How To Be a Real Data Monkey
Hacking the Infochimps API with R

Drew Conway

December 16, 2010

Introduction

What is infochimps?

- Data clearinghouse
- API

The `infochimps` R package

- Basic usage framework
- Looking at the guts

Examples

- Geo-location of blog hits
- Programming language mentions on Twitter

Infochimps.com

DISCLAIMER: I do not work for Infochimps

- I just think they're awesome
- Much more info at `http://infochimps.com/about/`

Infochimps.com

DISCLAIMER: I do not work for Infochimps

- I just think they're awesome
- Much more info at `http://infochimps.com/about/`

Mission

We make lists, spreadsheets and datasets easy to find and monkey around with.

Infochimps.com

DISCLAIMER: I do not work for Infochimps

- I just think they're awesome
- Much more info at `http://infochimps.com/about/`

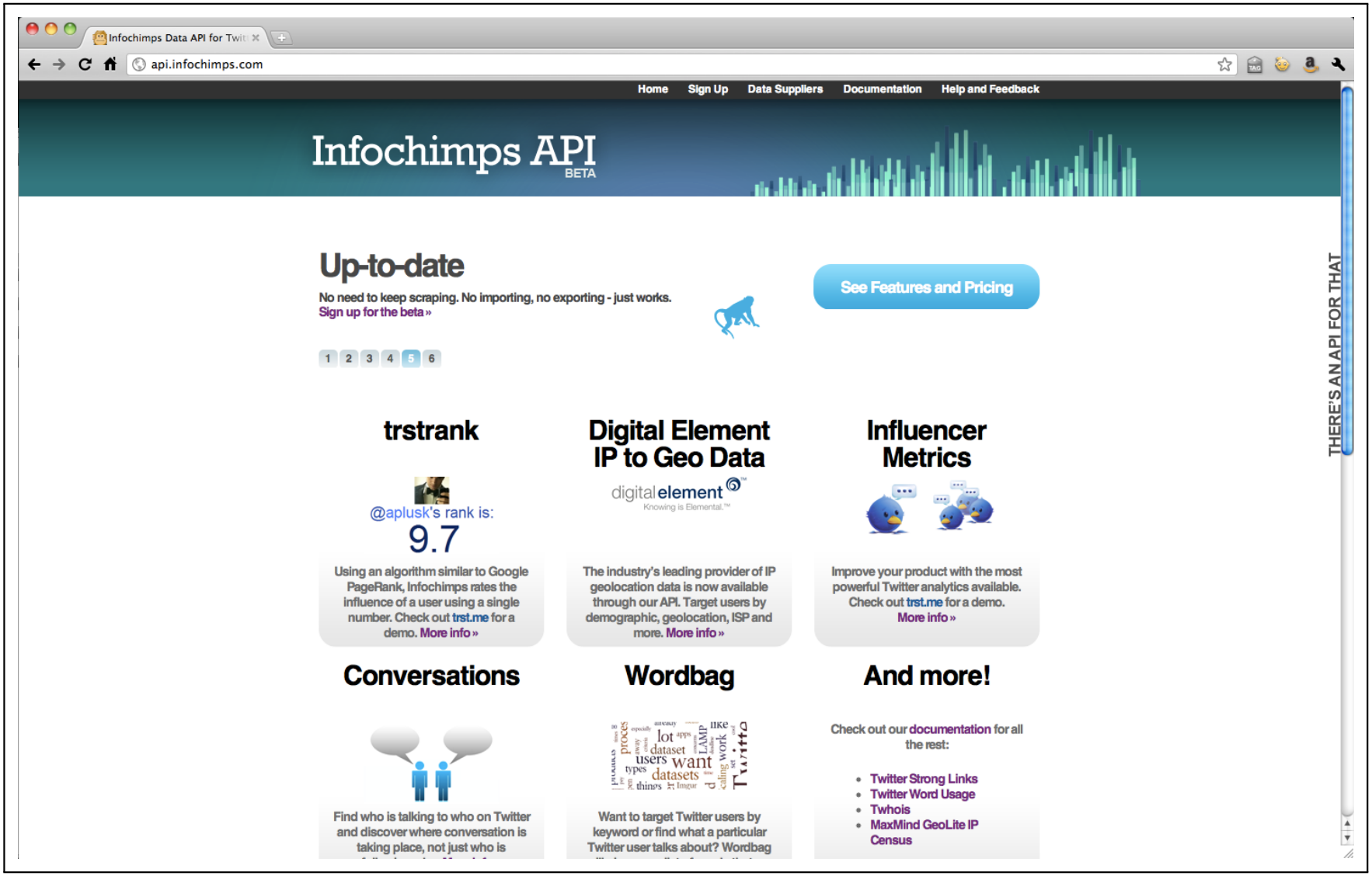
Mission

We make lists, spreadsheets and datasets easy to find and monkey around with.

