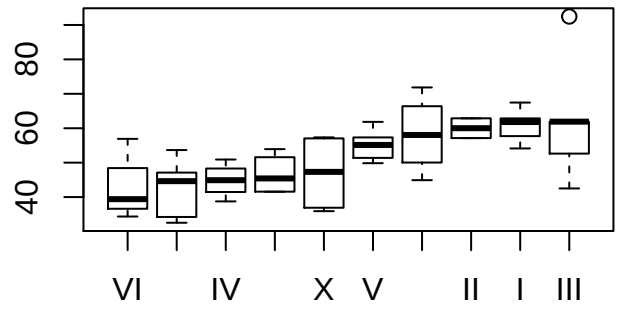
If you want to explore the distribution of lots of related variables, you could draw lots of histograms. For example, if you wanted to see the distribution of Obama votes by US region, you could use latticing/faceting to draw 10 histograms.

This is just about feasible, but it doesn't scale much further. If you need a hundred histograms, the space requirements can easily overwhelm the largest monitor. Box plots (sometimes called box and whisker plots) are a more space-efficient alternative that make it easy to compare many distributions at once. You don't get as much detail as with a histogram or kernel density plot, but simple higher-or-lower and narrower-or-wider comparisons can easily be made.

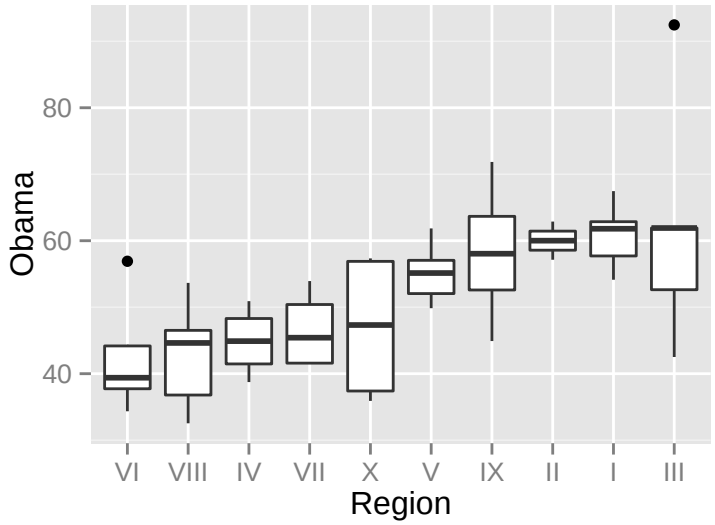
```
# base
boxplot(Obama ~ Region, data = obama_vs_mccain)
```




```
# This type of plot is often clearer if we reorder
# the box plots from smallest to largest, in some sense.
ovm <- within(
  obama_vs_mccain,
  Region <- reorder(Region, Obama, median)
)
boxplot(Obama ~ Region, data = ovm)
```




```
# ggplot2
ggplot(ovm, aes(Region, Obama)) +
  geom_boxplot()
```



Barchart

Bar charts (a.k.a. bar plots) are the natural way of displaying numeric variables split by a categorical variable.

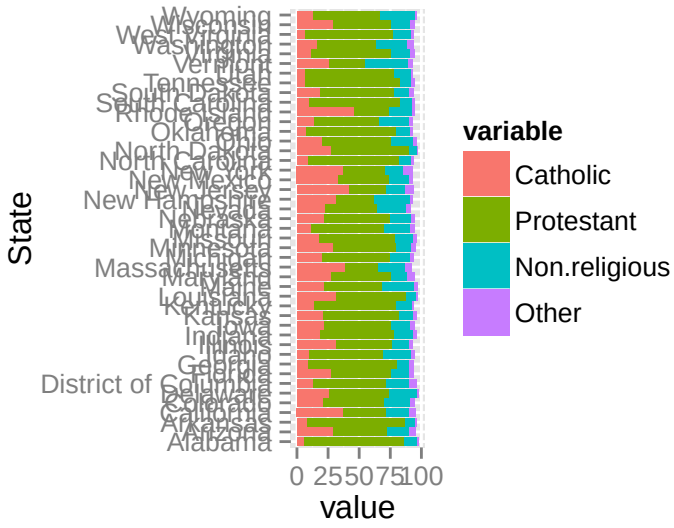
```r
# look at the distribution of religious identification across
# the US states. Data for Alaska and Hawaii are not included
# in the dataset, so we can remove those records:
ovm <- ovm[!(ovm$State %in% c("Alaska", "Hawaii")), ]
```

```r
# ggplot2 requires a tiny bit of work be done to the data to replicate this plot. We need the data in long
require(reshape2)
```

