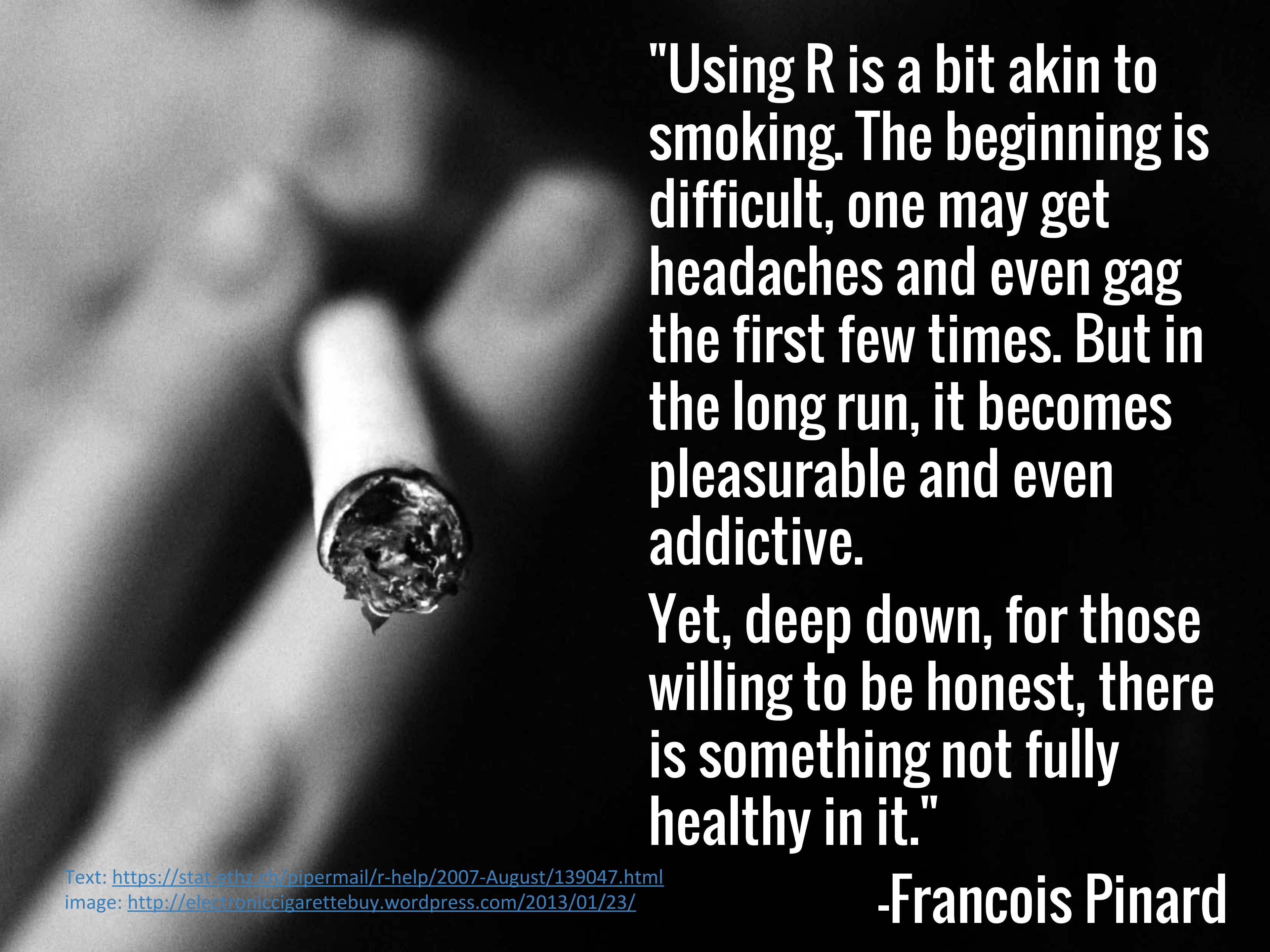
Introduction to R

for I-TIPP Fellows

Matthew J Maenner, PhD
13 October 2015

www.github.com/mjmaenner/rdemo

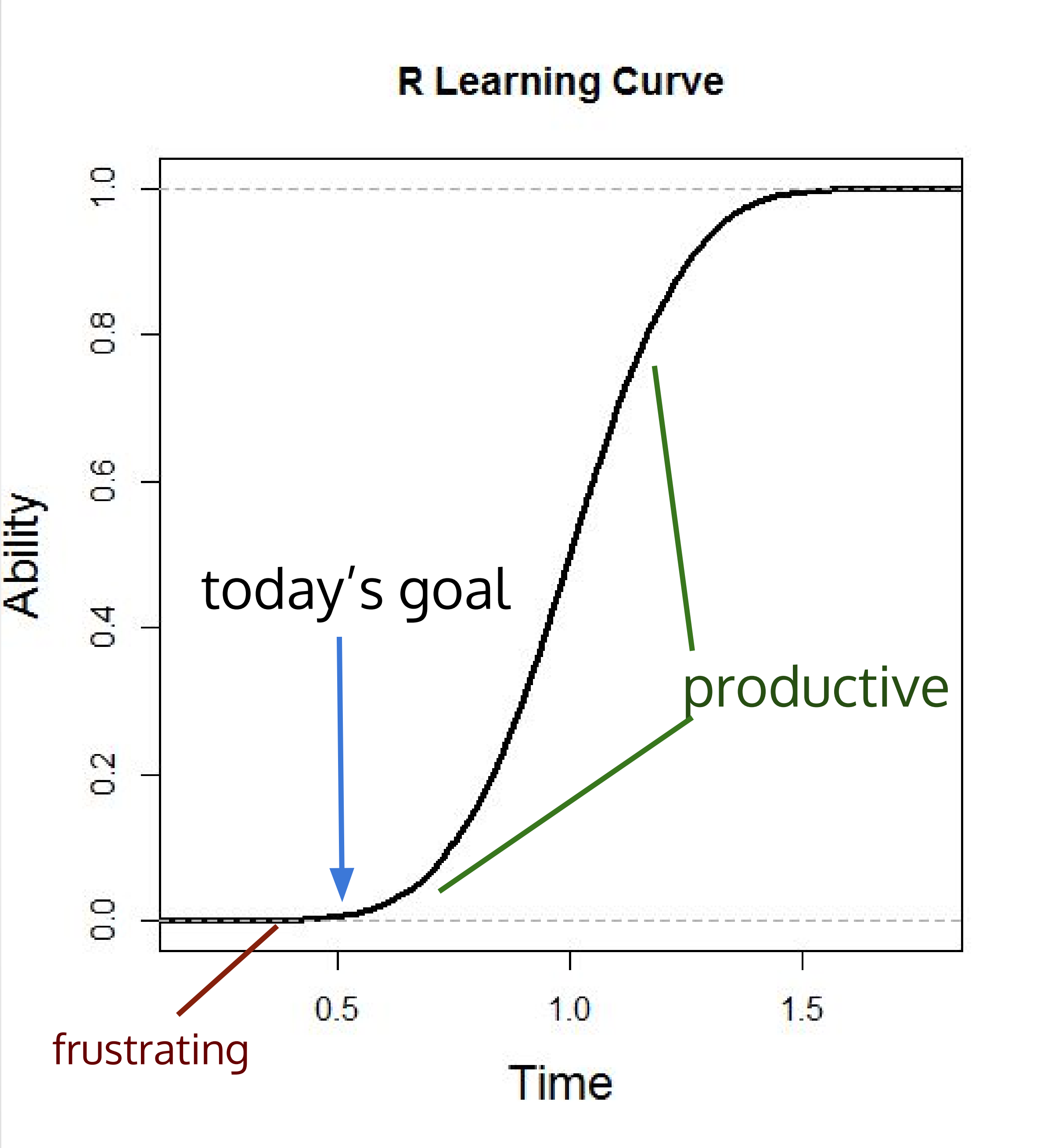


"Using R is a bit akin to smoking. The beginning is difficult, one may get headaches and even gag the first few times. But in the long run, it becomes pleasurable and even addictive.

Yet, deep down, for those willing to be honest, there is something not fully healthy in it."

-Francois Pinard

Text: https://stat.ethz.ch/pipermail/r-help/2007-August/139047.html
image: http://electroniccigarettebuy.wordpress.com/2013/01/23/



```
plot(ecdf(rnorm(n=10000, mean=1, sd=0.2)), xlab="Time",ylab="Ability", lwd=3,main="R Learning Curve", cex.lab=18/12)
```

A very brief orientation to these things:

1. Interacting with R and RStudio
2. Basic syntax, notation, and operators
3. Doing things (**functions)**, using packages
4. Strategies for learning and suggested tools to learn next

slides and code examples:

www.github.com/mjmaenner/rdemo

(feel free to interrupt and ask questions)