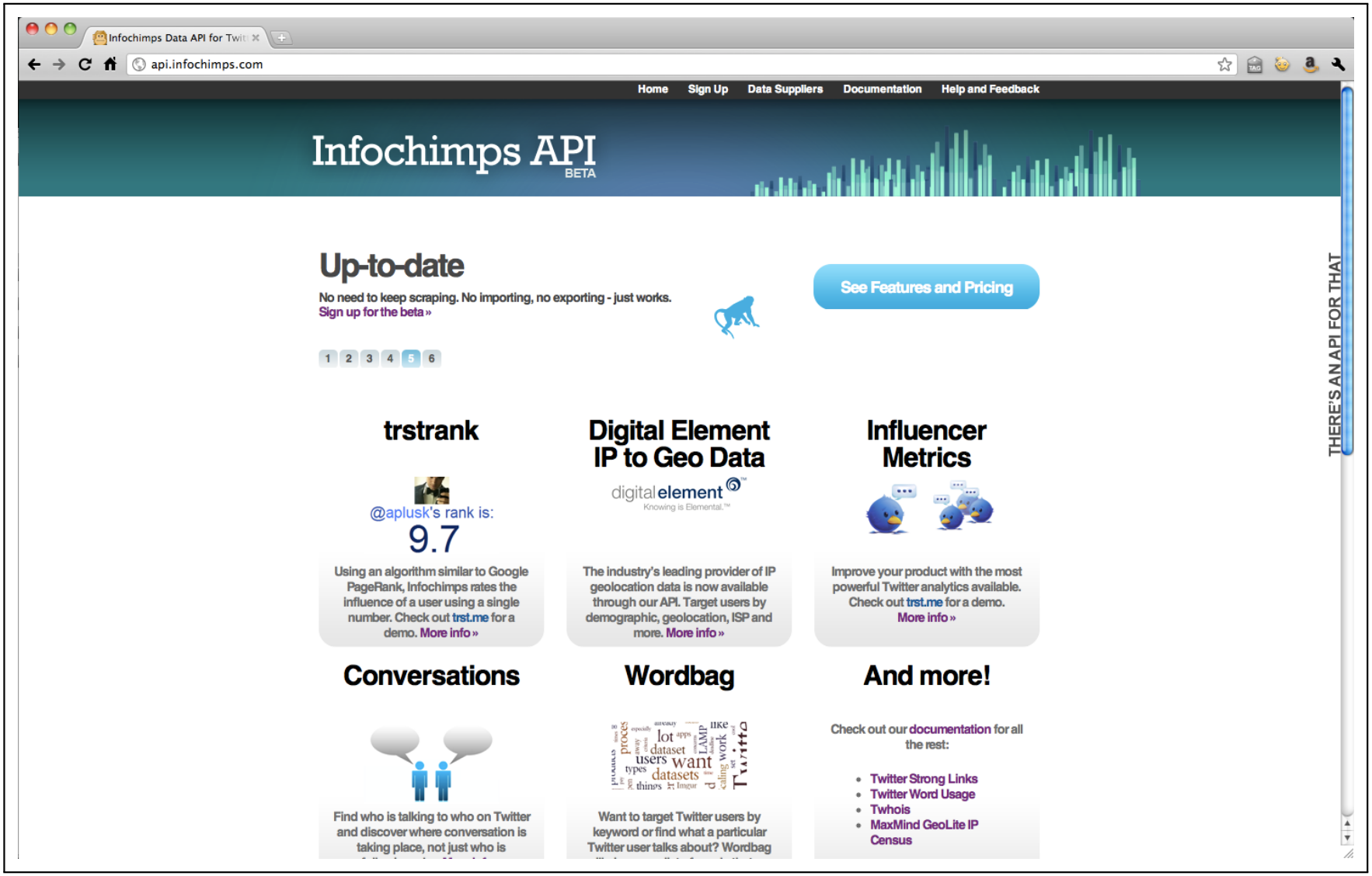
Data clearinghouse

- Buy and sell data sets
- Handle all overhead
- Many free data sets, very useful for researchers

Infochimps API



Infochimps API



For more info see vid by Flip Kromer: `http://vimeo.com/16819171`

infochimps R package

Idea: create functions for every API call to integrate querying in R

- My first package accepted to CRAN!
- http://cran.r-project.org/web/packages/infochimps/
- Update as new ones API calls roll-out

infochimps R package

Idea: create functions for every API call to integrate querying in R

- My first package accepted to CRAN!
- http://cran.r-project.org/web/packages/infochimps/
- Update as new ones API calls roll-out