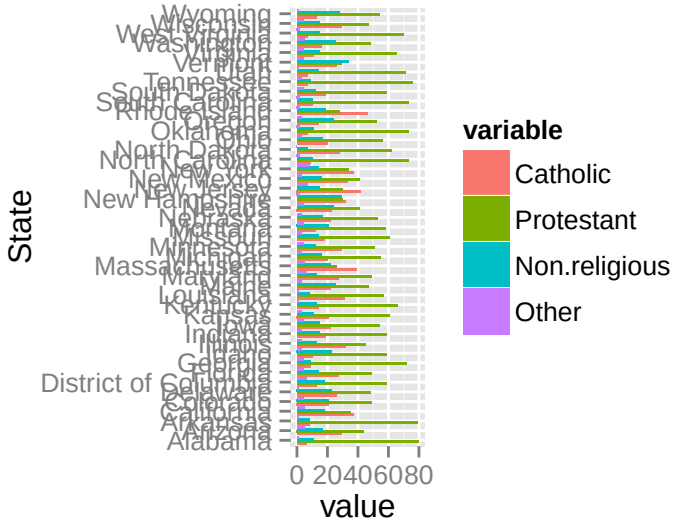
```
## Loading required package: reshape2
```

```r
religions_long <- melt(
  ovm,
  id.vars = "State",
  measure.vars = c("Catholic", "Protestant", "Non.religious", "Other")
)
```

```r
# Like base, gplot2 defaults to vertical bars; adding coord_flip swaps this. Finally, since we already h
ggplot(religions_long, aes(State, value, fill = variable)) +
  geom_bar(stat = "identity") +
  coord_flip()
```



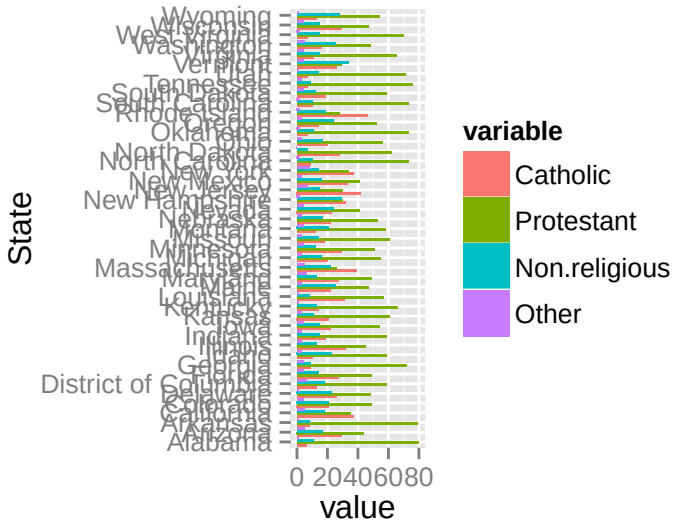

```
# To avoid the bars being stacked, we would have to pass the argument position = "dodge" to geom_bar.

ggplot(religions_long, aes(State, value, fill = variable)) +
  geom_bar(stat = "identity", position = "dodge") +
  coord_flip()
```




```
# The other possibility for that argument is position = "fill", which creates stacked bars that are all
```

To avoid the bars being stacked, we would have to pass the argument position = "dodge" to geom_bar.

```
ggplot(religions_long, aes(State, value, fill = variable)) +
  geom_bar(stat = "identity", position = "dodge") +
  coord_flip()
```



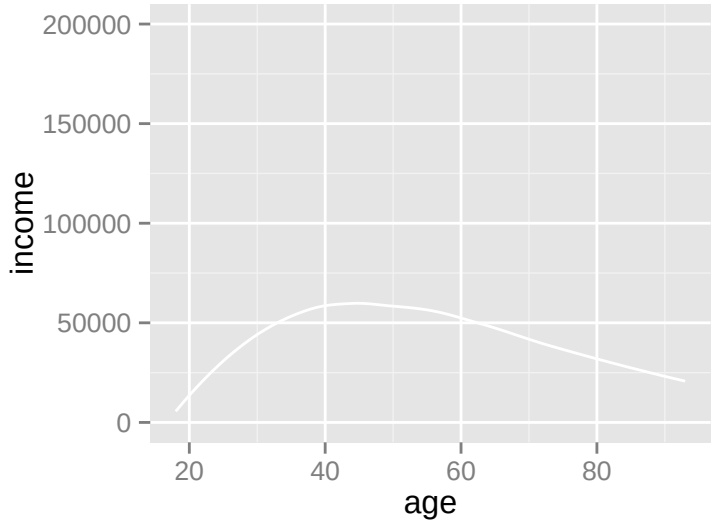
- We can see that there is a definite positive correlation between income and turnout, and it's stronger on the log-log scale.