R

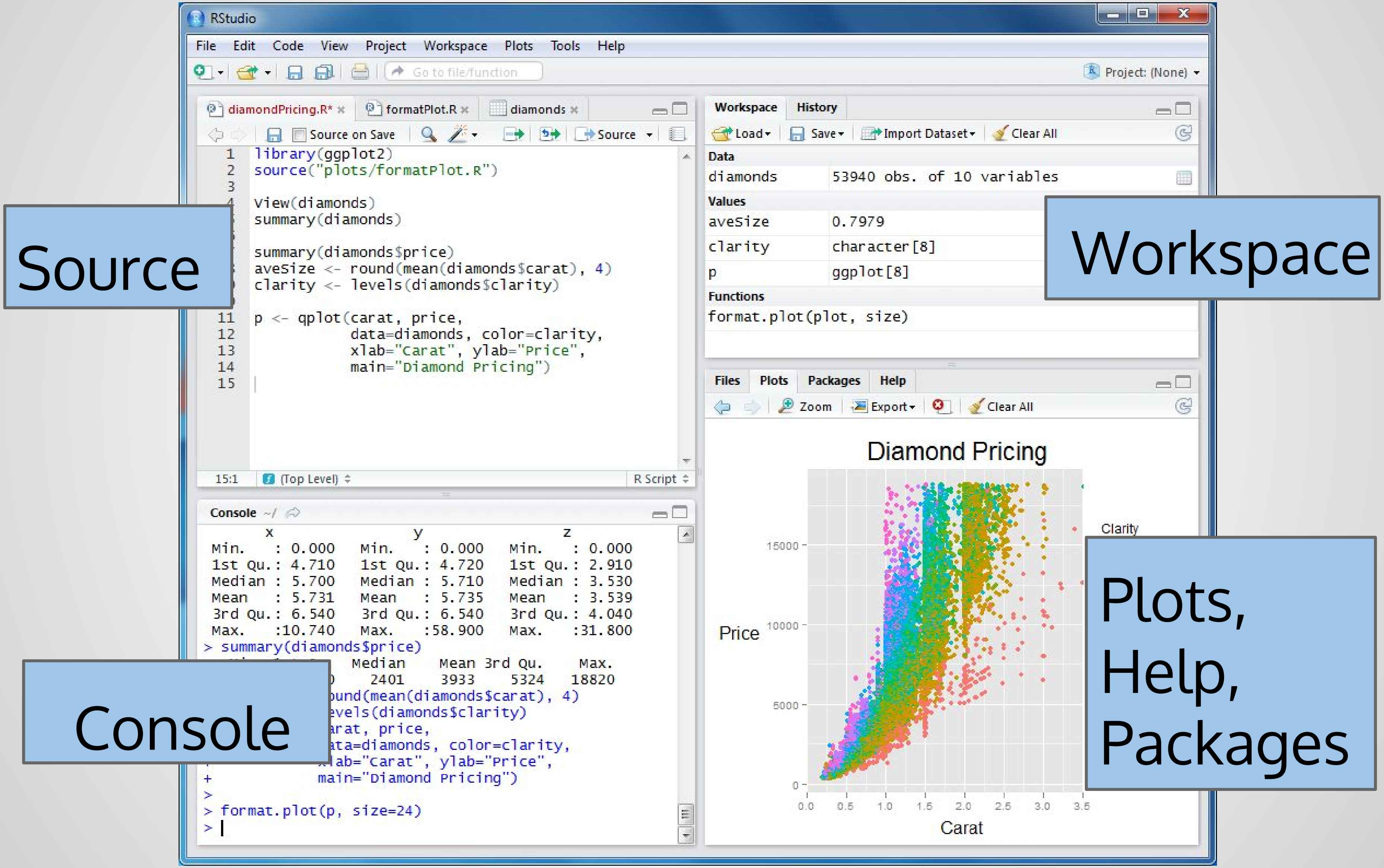
Open-source programming language and statistical computing environment

Created in 1990’s; implementation of the S language (Bell Labs)

The R Core Team now maintains “base” R, and 7000+ packages have been contributed by people all over the world.

Adapted from: http://en.wikipedia.org/wiki/R_(programming_language))

RStudio



Source

Workspace

Console

Plots, Help, Packages

```r
library(ggplot2)
source("plots/formatPlot.R")

View(diamonds)
summary(diamonds)

summary(diamonds$price)
aveSize <- round(mean(diamonds$carat), 4)
clarity <- levels(diamonds$clarity)

p <- qplot(carat, price,
           data=diamonds, color=clarity,
           xlab="Carat", ylab="Price",
           main="Diamond Pricing")
```

Data	
diamonds	53940 obs. of 10 variables
Values	
aveSize	0.7979
clarity	character[8]
p	ggplot[8]
Functions	
format.plot(plot, size)	

```
      x               y               z
 Min.   : 0.000  Min.   : 0.000  Min.   : 0.000
 1st Qu.: 4.710  1st Qu.: 4.720  1st Qu.: 2.910
 Median : 5.700  Median : 5.710  Median : 3.530
 Mean   : 5.731  Mean   : 5.735  Mean   : 3.539
 3rd Qu.: 6.540  3rd Qu.: 6.540  3rd Qu.: 4.040
 Max.   :10.740  Max.   :58.900  Max.   :31.800
> summary(diamonds$price)
                  Median    Mean 3rd Qu.    Max.
                )   2401    3933    5324   18820
                ound(mean(diamonds$carat), 4)
                evels(diamonds$clarity)
                arat, price,
                ata=diamonds, color=clarity,
                lab="Carat", ylab="Price",
+               main="Diamond Pricing")
>
> format.plot(p, size=24)
>
```

Default Rstudio setup http://www.rstudio.com/ide/screenshots/

RStudio Basics

Usually, type in the source window

file -> new file -> R Script (if you don’t see source window)

Ctrl+Enter submits line or selection to console

you can type in the console, too. Statements end at the end of lines, unless in a parentheses, bracket, or extended from previous line (like a + in ggplot2).

Tab completion for additional details:

type **summa** +tab to find **summary**; type **mean(** +tab to see options

At the R console, can recall previous commands with up arrow

A simplification

(most) everything is an object

either data or function

Data objects store information.

You do things *to* data objects

Functions do things; usually kept in packages

Call functions with **parentheses**: `citation()`

without parentheses, it prints the function code

Comprehensive R Archive Network

Perhaps **the** most compelling reason use R

CRAN is like R's App Store

```
> install.packages("ggplot2") #from CRAN
Installing package(s) into ‘C:/prog/R/R-3.2.1/library’ ...
> library(ggplot2) #loads for use
```

You only have install 1st time (or to update package).

Otherwise, only need `library()`

What package(s) should I use?

- Find out what others use (Google, scientific literature)
- CRAN task views: http://cran.r-project.org/web/views/
- Serious packages often have their own papers.

Basic Syntax and Operators

`#` comment (to end of line)

`<-` assign value to *objects*

inside functions, must assign *arguments* with "="

```
weighted.mean(x=rate, w=population)
```

`==` `test of equality`

`$` specifies variables (or sub-objects) in a data.frame or list

`c` concatenates multiple values into a vector

```
c(1, 2, 3)
```

How to get help

?function or help(function)

try it: `?mean, ?lm, ?summary, ?ggplot`

it only works for packages that are currently loaded

Google

specific task (e.g., calculate quantiles in R)

errors (e.g., “subscript out of bounds” R)

More targeted searches:

?? performs a text search across packages that you have installed

try it: `??mean`

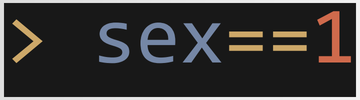
`RSiteSearch("ggplot2")` searches R site for string

Find out other arguments and options: `?RSiteSearch`

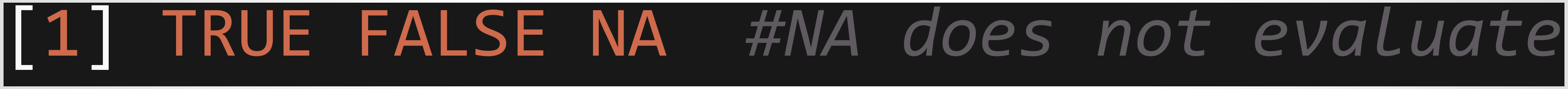
Missing Values

Represented as **NA**; you should explicitly account for NA values when transforming or subsetting data.

Try this: `sex<- c(1, 0, NA)`



```
> sex==1
```



```
[1] TRUE FALSE NA  #NA does not evaluate
```

also, notice the vector recycling

Try: **`is.na(sex)`** or **`!(is.na(sex))`** #What happens?

`is.finite(sex)`

more: http://www.statmethods.net/input/missingdata.html

Loading Data

Many people prefer **CSV**

```
crimeatl <- read.csv(“~/crime.csv”, header=TRUE)
```

read.csv is a special case of read.table

SPSS, **Stata** and SAS via the **`foreign`** & **`sas7bdat`** packages

```
data<- read.spss(file="c:/data.sav",to.data.frame=TRUE)
```

XLS or XLSX (not recommended, via **`xlsx`** pacakge)

```
data<-read.xlsx(file="c:/data.xlsx",sheetIndex=1)
```

Database connectivity via RODBC and others.

```
ch<-odbcConnect(dsn="SQLServer11", readOnly=TRUE)
data<-sqlFetch(channel=ch, sqtable='_DR_FINALDATA')
```

Loading Data (exercise)

local csv:

```
> crimeatl <- read.csv(“~/crime.csv”,
header=TRUE)
```

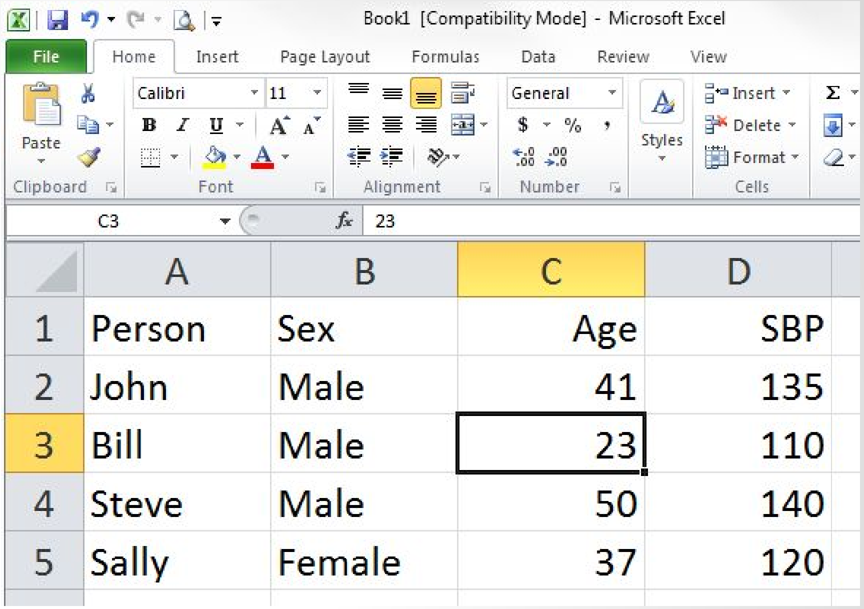
RSTUDIO GUI: environment tab (upper right)

click “import dataset” -- “from text file”

follow prompts

The data.frame

A rectangular data set
often abbreviated as “df”
Row : an observation
Column : a variable