Inspired by other R API wrappers

- twitteR by Jeff Gentry
- iBrokers by Jeffrey Ryan
- nytR by Shane Conway (archived)

infochimps R package

Idea: create functions for every API call to integrate querying in R

- My first package accepted to CRAN!
- http://cran.r-project.org/web/packages/infochimps/
- Update as new ones API calls roll-out

Inspired by other R API wrappers

- twitteR by Jeff Gentry
- iBrokers by Jeffrey Ryan
- nytR by Shane Conway (archived)

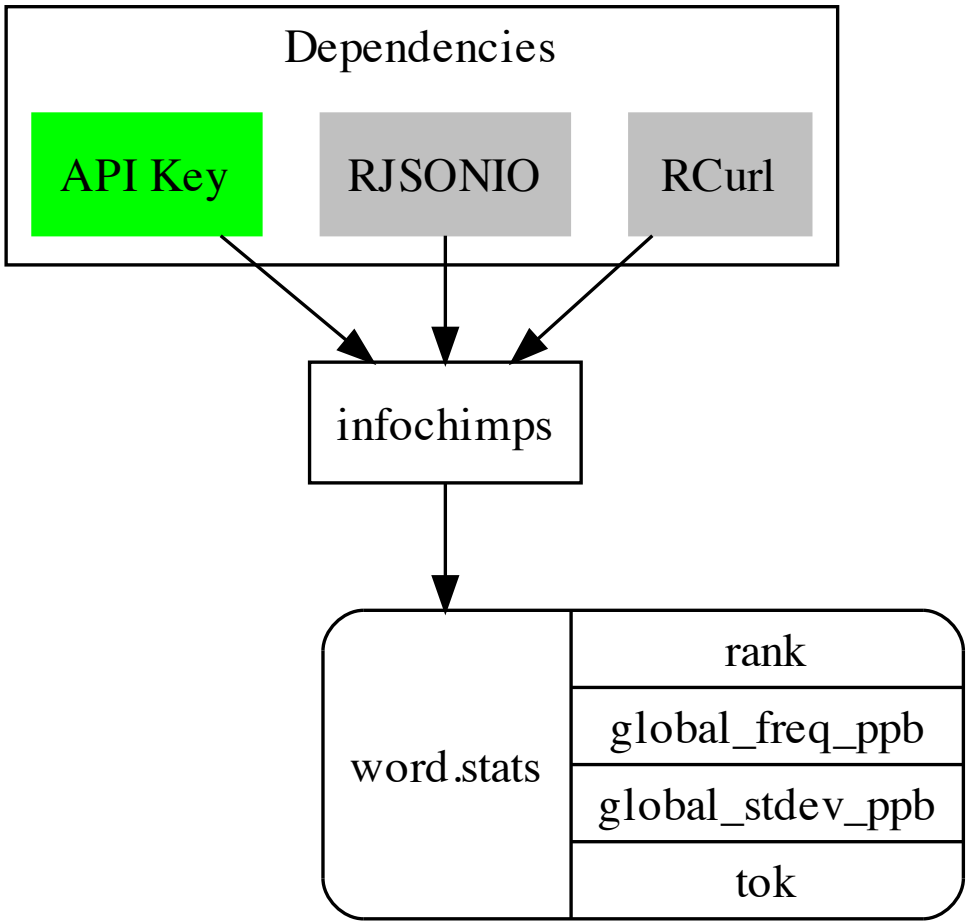
Twitter related

- conversations
- influence
- strong.links
- trstrank
- word.bag
- word.stats

Geo-location related

- census
- demographics
- domain
- ip.geo

Basic usage framework



My first infochimps call

Generate infochimps session

```
> library(infochimps)
> my.api<-''some.long.alpha.numeric''
> ic<-infochimps(my.api)
```

Get statistics for word "data"

```
> data.stats<-word.stats(''data'',ic)
> print(data.stats)
$global_stdev_ppb
[1] 2376464

$range
[1] 0.01266617

$tok
[1] ''data''

$global_freq_ppb
[1] 151562.4
```

My first infochimps call

Generate infochimps session

```
> library(infochimps)
> my.api<-''some.long.alpha.numeric''
> ic<-infochimps(my.api)
```

Get statistics for word "data"

```
> data.stats<-word.stats(''data'',ic)
> print(data.stats)
$global_stdev_ppb
[1] 2376464

$range
[1] 0.01266617

$tok
[1] ''data''

$global_freq_ppb
[1] 151562.4
```