Question: [2] Does the relationship hold across all of the USA?

To answer this, we can split the data up into the 10 Standard Federal Regions given in the **Region column**, and plot each of the subsets in a 'matrix' in one figure.

```
library(hexbin)

## Error in library(hexbin): there is no package called 'hexbin'

library(ggplot2)
custdata2 <- read.table('custdata2.tsv', header=T, sep='\t')

ggplot(custdata2, aes(x=age,y=income)) +
  geom_hex(binwidth=c(5,10000)) +
  geom_smooth(color = "white", se = F) +
  ylim(0,200000)
```

```
## geom_smooth: method="auto" and size of largest group is <1000, so using loess.
Use 'method = x' to change the smoothing method.

## Warning: Removed 32 rows containing missing values (stat_smooth).
```



Graph type	Uses
Line plot	Shows the relationship between two continuous variables. Best when that relationship is functional, or nearly so.
Scatter plot	Shows the relationship between two continuous variables. Best when the relationship is too loose or cloud-like to be easily seen on a line plot.
Smoothing curve	Shows underlying "average" relationship, or trend, between two continuous variables. Can also be used to show the relationship between a continuous and a binary or Boolean variable: the fraction of *true* values of the discrete variable as a function of the continuous variable.
Hexbin plot	Shows the relationship between two continuous variables when the data is very dense.
Stacked bar chart	Shows the relationship between two categorical variables (var1 and var2). Highlights the frequencies of each value of var1.
Side-by-side bar chart	Shows the relationship between two categorical variables (var1 and var2). Good for comparing the frequencies of each value of var2 across the values of var1. Works best when var2 is binary.
Filled bar chart	Shows the relationship between two categorical variables (var1 and var2). Good for comparing the relative frequencies of each value of var2 within each value of var1. Works best when var2 is binary.
Bar chart with faceting	Shows the relationship between two categorical variables (var1 and var2). Best for comparing the relative frequencies of each value of var2 within each value of var1 when var2 takes on more than two values.

4 Spotting problems using graphics and visualization

4.1 Typical problems revealed by data summaries

Missing values Invalid values and outliers Data range: How narrow is 'too narrow' a data range?

Key takeaways

- Take the time to examine your data before diving into the modeling.
- The summary command helps you spot issues with data range, units, data type, and missing or invalid values.
- Visualization additionally gives you a sense of data distribution and relationships among variables.
- Visualization is an iterative process and helps answer questions about the data. Time spent here is time not wasted during the modeling process.

5 EDA with data.table Package

```
library(data.table)
# salaries <- read.csv("http://dgrtwo.github.io/pages/lahman/Salaries.csv")
salaries <- read.csv("Salaries.csv")
# use the as.data.table function to replace salaries with a data.table version.
salaries = as.data.table(salaries)
salaries
```

```
##        yearID teamID lgID  playerID  salary
##     1:   1985    BAL   AL murraed02 1472819
##     2:   1985    BAL   AL  lynnfr01 1090000
##     3:   1985    BAL   AL ripkeca01  800000
##     4:   1985    BAL   AL  lacyle01  725000
##     5:   1985    BAL   AL flanami01  641667
##    ---
## 23952:   2013    WAS   NL matthry01  504500
## 23953:   2013    WAS   NL lombast02  501250
## 23954:   2013    WAS   NL ramoswi01  501250
## 23955:   2013    WAS   NL rodrihe03  501000
## 23956:   2013    WAS   NL moorety01  493000
```

Notice that it contains the same information, but only shows the first five rows, then the last five rows, which is generally a more convenient representation. This more compact way of printing a `data.table` is the first benefit of using the package.