	A	B	C	D
1	Person	Sex	Age	SBP
2	John	Male	41	135
3	Bill	Male	23	110
4	Steve	Male	50	140
5	Sally	Female	37	120

each column has its own **class**:

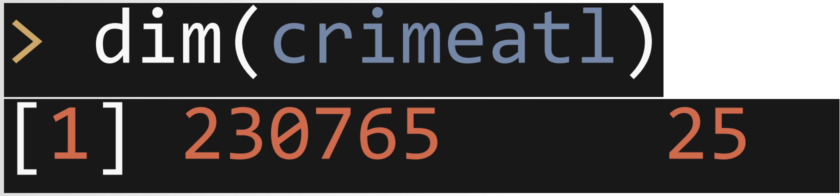
- **numeric** values
- **character** strings
- **factors** (categorical; may or may not be ordered)

Check the class of your variable:

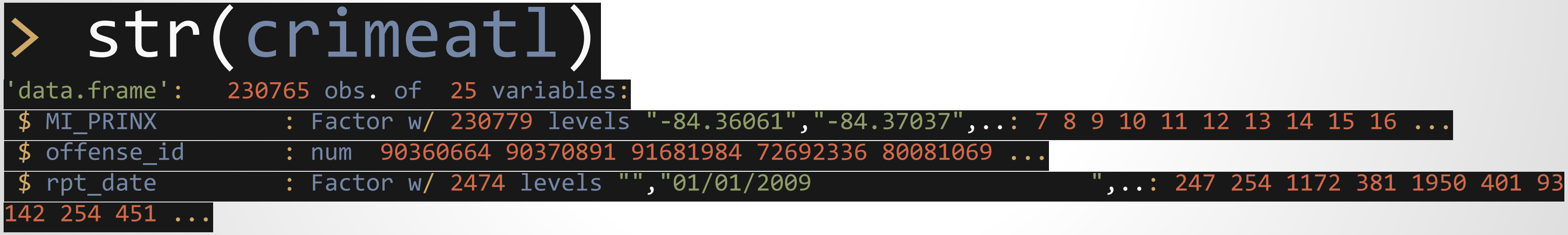
```
>class(crimeatl$beat)
>[1] "factor"
```

Look at your data

Did it load the way you thought it would?



```
> dim(crimeatl)
[1] 230765      25 # number of rows and columns
```



```
> str(crimeatl)
'data.frame':	230765 obs. of  25 variables:
 $ MI_PRINX         : Factor w/ 230779 levels "-84.36061","-84.37037",..: 7 8 9 10 11 12 13 14 15 16 ...
 $ offense_id       : num  90360664 90370891 91681984 72692336 80081069 ...
 $ rpt_date         : Factor w/ 2474 levels "","01/01/2009                    ",..: 247 254 1172 381 1950 401 93
142 254 451 ...
```

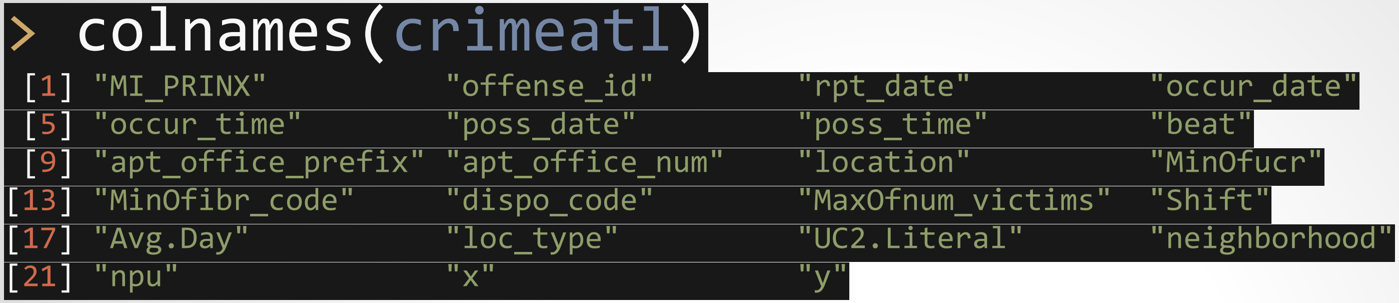
Look at your data

Did it load the way you thought it would?

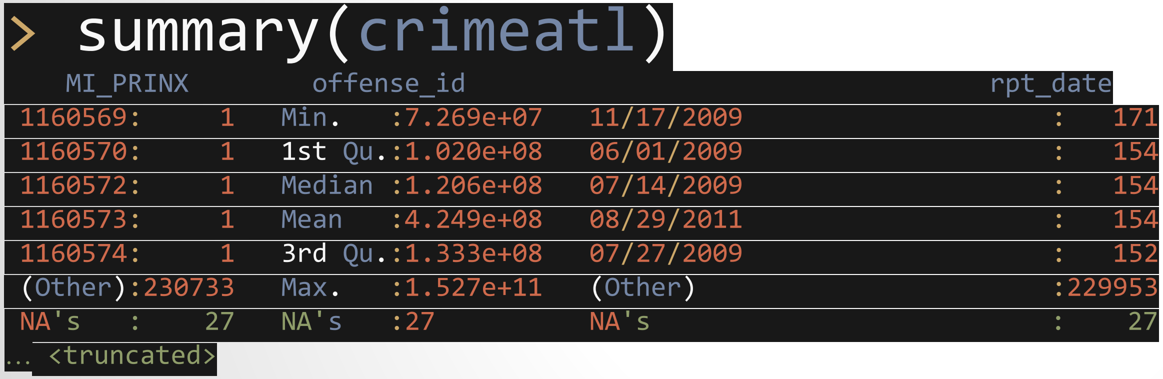
```
> View(crimeatl) # opens spreadsheet like view
```

```
> colnames(crimeatl)
 [1] "MI_PRINX"          "offense_id"        "rpt_date"          "occur_date"
 [5] "occur_time"        "poss_date"         "poss_time"         "beat"
 [9] "apt_office_prefix" "apt_office_num"    "location"          "MinOfucr"
[13] "MinOfibr_code"     "dispo_code"        "MaxOfnum_victims"  "Shift"
[17] "Avg.Day"           "loc_type"          "UC2.Literal"       "neighborhood"
[21] "npu"               "x"                 "y"
```

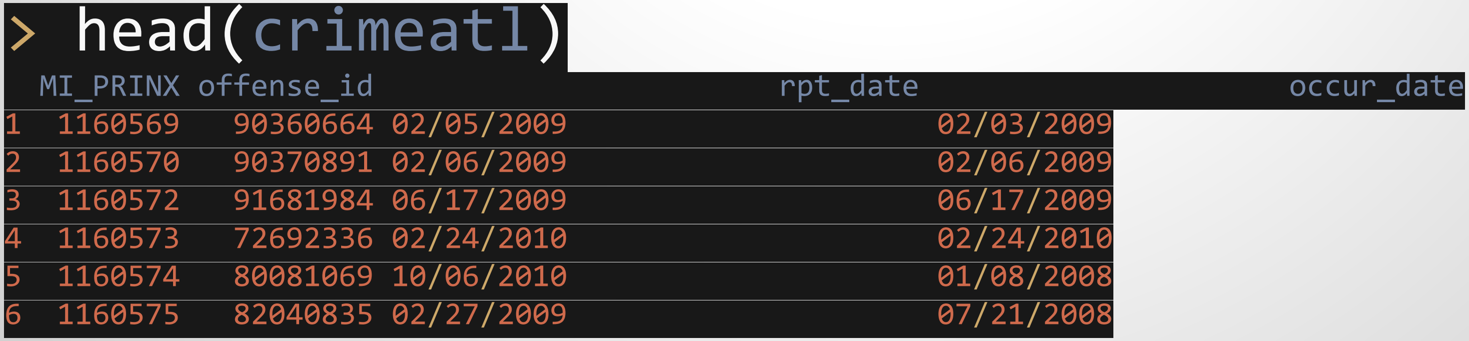
Look at your data (cont'd)



```
> colnames(crimeatl)
 [1] "MI_PRINX"          "offense_id"        "rpt_date"          "occur_date"
 [5] "occur_time"        "poss_date"         "poss_time"         "beat"
 [9] "apt_office_prefix" "apt_office_num"    "location"          "MinOfucr"
[13] "MinOfibr_code"     "dispo_code"        "MaxOfnum_victims"  "Shift"
[17] "Avg.Day"           "loc_type"          "UC2.Literal"       "neighborhood"
[21] "npu"               "x"                 "y"
```



```
> summary(crimeatl)
    MI_PRINX          offense_id                                         rpt_date
 1160569:      1    Min.    :7.269e+07    11/17/2009                   :    171
 1160570:      1    1st Qu.:1.020e+08    06/01/2009                   :    154
 1160572:      1    Median :1.206e+08    07/14/2009                   :    154
 1160573:      1    Mean    :4.249e+08    08/29/2011                   :    154
 1160574:      1    3rd Qu.:1.333e+08    07/27/2009                   :    152
 (Other):230733    Max.    :1.527e+11    (Other)                      :229953
 NA's    :     27    NA's    :27           NA's                         :     27
... <truncated>
```



```
> head(crimeatl)
  MI_PRINX offense_id                          rpt_date                   occur_date
1   1160569    90360664 02/05/2009                          02/03/2009
2   1160570    90370891 02/06/2009                          02/06/2009
3   1160572    91681984 06/17/2009                          06/17/2009
4   1160573    72692336 02/24/2010                          02/24/2010
5   1160574    80081069 10/06/2010                          01/08/2008
6   1160575    82040835 02/27/2009                          07/21/2008
```