The word.stats function

```
word.stats <-
function(tok,session) {
    word.url<-paste(session$base,"word_stats.json?tok=",
        tok,"&apikey=",session$api.key,sep="")
    word.get<-getURL(word.url)
    word.data<-fromJSON(word.get)
    # Simple error checking
    if(is.null(word.data$error)) {
        return(word.data)
    }
    else {
        warning(word.data$message[[1]])
        return(NA)
    }
}
```

The word.stats function

```
word.stats <-
function(tok,session) {
    word.url<-paste(session$base,"word_stats.json?tok=",
        tok,"&apikey=",session$api.key,sep="")
    word.get<-getURL(word.url)
    word.data<-fromJSON(word.get)
    # Simple error checking
    if(is.null(word.data$error)) {
        return(word.data)
    }
    else {
        warning(word.data$message[[1]])
        return(NA)
    }
}
```

All function follow this basic framework

Visualizing the location of blog visitors

The `ip.geo` function provides detailed geo-location data for a given IP address

- City, metro, country and continent codes (with confidence)
- Zip codes
- Latitude/Longitude
- Much more...

Visualizing the location of blog visitors

The `ip.geo` function provides detailed geo-location data for a given IP address

- City, metro, country and continent codes (with confidence)
- Zip codes
- Latitude/Longitude
- Much more...

Using web log data from `http://drewconway.com/zia`, visualize one days worth of blog hits

1. Parse log file by IP address and date/time
2. Use `ip.geo` to find lat/long for each hit
3. Plot on map using `ggplot2`

Step 1: Parse log file

```
# Load libraries
library(infochimps)
library(ggplot2)
library(maps)

# Need to load and clean the data
log.data<-read.delim("data/drewconway_com-Dec-2010.txt",
    sep="-", header=FALSE, as.is=TRUE)
log.data<-data.frame(list("IP"=log.data$V1, "Date.Time"=log.data$V3,
    "Log"=log.data$V4), stringsAsFactors=FALSE)
log.data$IP<-gsub(" ","", log.data$IP)

# First, get the dates in useable format
log.data$Date.Time<-gsub("[\\[ ]","",log.data$Date.Time)
log.data$Date.Time<-strptime(log.data$Date.Time, format="%d/%b/%Y:%H:%M:%S ")

# Filter out only those logs accessing the right blog post
log.data<-log.data[grep("(\\?p\\=|index\\.php)",log.data$Log),]
```

Step 2: Get lat/long data

```
# Create infochimps session
api.key<-"my.long.alpha.numeric"
ic<-infochimps(api.key)

# Get lattitude and longitude data for all of the IPs
ips<-unique(log.data$IP)

get.latlong<-function(ip) {
    geo.data<-ip.geo(ip,ic)
    return(c(ip, geo.data$lat,geo.data$longitude))
}

# Create data frame to merge into log data
geo.data<-lapply(ips, get.latlong)
geo.df<-as.data.frame(do.call("rbind", geo.data),stringsAsFactors=FALSE)
names(geo.df)<-c("IP","Latitude","Longitude")
log.geo<-merge(log.data,geo.df,by="IP")
log.geo$Latitude<-as.numeric(log.geo$Latitude)
log.geo$Longitude<-as.numeric(log.geo$Longitude)

# Create counts, and sort chronologically
log.count<-ddply(log.geo,.(IP, Date.Time, Latitude, Longitude),
    summarise, Count=length(Log))
log.count<-log.count[with(log.count, order(Date.Time)),]
```