A lot of things work just the same way as they do in a `data.frame`.

```
salaries$salary
salaries[1,]
salaries[1:6,]
```

One thing that did work on `data frames` but doesn't work on `data tables` is extracting a column based on an index. In a `data.frame`, you could extract the first column by putting the index after the comma:

```
salaries[,1]
# But that doesn't work in data.table. Instead, you
# can put the name of the column, without
# quotes, after the comma:
salaries[,yearID]

#You can also grab multiple columns (for example, just the year and the salary) using list:
salaries[, list(yearID, salary)]

salaries[yearID > 2000, ]
# selected just the American League teams.
salaries[lgID == "AL", ]
# filter for all the rows in the American League that were after 1990.
salaries[lgID == "AL" & yearID >= 1990, ]


#We can also sort the data easily, using the order function in the area before the comma:
salaries[order(salary), ]

# What if we want to sort first by year,
# and then breaking ties with salary? We can do that
# by providing two arguments to the order function:
salaries[order(yearID, salary), ]
```

Note that we can perform multiple operations all in a sequence, by saving the intermediate results. For instance, we can first perform a filtering operation and save it as salaries.filtered:

```
salaries.filtered = salaries[lgID == "AL" & yearID >= 1990, ]
```

Then we can sort it by salary and save it into a new `data table`, which is now both filtered and sorted.

```
salaries.filtered.sorted = salaries.filtered[order(salary), ]
salaries.filtered.sorted
```

```
##        yearID teamID lgID  playerID   salary
##     1:   1993    NYA   AL jamesdi01        0
##     2:   1993    NYA   AL silveda01    10900
##     3:   1994    CHA   AL  carych01    50000
##     4:   1990    BAL   AL  bellju01   100000
##     5:   1990    BAL   AL brownma03   100000
##    ---
## 10023:   2013    NYA   AL rodrial01 29000000
## 10024:   2012    NYA   AL rodrial01 30000000
## 10025:   2011    NYA   AL rodrial01 32000000
## 10026:   2009    NYA   AL rodrial01 33000000
## 10027:   2010    NYA   AL rodrial01 33000000
```

These operations let us easily explore the data and answer basic questions.

5.1 Summarizing Data Within Groups

In our last segment we learned how to download a dataset on baseball player salaries and turn it into a data table, and then to perform some basic organizations on it like filtering and sorting. Now we're going to learn about a more sophisticated and powerful way of processing the data, namely performing summary operations within groups. This is an important and omnipresent task in data analysis.

```
mean(salaries$salary)
max(salaries$salary)
salaries[yearID == 2000, ]$salary
mean(salaries[yearID == 2000, ]$salary)
summarized.year = salaries[, mean(salary), by="yearID"]
```

```
## Warning in gmean(salary): Group 21 summed to more than type 'integer' can hold
so the result has been coerced to 'numeric' automatically, for convenience.
```

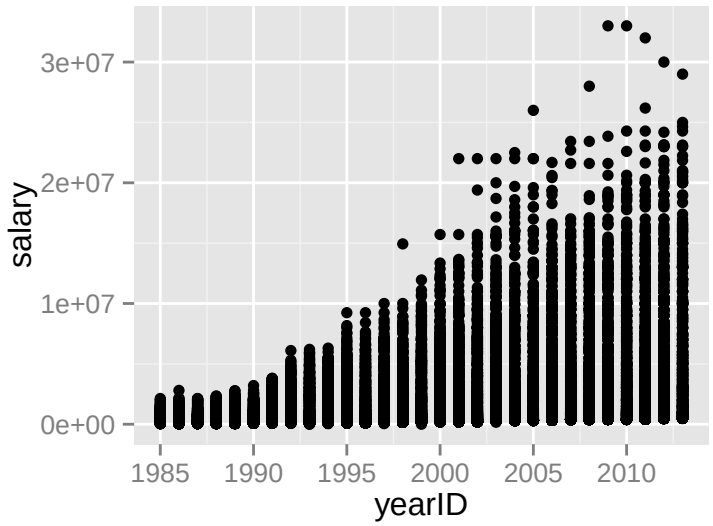
```
summarized.year = salaries[, list(Average=mean(salary)), by="yearID"]
```

```
## Warning in gmean(salary): Group 21 summed to more than type 'integer' can hold
so the result has been coerced to 'numeric' automatically, for convenience.
```