Using Functions

```
function(arg1 = value, arg2 = value, …)
```

- Follow the arguments (help or tab complete)
- Explicitly name arguments, especially when using something new
  - R will also do partial matching
- Decide where output should go:
  - console, or to be saved to a new object

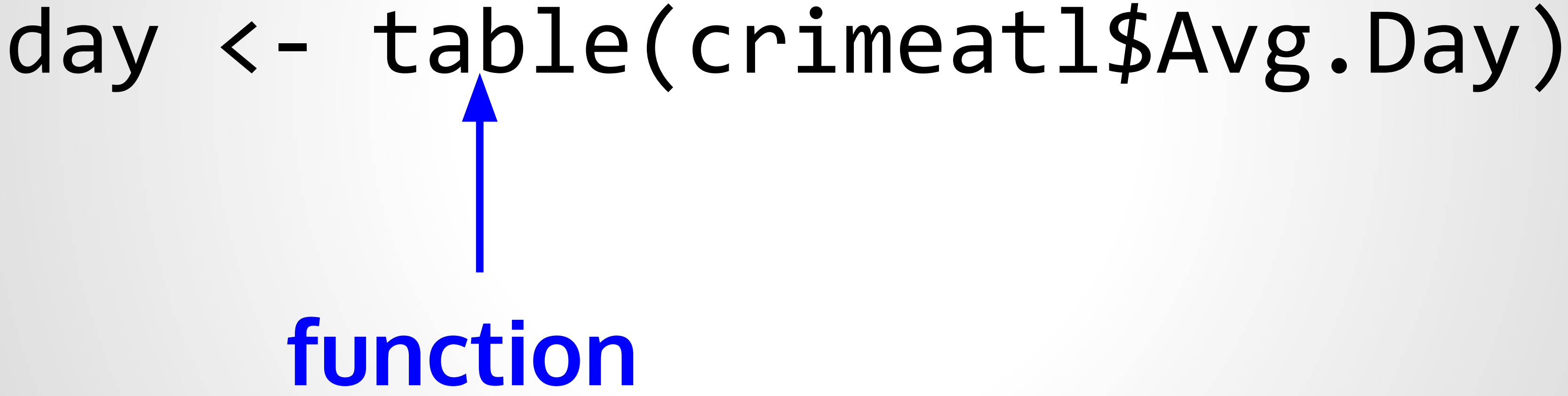
Functions

```
function(arg1 = value, arg2 = value, …)
```

```
day <- table(crimeatl$Avg.Day)
```

Functions

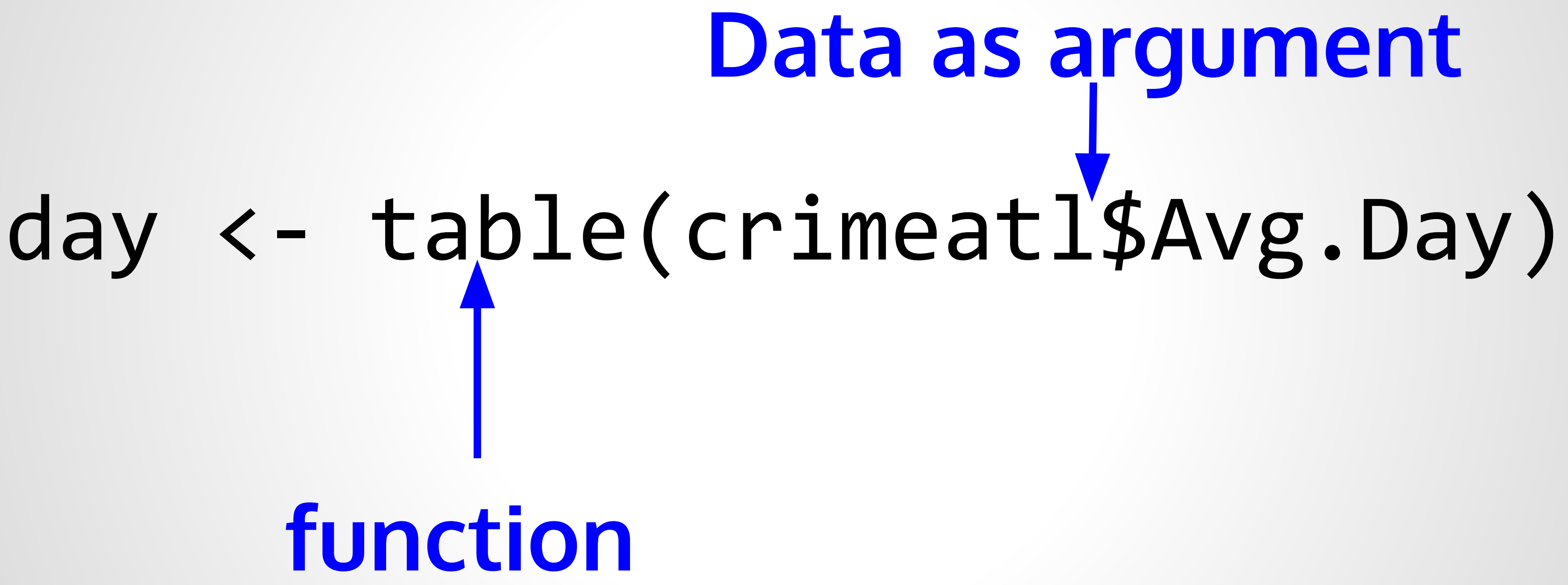
```
function(arg1 = value, arg2 = value, ...)
```



```
day <- table(crimeatl$Avg.Day)
```

Functions

```
function(arg1 = value, arg2 = value, …)
```



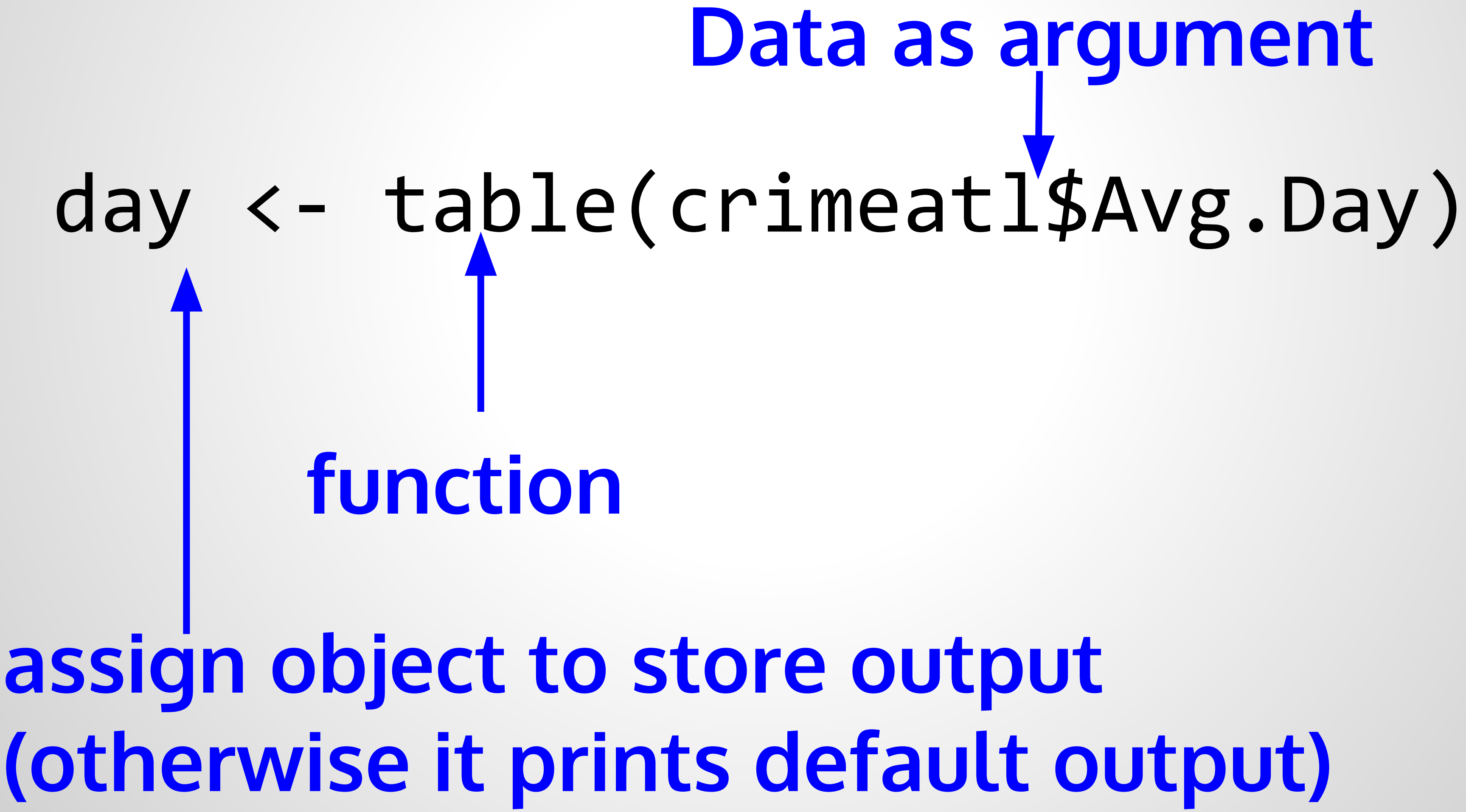
Data as argument

```
day <- table(crimeatl$Avg.Day)
```

function

Functions

```
function(arg1 = value, arg2 = value, ...)
```



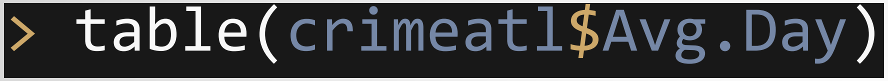
Data as argument

```
day <- table(crimeatl$Avg.Day)
```