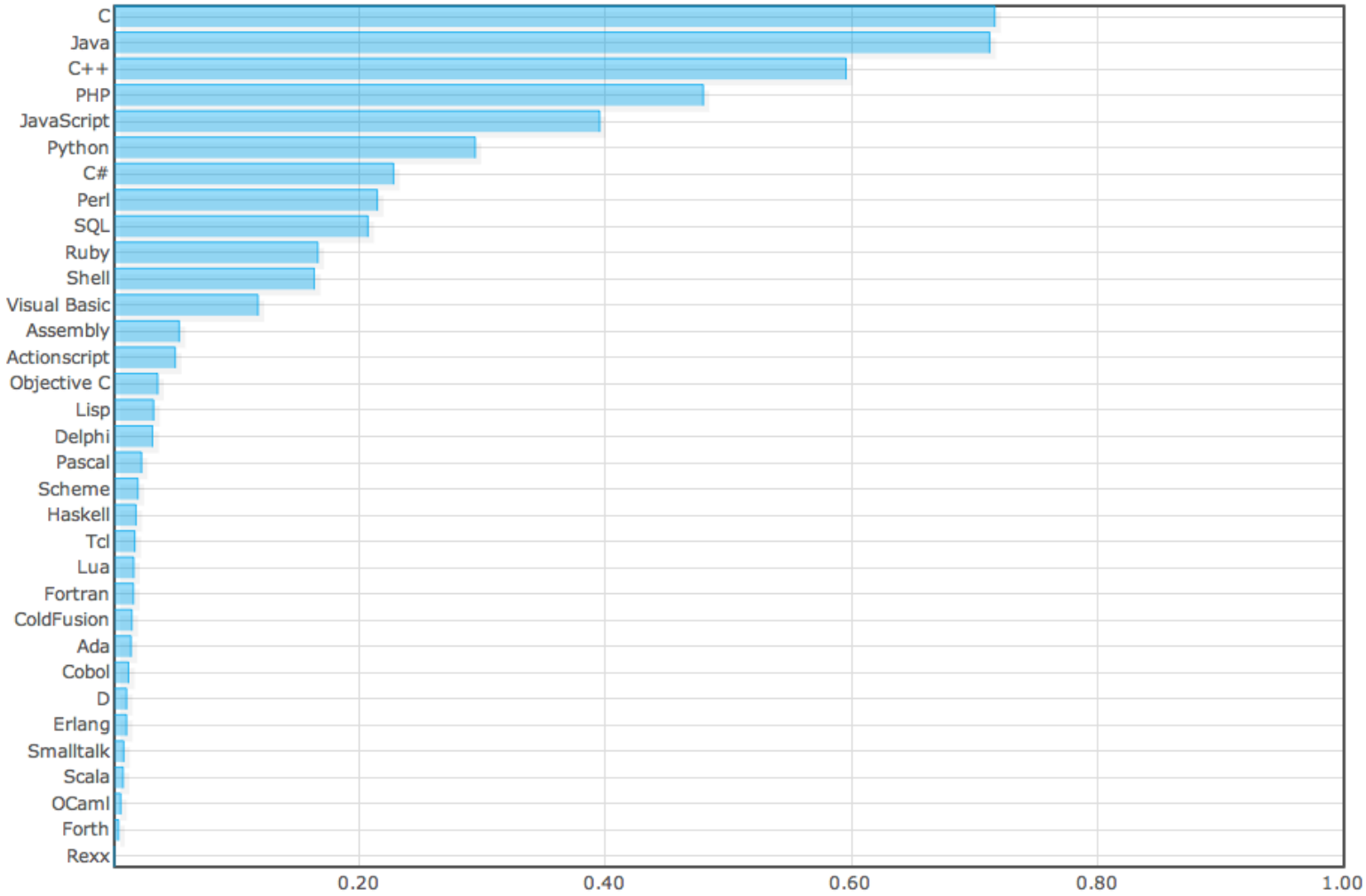
Step 3: Visualize on global map

```
# Ready to visualize
world.map<-data.frame(map(plot=FALSE)[c("x","y")])

# Create frame for every second in data
plot.num<-1
for(d in strftime(log.count$Date.Time)) {
    log.sub<-log.count[which(strftime(log.count$Date.Time)==d),]
    geo.plot<-ggplot(world.map, aes(x=x,y=y))+geom_path(aes(colour=``grey''))
    geo.plot<-geo.plot+geom_point(data=log.sub, aes(x=Longitude, y=Latitude,
        color=``red'', alpha=0.75, size=Count))+
      annotate(``text'',x=-125,y=-5,label=strftime(d, format=``%H:%M:%S''))+
      theme_bw()+scale_colour_manual(values=c(``grey''=``grey'',``red''=``red''),
        legend=FALSE)+
      scale_alpha(legend=FALSE)+scale_size(legend=FALSE)+
        coord_map(projection=``lagrange'',ylim=c(-40,70),xlim=c(-145,155))+
      opts(panel.grid.major=theme_blank(),axis.ticks=theme_blank(),
        axis.text.x=theme_blank(),axis.text.y=theme_blank())+
      xlab(````)+ylab(````)
    ggsave(plot=geo.plot,filename=paste(``images/maps/'',plot.num,``.png'',sep=````),
      width=6,height=4)
    plot.num<-plot.num+1
}

# Run this at the command-line to join the files as a movie
# ffmpeg -f image2 -r 5 -i images/maps/%d.png -b 600k blogpost.mp4
```

The movie

Blog hits map

How do people tweet about different languages?