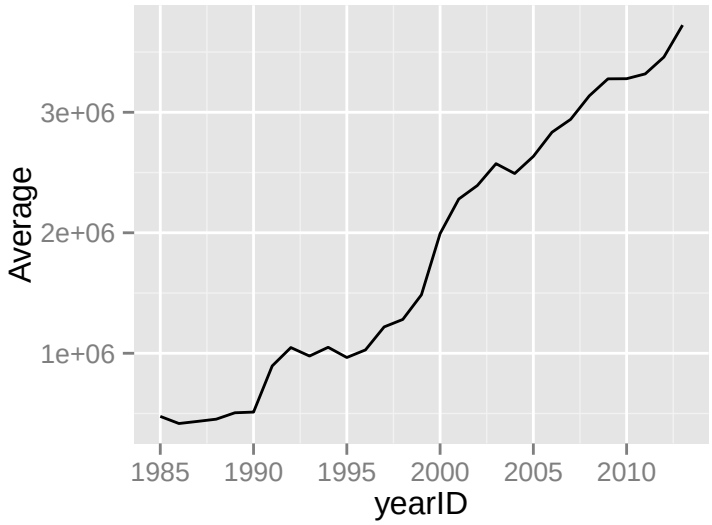
```
summarized.year = salaries[, list(Average=mean(salary), Maximum=max(salary)), by="yearID"]
summarized.year

summarized.lg = salaries[, list(Average=mean(salary), Maximum=max(salary)), by="lgID"]

summarized.year.lg = salaries[, list(Average=mean(salary), Maximum=max(salary)), by=c("yearID", "lgI

summarized.team = salaries[, list(Average=mean(salary), Maximum=max(salary)), by="teamID"]
```

5.2 Exlporing data

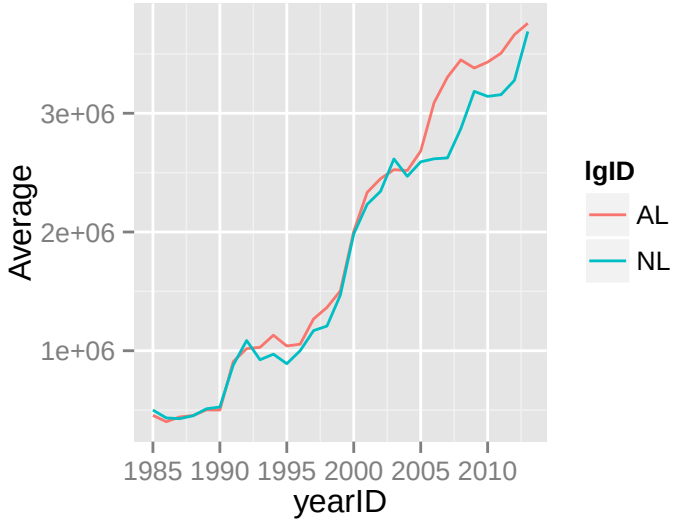
```
ggplot(salaries, aes(yearID, salary)) + geom_point()
```




```
ggplot(summarized.year, aes(yearID, Average)) + geom_line()
```




```
ggplot(summarized.year.lg, aes(yearID, Average, col=lgID)) + geom_line()
```



5.3 Merging data

```
View(Salaries)
```

We find that the players are not represented by their actual first and last names- they're represented by some kind of ID. This ID looks pretty unhelpful: why not just put their names in that column?

The first reason is that there are multiple players in history that have the same name, and at that point if you used their names to identify them, it wouldn't be possible to tell them apart in the data. Meanwhile, these IDs are guaranteed to be unique per player. There are other advantages: for example, the player ID is shorter and therefore takes up less storage space- but the uniqueness is the most important. That ID can be used to connect this column to other datasets.

```
# master = read.csv("http://dgrtwo.github.io/pages/lahman/Master.csv")
master <- read.csv("Master.csv")
master <- as.data.table(master)
# View(master)
```

This is a master list of the baseball players based on their ID. Here in the first column you can see the `playerIDs` that appeared in the salaries data. But you can also see a lot of biographical information, like their birthday and birthplace, their weight and height, the date of their death, and most importantly, their full name.

```
# The "by" argument defines what column we should use to merge them.
merged.salaries <- merge(salaries, master, by="playerID")
```

So we've combined these two tables based on this common column: we have them all in one place. If you wanted to look for trends in salary- for instance, a connection of salary to a player's height, weight, or birth country- you now have all the information in one data table.