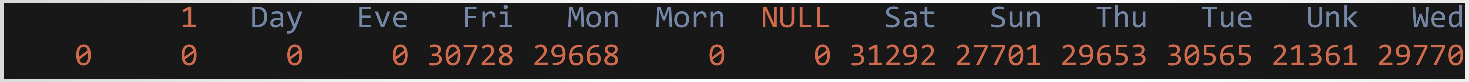
function

assign object to store output (otherwise it prints default output)

Useful Descriptive Functions



```
> table(crimeatl$Avg.Day)
```



```
          1   Day   Eve   Fri   Mon  Morn  NULL   Sat   Sun   Thu   Tue   Unk   Wed
    0     0     0     0 30728 29668     0     0 31292 27701 29653 30565 21361 29770
```

#try adding ,useNA= “always”

```
aggregate() # performing function on groups
mean()
quantile()
```

Accessing parts of a data frame

```
data.frame[row, column]
```

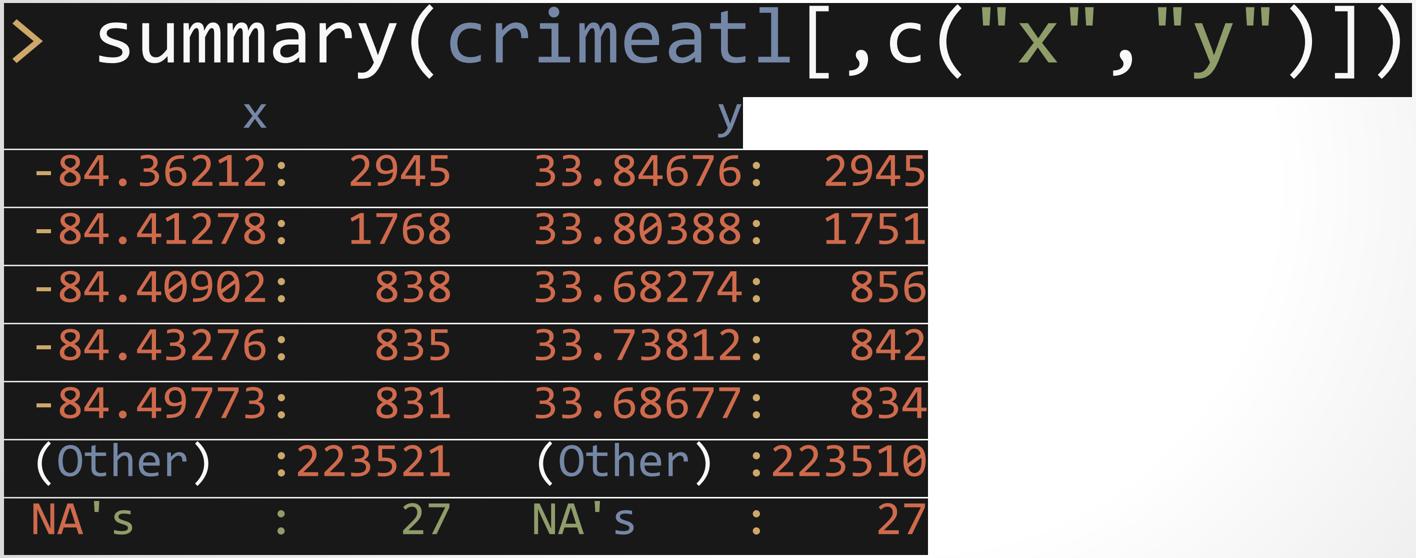
```
crimeatl[ 1:3 , ] # returns rows 1-3
crimeatl[ , 1:3 ] # returns columns 1-3
```

These return the **same** **values**:

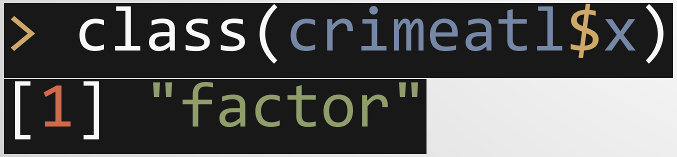
```
crimeatl$Avg.Day #tab-complete after $
crimeatl[ , 17 ]
crimeatl[ , ‘Avg.Day’ ]
crimeatl[ ‘Avg.Day’ ] # !!! returns df!
```

example

We are going to make a map of reported crimes, so I know where to buy a house in Atlanta.



```
> summary(crimeatl[,c("x","y")])
         x                  y
 -84.36212:  2945   33.84676:  2945
 -84.41278:  1768   33.80388:  1751
 -84.40902:   838   33.68274:   856
 -84.43276:   835   33.73812:   842
 -84.49773:   831   33.68677:   834
 (Other)  :223521   (Other)  :223510
 NA's     :    27   NA's     :    27
```



```
> class(crimeatl$x)
[1] "factor"
```

example (cont’d)

We need to convert the factor variables to numeric:

```
crimeatl$lon<-as.numeric(as.character(crimeatl$x))
crimeatl$lat<-as.numeric(as.character(crimeatl$y))
```

Hint: always go factor -> character -> numeric

example (cont'd)

```
> summary(crimeatl[, c("lon","lat")])
      lon              lat       
 Min.   :-84.55   Min.   :-84.50  
 1st Qu.:-84.43   1st Qu.: 33.73  
 Median :-84.40   Median : 33.76  
 Mean   :-84.41   Mean   : 33.75  
 3rd Qu.:-84.37   3rd Qu.: 33.78  
 Max.   :-84.29   Max.   : 33.89  
 NA's   :41       NA's   :27      
```

Dropping the outlier lat value(s)

```
crimeatl<- crimeatl[ crimeatl$lat > 33, ]
```

example (cont'd)

In order to make a map, we're going to need some functions that are stored in packages.

If this is your first time, run

```
install.packages(c("ggplot2","ggmap","ggthemes","lubridate"))
```

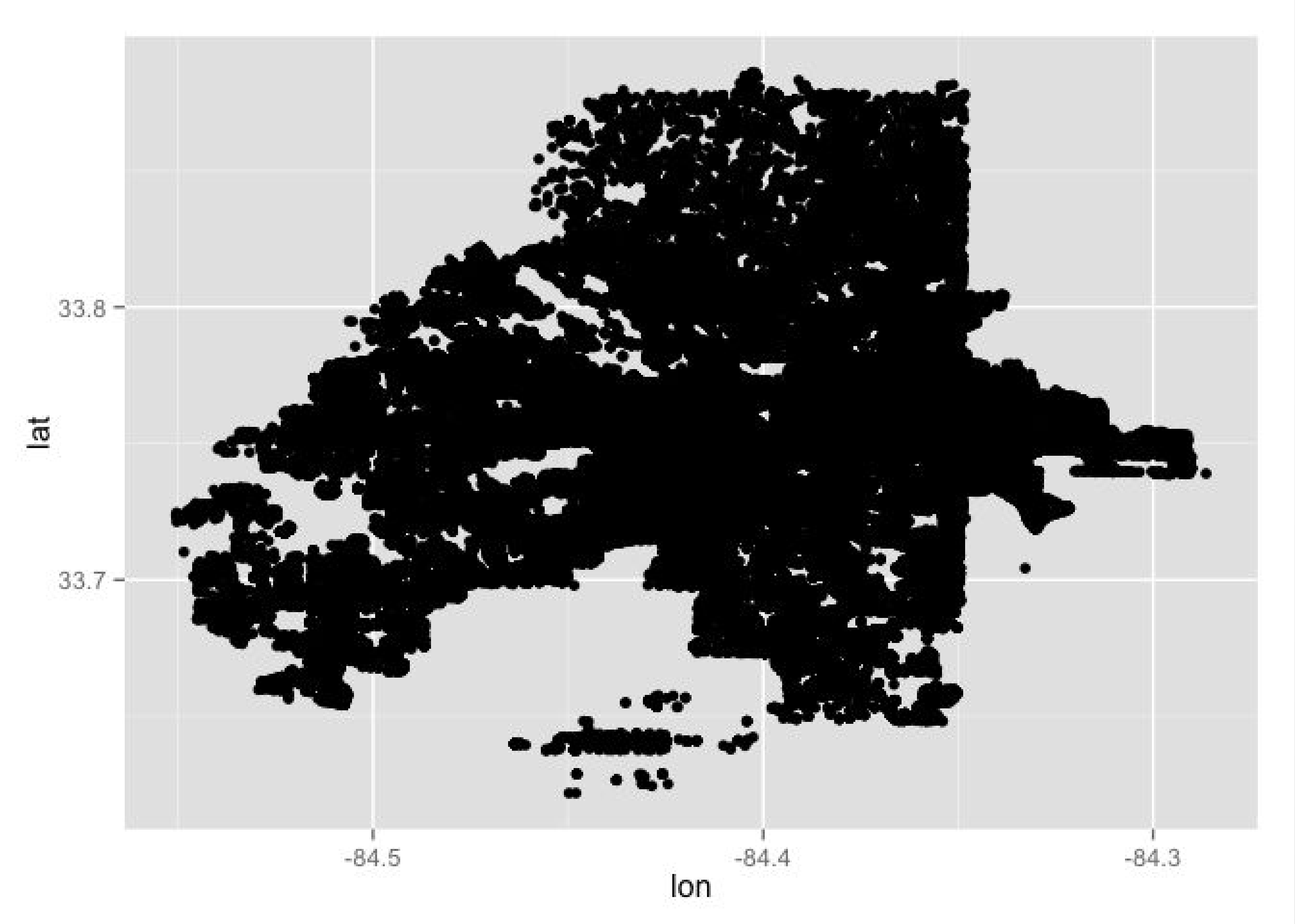
to download everything.