Source: http://langpop.com/, Last updated Sat Nov 27 08:45:50

Use the word.stats function

The word.stats function returns token frequency data

- Global frequency (parts per-billion)
- Standard deviation of frequency (parts per-billion)
- Range (normalized number of unique users who have used it)

Use the word.stats function

The word.stats function returns token frequency data

- Global frequency (parts per-billion)
- Standard deviation of frequency (parts per-billion)
- Range (normalized number of unique users who have used it)

Use the global frequency to create chart for Twitter mentions

1. Create vector of computer languages
2. User word.stats to collect frequency data
3. Plot bar chat with ggplot2

Use the word.stats function

The word.stats function returns token frequency data

- Global frequency (parts per-billion)
- Standard deviation of frequency (parts per-billion)
- Range (normalized number of unique users who have used it)

Use the global frequency to create chart for Twitter mentions

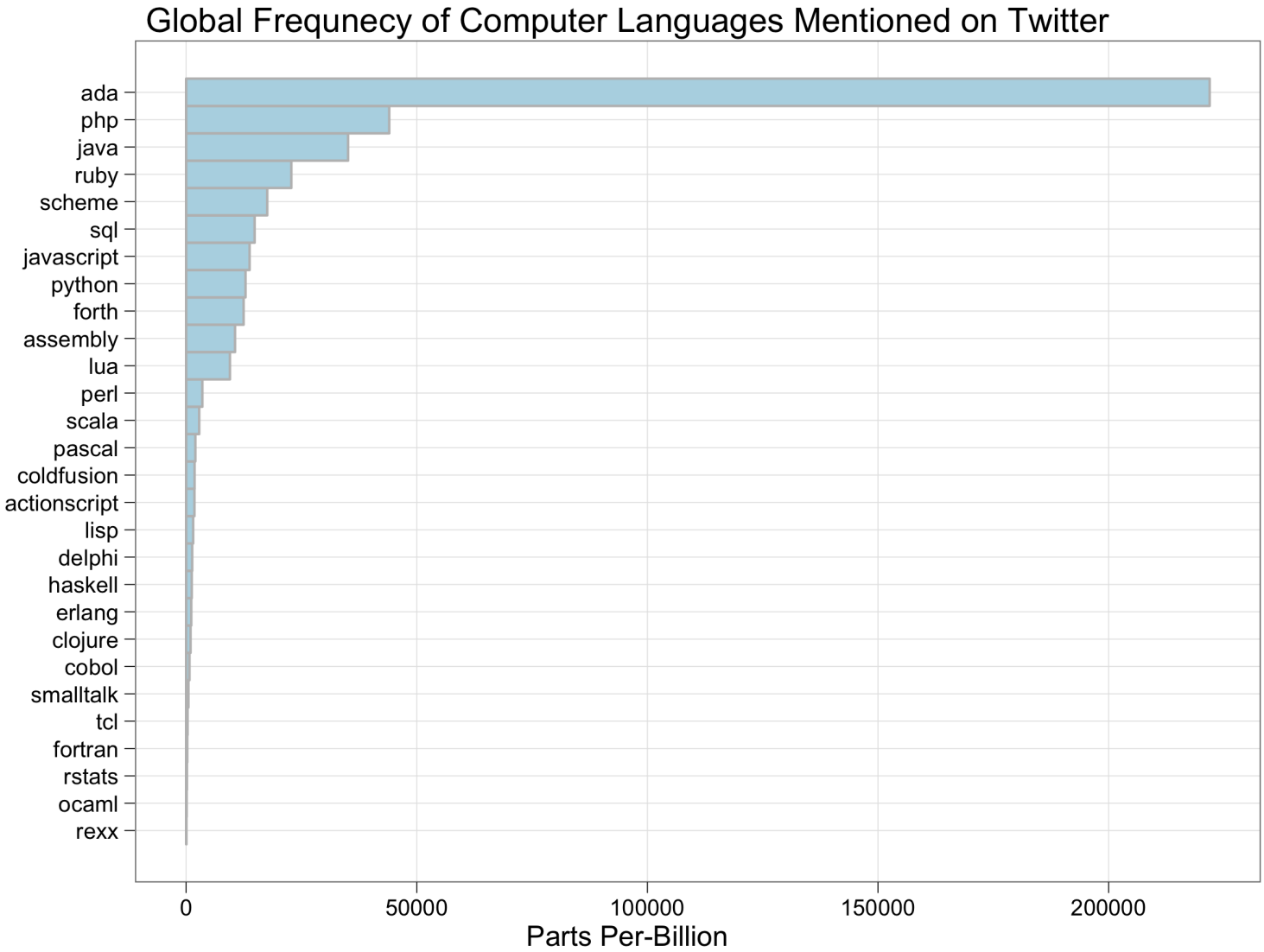
1. Create vector of computer languages
2. User word.stats to collect frequency data
3. Plot bar chat with ggplot2