One note, having their first and last names as different columns is useful, but we'd like to combine them together into a new column, of first name-space-last name. One way we can create a new column in a data.table is with the `:= operator`:

```
merged.salaries[, name:= paste(nameFirst, nameLast)]
```

```
##         playerID yearID teamID lgID  salary birthYear birthMonth birthDay
##     1: aardsda01   2004    SFN   NL  300000      1981         12       27
##     2: aardsda01   2007    CHA   AL  387500      1981         12       27
##     3: aardsda01   2008    BOS   AL  403250      1981         12       27
##     4: aardsda01   2009    SEA   AL  419000      1981         12       27
##     5: aardsda01   2010    SEA   AL 2750000      1981         12       27
##    ---
## 23952: zumayjo01   2011    DET   AL 1400000      1984         11        9
## 23953: zupcibo01   1991    BOS   AL  100000      1966          8       18
## 23954: zupcibo01   1992    BOS   AL  109000      1966          8       18
## 23955: zupcibo01   1993    BOS   AL  222000      1966          8       18
```

```
## 23956: zuvelpa01   1989    ATL   NL  145000      1958         10       31
##        birthCountry birthState   birthCity deathYear deathMonth deathDay
##     1:          USA         CO      Denver        NA         NA       NA
##     2:          USA         CO      Denver        NA         NA       NA
##     3:          USA         CO      Denver        NA         NA       NA
##     4:          USA         CO      Denver        NA         NA       NA
##     5:          USA         CO      Denver        NA         NA       NA
##    ---
## 23952:          USA         CA Chula Vista        NA         NA       NA
## 23953:          USA         PA  Pittsburgh        NA         NA       NA
## 23954:          USA         PA  Pittsburgh        NA         NA       NA
## 23955:          USA         PA  Pittsburgh        NA         NA       NA
## 23956:          USA         CA   San Mateo        NA         NA       NA
##        deathCountry deathState deathCity nameFirst nameLast   nameGiven
##     1:                                      David  Aardsma David Allan
##     2:                                      David  Aardsma David Allan
##     3:                                      David  Aardsma David Allan
##     4:                                      David  Aardsma David Allan
##     5:                                      David  Aardsma David Allan
##    ---
## 23952:                                       Joel   Zumaya Joel Martin
## 23953:                                        Bob   Zupcic      Robert
## 23954:                                        Bob   Zupcic      Robert
## 23955:                                        Bob   Zupcic      Robert
## 23956:                                       Paul  Zuvella        Paul
##        weight height bats throws      debut  finalGame  retroID   bbrefID
##     1:    205     75    R      R 2004-04-06 2013-09-28 aardd001 aardsda01
##     2:    205     75    R      R 2004-04-06 2013-09-28 aardd001 aardsda01
##     3:    205     75    R      R 2004-04-06 2013-09-28 aardd001 aardsda01
##     4:    205     75    R      R 2004-04-06 2013-09-28 aardd001 aardsda01
##     5:    205     75    R      R 2004-04-06 2013-09-28 aardd001 aardsda01
##    ---
## 23952:    215     75    R      R 2006-04-03 2010-06-28 zumaj001 zumayjo01
## 23953:    220     76    R      R 1991-09-07 1994-08-04 zupcb001 zupcibo01
## 23954:    220     76    R      R 1991-09-07 1994-08-04 zupcb001 zupcibo01
## 23955:    220     76    R      R 1991-09-07 1994-08-04 zupcb001 zupcibo01
## 23956:    173     72    R      R 1982-09-04 1991-05-02 zuvep001 zuvelpa01
##                name
##     1: David Aardsma
##     2: David Aardsma
##     3: David Aardsma
```

```
##     4: David Aardsma
##     5: David Aardsma
##    ---
## 23952:   Joel Zumaya
## 23953:    Bob Zupcic
## 23954:    Bob Zupcic
## 23955:    Bob Zupcic
## 23956:  Paul Zuvella
```

This means assign a new column, name, and now we can give it a value based on other columns in the dataset. The paste function is a useful function in R for combining two vectors of strings to be separated by spaces. If we put nameFirst and nameLast, because we're within the data.table, that we want to combine those two names into a new name.

Merging can sometimes be a bit more complicated. For example, let's bring in one more dataset, this one a history of each player's batting statistics for each year.

```
# batting = read.csv("http://dgrtwo.github.io/pages/lahman/Batting.csv")
batting <- read.csv("Batting.csv")
batting <- as.data.table(batting)
```

This is the most complex dataset yet. Here, like the salary data, we have one row per player per year, and their team ID and league ID. But we also have many statistics summarizing how well he did at batting that year. For instance, G represents how many games the player played in, AB represents the number of times a player went up to bat (how many chances they had to get a hit), H represents the number of hits, and HR represents the number of home runs he scored (hitting the ball out of the park, which gets a run in just one hit).