Then, load the packages with `library()`

```
library(ggplot2)
library(scales)
library(ggmap)
library(ggthemes)
```

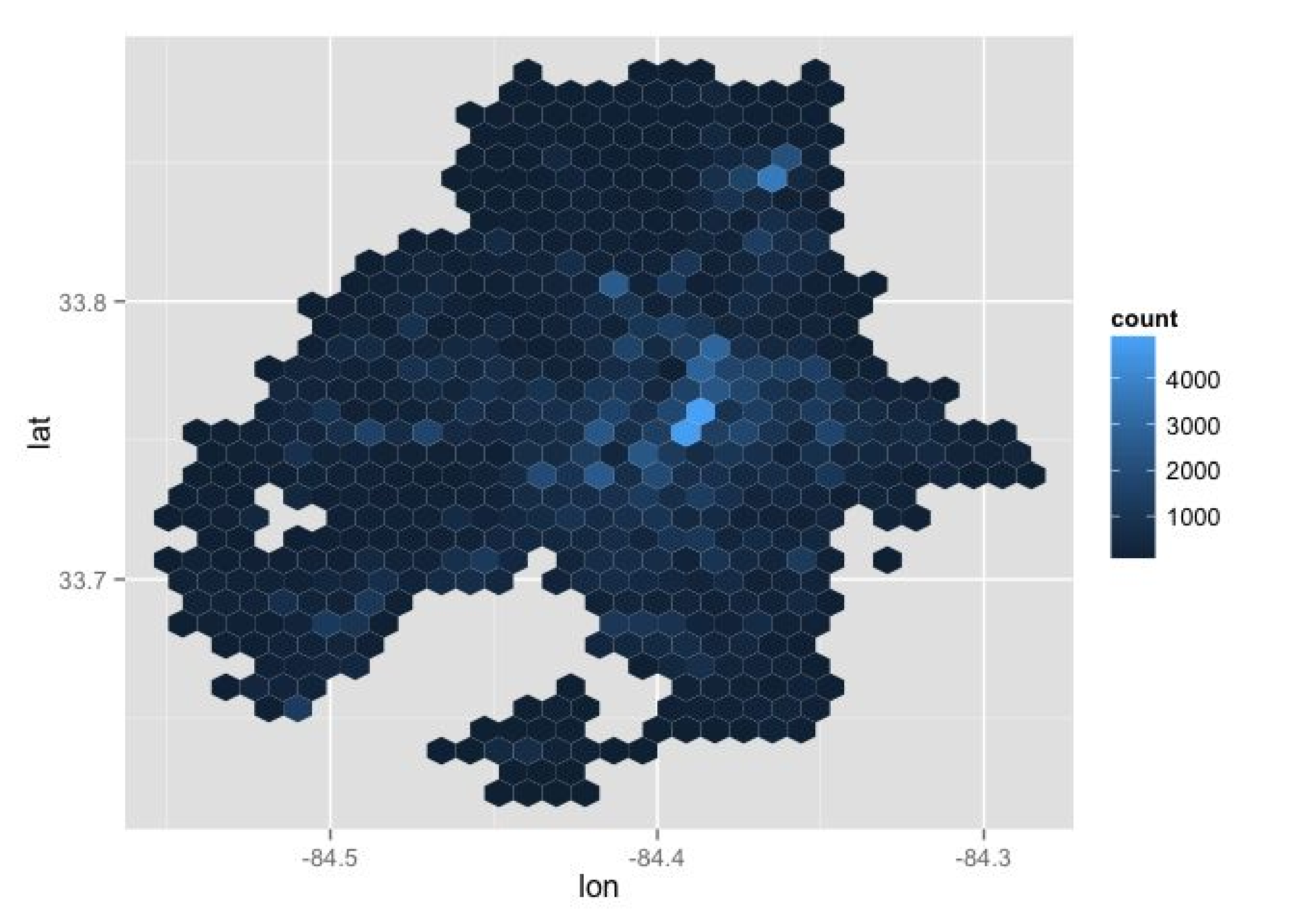
example (cont'd)

```
ggplot(data=crimeatl, aes(x=lon, y=lat)) +
    geom_point()
```



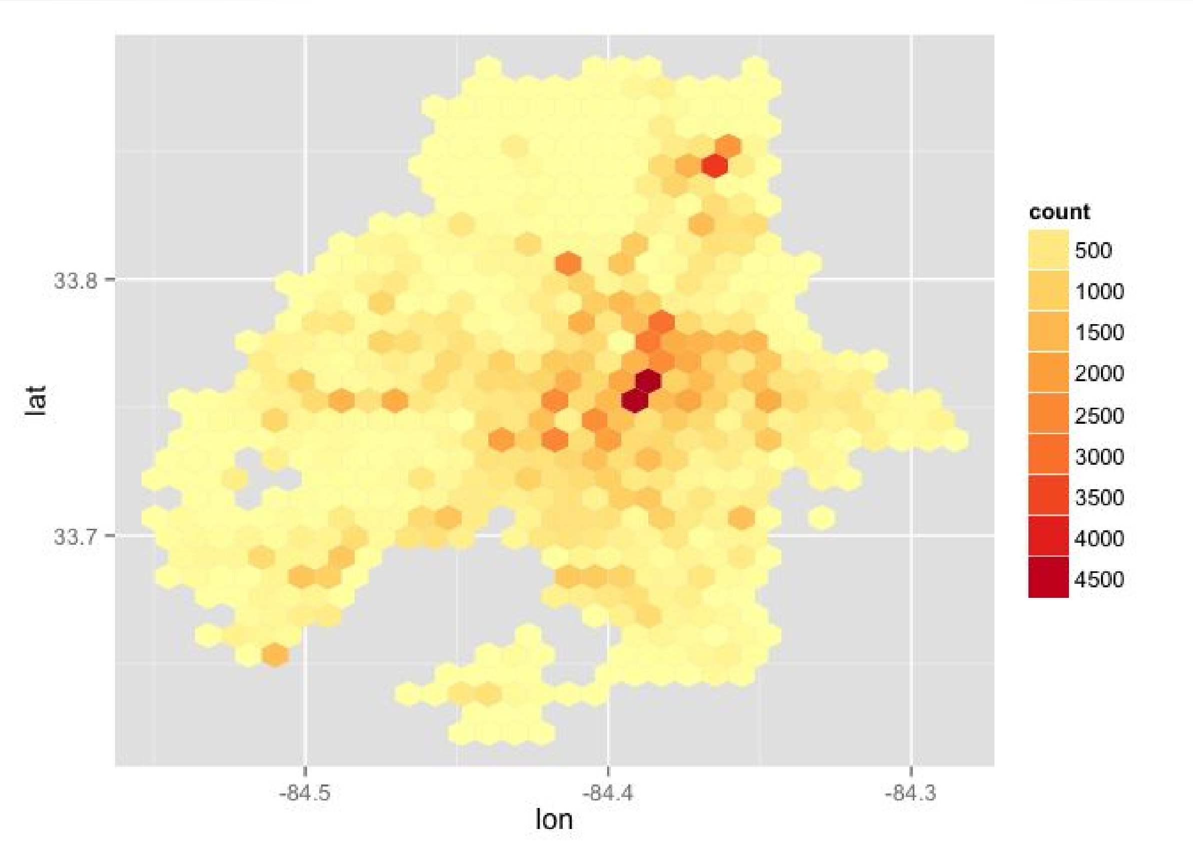
example (cont'd)

```
ggplot(data=crimeatl, aes(x=lon, y=lat)) +
    geom_hex()
```



example (cont’d)

```
ggplot(data=crimeatl, aes(x=lon, y=lat)) +
    geom_hex()+
    scale_fill_distiller(palette="YlOrRd",
    breaks=pretty_breaks(n=10))
```



example (cont'd)

We need a map to go with the data. First, we will get the bounds of the coordinates we need:

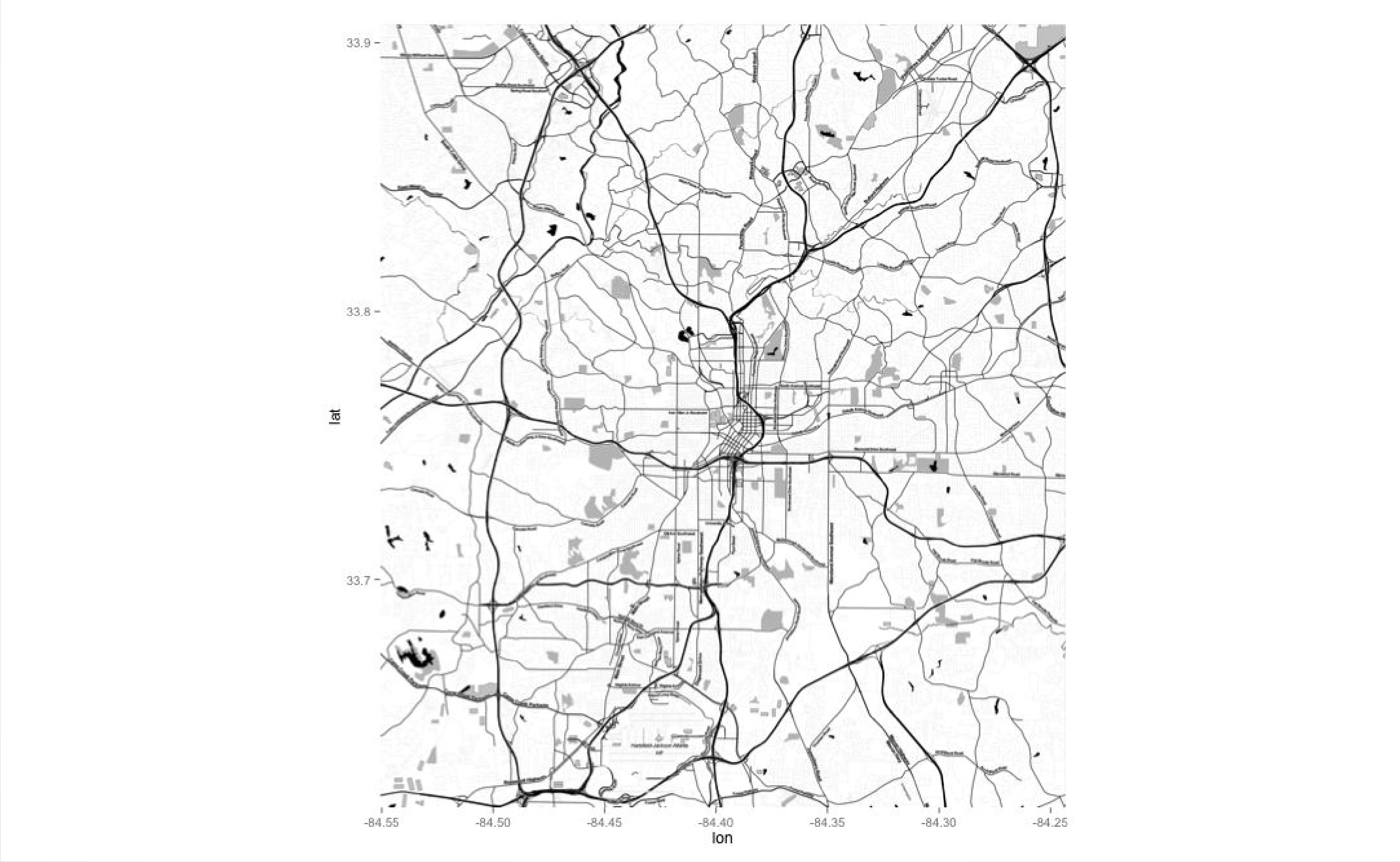
```
> r.lon <- range(crimeatl$lon, na.rm=TRUE)
> r.lat <- range(crimeatl$lat, na.rm=TRUE)
> bounds<-c(r.lon[1], r.lat[1], r.lon[2], r.lat[2])
> bounds
[1] -84.55049  33.62172 -84.28641  33.88613
```

Using the bounds, we request a map:

```
atl.map <- get_map(location=bounds, maptype = "toner",
                   zoom=13, crop=FALSE)
```

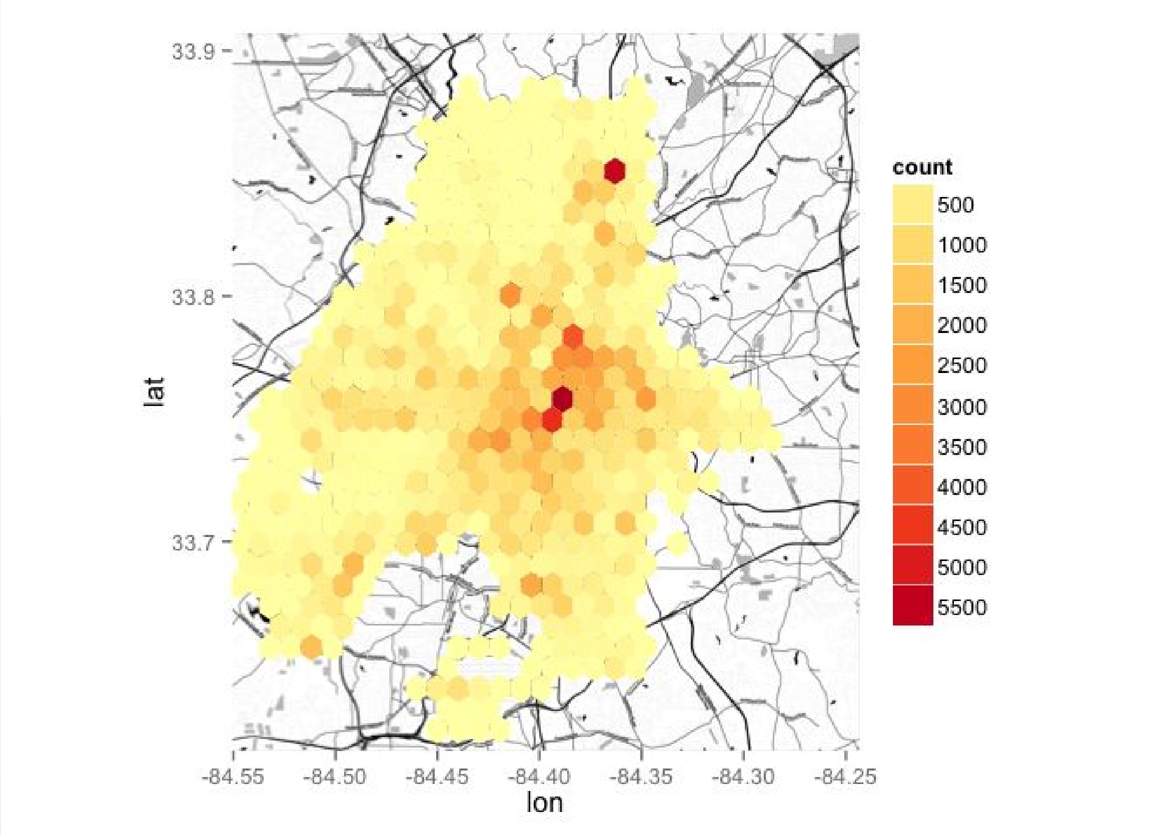
example (cont’d)

```
ggmap(atl.map)
```



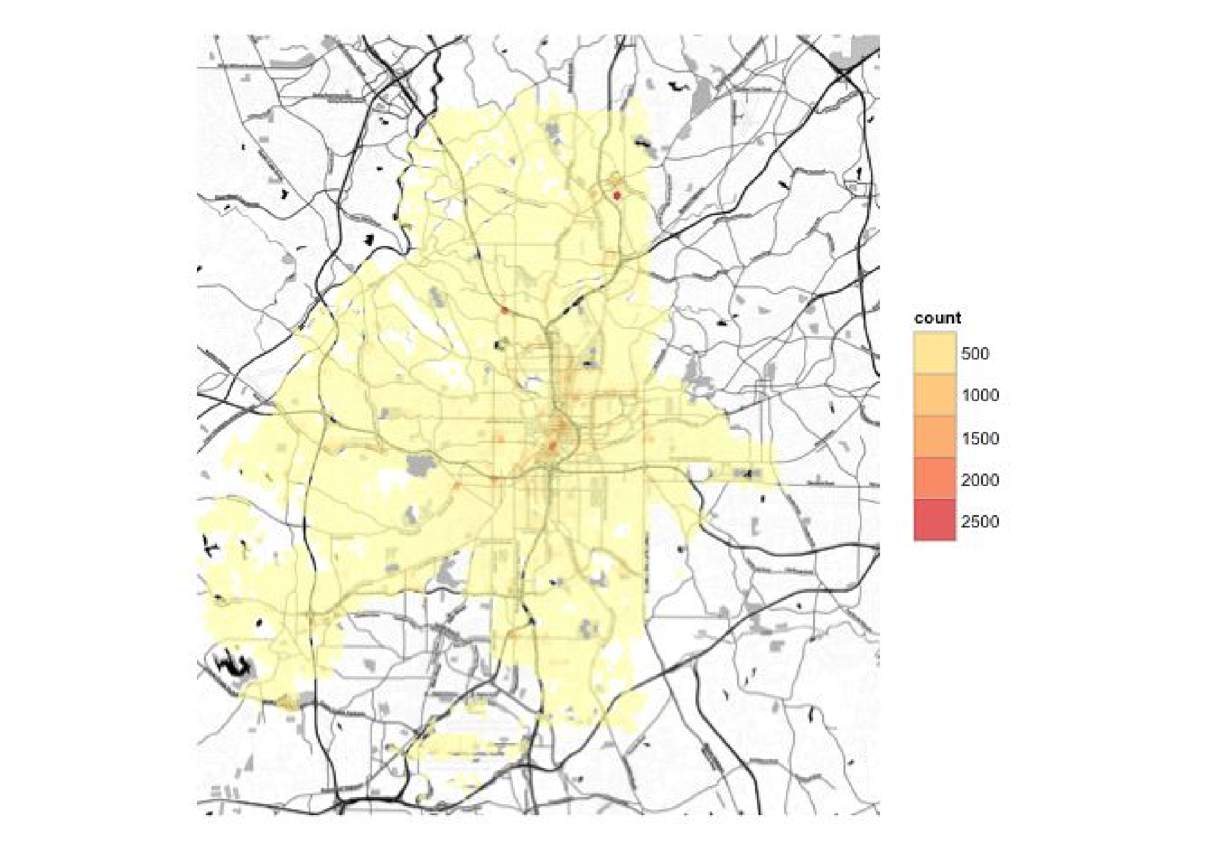
example (cont'd)

```
ggmap(atl.map) +
  geom_hex(data=crimeatl, aes(x=lon, y=lat)+
  scale_fill_distiller(palette="YlOrRd",
                       breaks=pretty_breaks(n=10))+
```



example (cont'd)

```
ggmap(atl.map) +
  geom_hex(data=crimeatl, aes(x=lon, y=lat), alpha=.7, bins=100) +
  scale_fill_distiller(palette="YlOrRd",
                       breaks=pretty_breaks(n=10))+
  theme_map()+theme(legend.position="right")
```



example (cont’d)