Disadvantage: cannot get stats for languages like C, C, C++

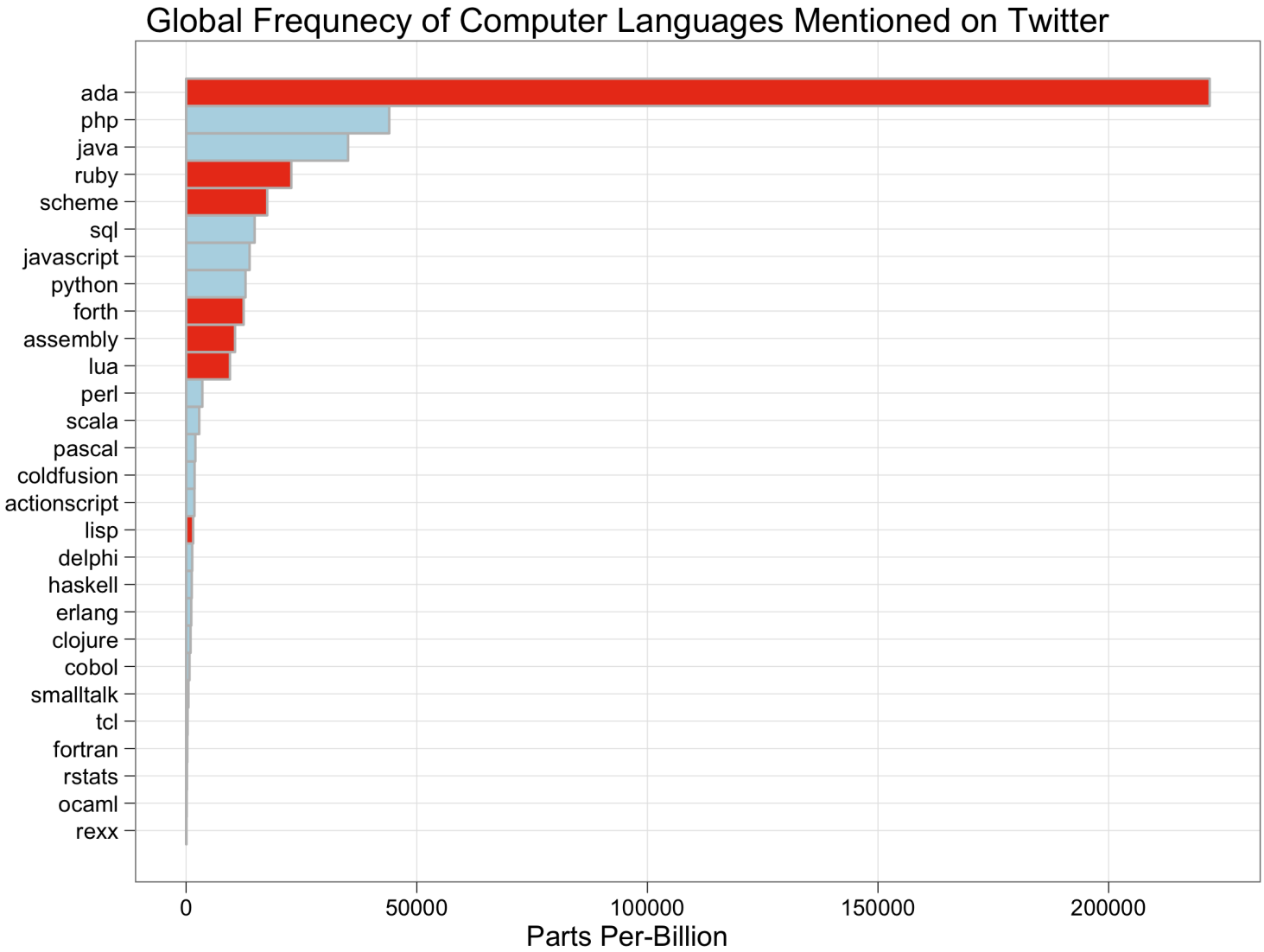
Getting the data

```
# rogramming languages
prog.langs<-c("java","php","javascript","python","sql","perl","ruby",
    "actionscript","assembly","lisp","delphi","pascal","scheme","haskell",
    "tcl","lua","fortran","coldfusion","ada","cobol","erlang","smalltalk",
    "scala","ocaml","forth","rexx","rstats")

# Get word stats for all languages
lang.stats<-lapply(prog.langs,function(t) unlist(word.stats(t,ic)))
lang.df<-as.data.frame(do.call("rbind",lang.stats), stringsAsFactors=FALSE)
lang.df$global_stdev_ppb<-as.numeric(lang.df$global_stdev_ppb)
lang.df$range<-as.numeric(lang.df$range)
lang.df$global_freq_ppb<-as.numeric(lang.df$global_freq_ppb)

# Dummy for common words
common<-rep(0,nrow(lang.df))
common[match(c("ruby","assembly","lisp","scheme","ada","forth"),lang.df$tok)]<-1
lang.df$common<-as.factor(common)

# Sort by frequency
lang.df<-lang.df[with(lang.df, order(global_freq_ppb)),]
```