Now, let's say we want to combine this data with the salary data- for example so we can see **how salary is correlated with performance**. First, notice that the salary table and the batting table **don't share only one column** of player ID: they share four: playerID, teamID, leagueID and yearID. That's because we have multiple batting statistics and salary for each single player. This means we won't just be merging by player: we'll be **merging them based on the combination of all four columns.**

The way we do that is with the by argument to merged. Instead of giving just the playerID to by, we give a vector of the four shared columns.

```
merged.batting <- merge(batting, salaries, by=c("playerID", "yearID", "teamID", "lgID"))
```

Now it has all the information that was in the batting dataset, but it also added a column for salary. Another thing to note is that we don't have salary information on every player in every

year: in particular, we've lost all information on players before 1985. There is a way we can fix this, by adding the all.x option to the merge function:

```
merged.batting <- merge(batting, salaries, by=c("playerID", "yearID", "teamID", "lgID"), all.x=TRUE)
```

This means "keep everything in the first dataset we're merging," which is batting (all.y would mean "keep everything in the second dataset"). Notice now that now all rows have information in the salary column: some have NA, which means "missing value," or "not applicable." So notice that all the rows where we have salary data get to keep their value, while all the ones that don't get filled in by the missing value NA.

Now we can take this merged dataset and merge it with our biographical data in the master list. Here that would be

```
merged.all <- merge(merged.batting, master, by="playerID")
```

Now we see we still have the same batting information, but we also have the biographical information from the master list- for example, each player's real name. We've created one mega-dataset covering all three kinds of information. The Lahman baseball dataset contains a lot more information, including player's fielding statistics, presence in the Hall of Fame, pitchers, managers, and so on, all sharing these same IDs. **By merging these datasets in the right way, you can answer very complex and interesting questions**.

5.4 EDA

So let's wrap up by taking all these tools together on our **mega-merged dataset**. Just like any other dataset, we can filter and process this. For example, this dataset includes pitchers, who might never go up to bat in a whole season. That could end up skewing our analysis.

```
#head(merged.all)
```

An example would be David Aardsma, who in many years never even had a single At Bat (AB is 0). We can start by filtering out all the years in which someone has no At Bats.

```
merged.all <- merged.all[AB > 0, ]
# Now we can see that all At Bat's are at least 1.
```

Now, one thing baseball fans like looking for is career records. That means we want to summarize across all the years that a batter played, and find, for example, the total number of home runs each player hit. Recall that we learned to do that with "by". For example:

```
#Here we create one column, Total.HR, which we define as the sum of home runs for each player, and we tel
summarized.batters <- merged.all[, list(Total.HR=sum(HR)), by="playerID"]
```

Now we can see that we've created a new data.table that contains each player's ID and their total career home runs. But in the process, since the only thing we're summarizing by is the player ID, we lost track of their actual first and last names. There's a simple way around that. First, recall that we can create a new column that combines the players' first and last names using `paste` and `:=`, and let's try the same trick again, this time on `merged.all`:

```
merged.all[, name := paste(nameFirst, nameLast)]
#Now we've added to merged.all a name column:
merged.all
```

Now when we perform this summary, let's do it not just on the player ID, but also on their name:

```
summarized.batters <- merged.all[, list(Total.HR=sum(HR)), by=c("playerID", "name")]
summarized.batters
```

By summarizing based on these two columns, we can keep both their ID and their real name.

Now, just like any data.table, we can sort it to find out who the top home-run hitters are. For this we use the order function:

```
summarized.batters[order(Total.HR), ]
```

```
##         playerID           name Total.HR
##     1: aardsda01  David Aardsma        0
##     2:  aasedo01       Don Aase        0
##     3:  abadan01      Andy Abad        0
##     4:  abadfe01  Fernando Abad        0
##     5: abadijo01    John Abadie        0
##    ---
## 16336: rodrial01 Alex Rodriguez      654
## 16337:  mayswi01    Willie Mays      660
## 16338:  ruthba01      Babe Ruth      714
## 16339: aaronha01     Hank Aaron      755
## 16340: bondsba01    Barry Bonds      762
```

Baseball fans won't be surprised that at the top we can see Barry Bonds, Hank Aaron, Babe Ruth, and some other legendary baseball hitters. In the same way we can summarize by other