Suppose we’re mostly interested in homicides:

```
murderatl <- crimeatl[ crimeatl$UC2.Literal == "HOMICIDE",]
```

and we want to know when they happen

```
> library(lubridate)
> murderatl$murderdate<-mdy(murderatl$occur_date)
> murderatl$murderyear<-year(murderatl$murderdate)
>
> table(murderatl$murderyear)

2001 2006 2007 2008 2009 2010 2011 2012 2013 2014 2015
   1    2    2    2   76   88   88   82   83   92   69
```

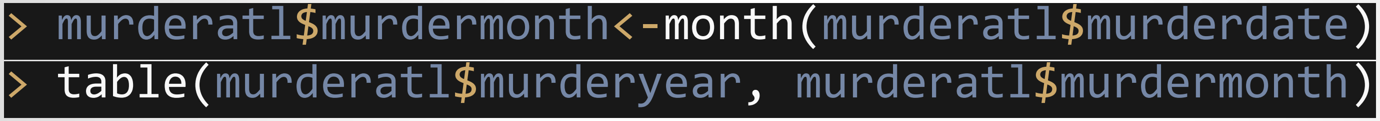
example (cont'd)

69 murders so far this year… through week 40

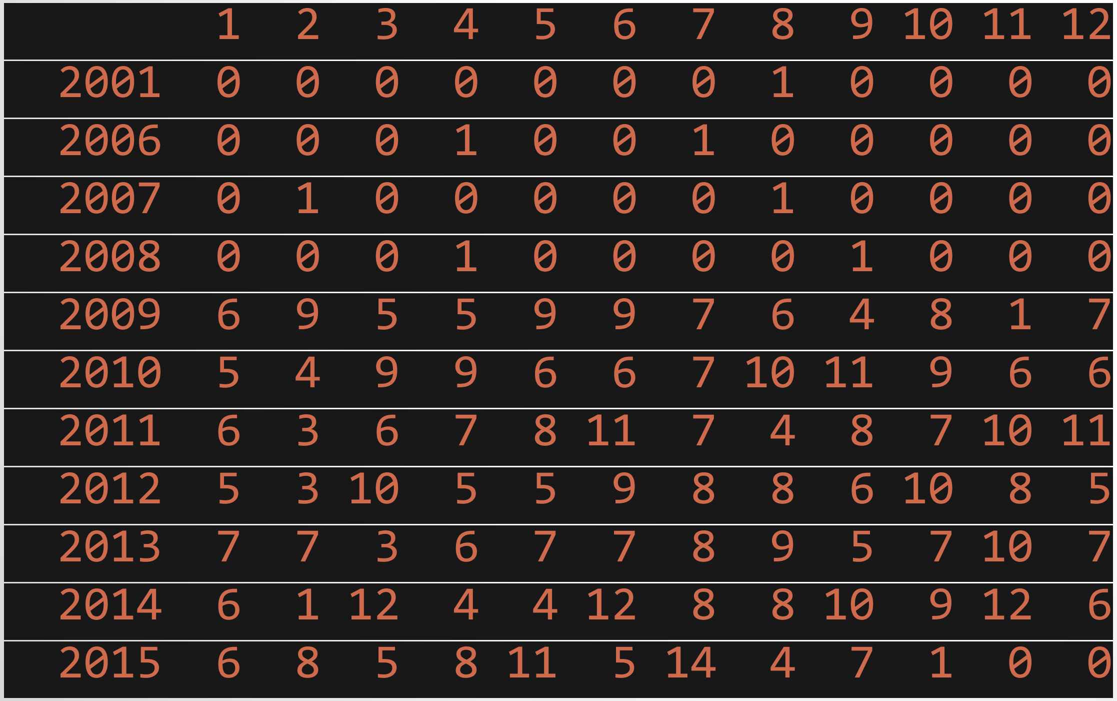
```
> table(murderatl$murderyear)
2001 2006 2007 2008 2009 2010 2011 2012 2013 2014 2015
   1    2    2    2   76   88   88   82   83   92   69

> 69/(40/52)
[1] 89.7
```

example (cont'd)

Can also look by month



```
> murderatl$murdermonth<-month(murderatl$murderdate)
> table(murderatl$murderyear, murderatl$murdermonth)
```

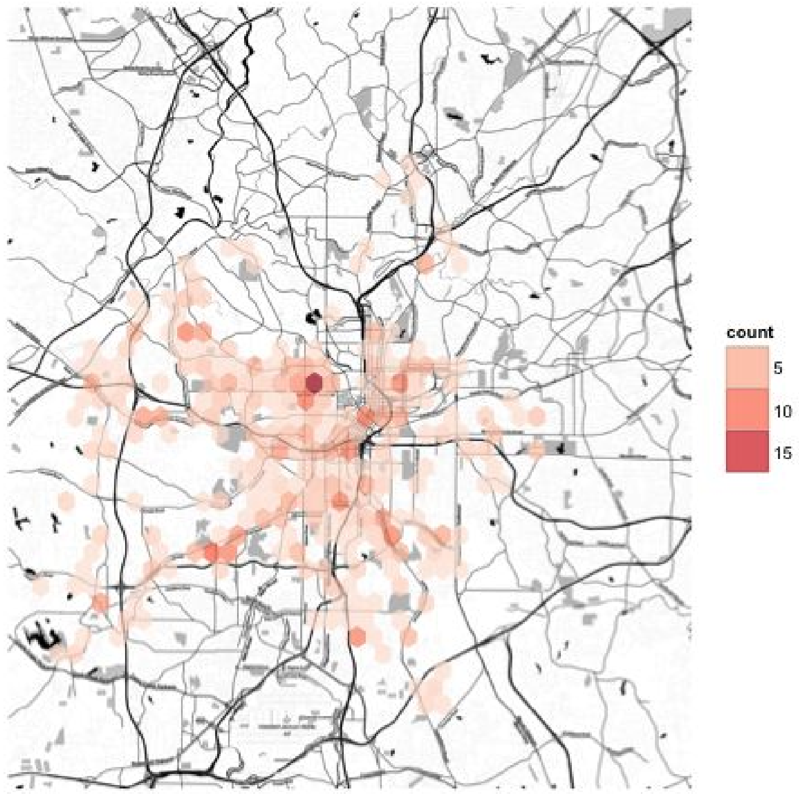


```
        1  2  3  4  5  6  7  8  9 10 11 12
  2001  0  0  0  0  0  0  0  1  0  0  0  0
  2006  0  0  0  1  0  0  1  0  0  0  0  0
  2007  0  1  0  0  0  0  0  1  0  0  0  0
  2008  0  0  0  1  0  0  0  0  1  0  0  0
  2009  6  9  5  5  9  9  7  6  4  8  1  7
  2010  5  4  9  9  6  6  7 10 11  9  6  6
  2011  6  3  6  7  8 11  7  4  8  7 10 11
  2012  5  3 10  5  5  9  8  8  6 10  8  5
  2013  7  7  3  6  7  7  8  9  5  7 10  7
  2014  6  1 12  4  4 12  8  8 10  9 12  6
  2015  6  8  5  8 11  5 14  4  7  1  0  0
```

example (cont'd)

Map of homicides only:

```
ggmap(atl.map) +
  geom_hex(data=murderatl, aes(x=lon, y=lat), alpha=.7, bins=40) +
  scale_fill_distiller(palette="Reds")+
  theme_map()+theme(legend.position="right")
```



Where to go from here?

- Practice, experiment, make mistakes

- find packages that make your work easier
  - tutorials (“vignettes”) often available

- `?help` or `help(function)`

- google your errors
  - stack exchange
  - r mailing list

The “Hadleyverse”

Hadley and his colleagues make some of the most popular packages. Some serve as elegant wrappers for existing R functionality, others make calls to functions written in C.

reshape2 & **tidyR**- reshaping data (e.g., from 'wide' to 'long')

plyr & dplyr - transform data using split-apply-combine framework

lubridate - magically work with dates, times, durations

stringr - manipulate character strings

ggplot2 - amazing graphing packages

rvest, devtools, haven… lots

Excellent resources to help along the way...

Jenny Bryan's STAT545 course materials
http://www.stat.ubc.ca/~jenny/STAT545A/current.html

R reference card, http://cran.r-project.org/doc/contrib/Short-refcard.pdf PDF cheatsheet

R Cheat Sheets: https://drive.google.com/folderview?pli=1&id=0ByIrJAE4KMTtcVBmdm1BOEZoeEk#

Quick-R, http://www.statmethods.net/ : Excellent Info on basic functions

R Twotorials, http://www.twotorials.com/: Two-minute "how-to" presentations

R Cookbook, http://wiki.stdout.org/rcookbook/: Great R info in general, especially graphics

R for SPSS & SAS users, https://sites.google.com/site/r4statistics/books/free-version:
PDF for transitioning to R
https://science.nature.nps.gov/im/datamgmt/statistics/R/documents/R_for_SAS_SPSS_users.pdf

swirl package (on CRAN and www.swirlstats.com)

R Inferno: The 9 circles of R Hell http://www.burns-stat.com/pages/Tutor/R_inferno.pdf slides and code examples:

www.github.com/mjmaenner/rdemo