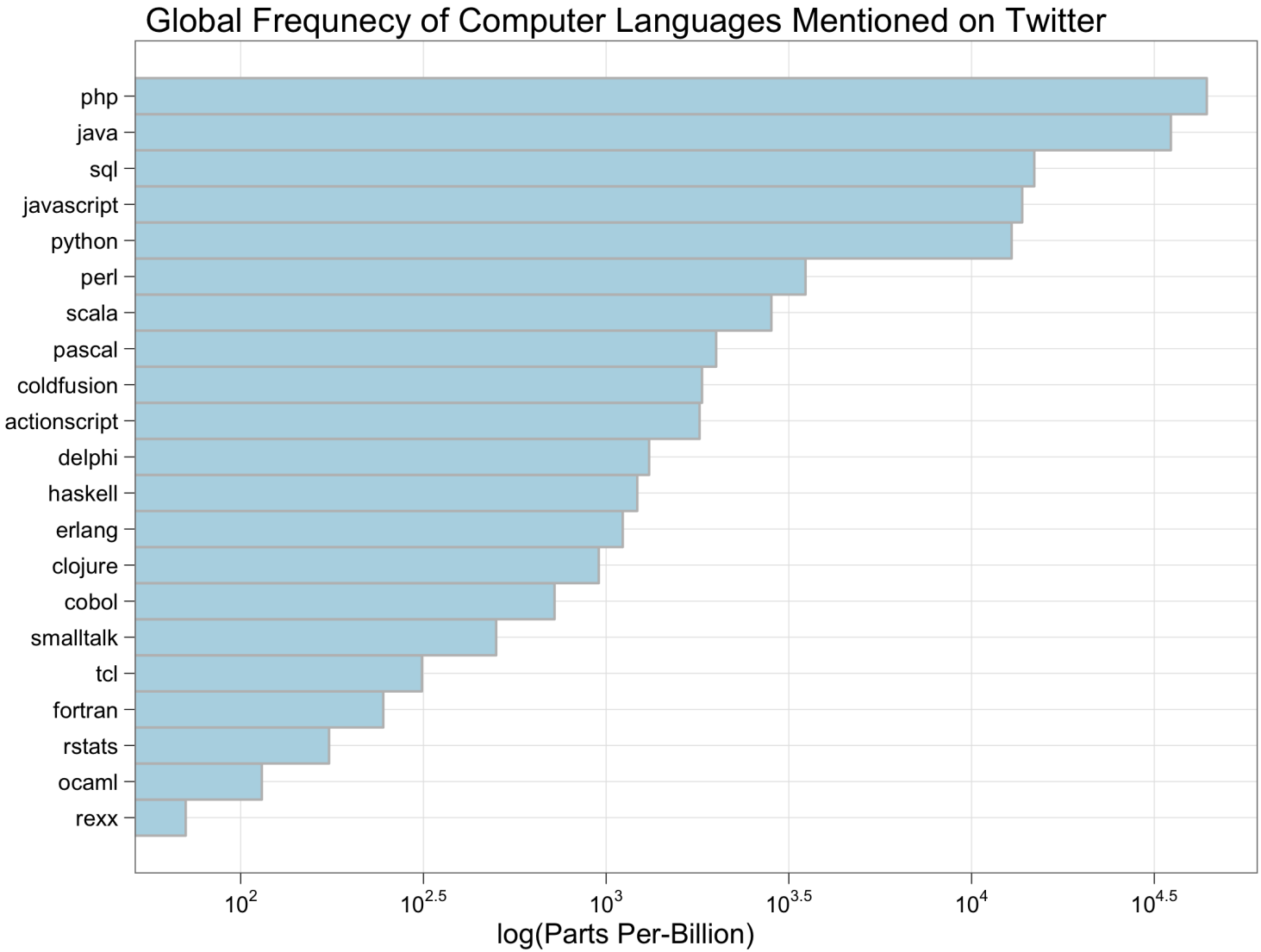
Visualizing computer languages mentioned on Twitter



Ambiguous terms



Terms cleans and logs taken



Thank You!

E-mail: drew.conway@nyu.edu
Web: http://drewconway.com/zia
Twitter: @drewconway