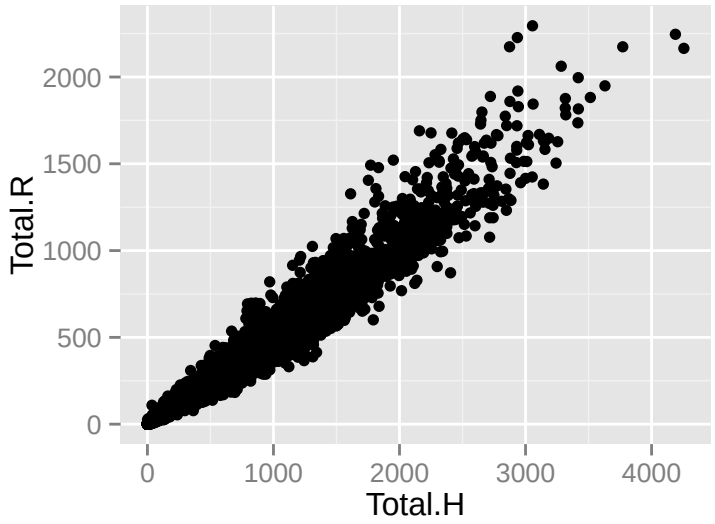
statistics, like total number of hits or runs. For instance, here let's add Total.R for total number of runs, and Total.H for total number of hits.

```
summarized.batters <- merged.all[, list(Total.HR=sum(HR), Total.R=sum(R), Total.H=sum(H)), by=c("play

summarized.batters
```

Now we've saved all that career information into summarized.batters.

The more a player gets hits in baseball, the more chance they have to actually score runs. That means it's not surprising that there's a correlation between them. We can take a look at that correlation through ggplot. We'll put total hits (Total.H) on the x-axis and total runs (Total.R) on the y-axis.

```
ggplot(summarized.batters, aes(Total.H, Total.R)) + geom_point()
```



Here we can see a clear correlation between the number of hits a player gets and the number of runs.

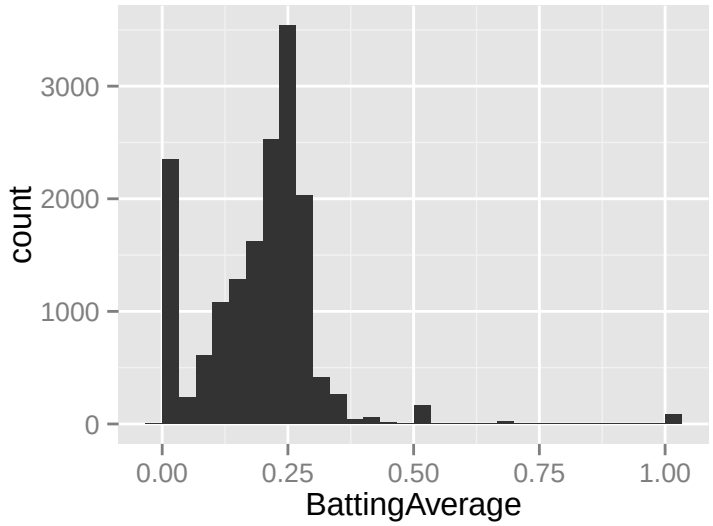
So far each of these summaries has been of one statistic: the total number of home runs, or the total number of hits. But some baseball statistics are calculated based on several of a player's statistics. For example, consider the batting average, which is the number of hits a player gets, divided by the number of times he goes up to bat.

```
#head(merged.all)
#So in our batting dataset, for Hank Aaron in 1955, we can see that he had 189 hits out of 602 at-bats. We
```

```
summarized.batters <- merged.all[, list(Total.HR=sum(HR), Total.R=sum(R), Total.H=sum(H), BattingAver

summarized.batters
```

This kind of summary operation thus lets us generate any statistic we're interested in. We could then, for instance, put it into a histogram to find out its distribution:

```
ggplot(summarized.batters, aes(BattingAverage)) + geom_histogram()

## stat_bin: binwidth defaulted to range/30. Use 'binwidth = x' to adjust this.
```



We can see that they center around about 25%, with a large number of people with close to 0 batting average, which would mostly be pitchers.

In this way you're able to test hypotheses almost as fast as you can think of them. This loop of asking questions about your data and getting answers back is the core of exploratory data analysis.

References

Richard Cotton. *Learning R*. " O'Reilly Media, Inc.", 2013.