Automating System Data Analysis Using R

Bob Ballance

ballance@swcp.com
https://linkedin.com/in/bobballance/

November 1, 2017 / USENIX LISA 2017

Outline

Goals for Today

- Explore R interfaces to UNIX tools
- Hint at how to simplify your work-life by automating data analysis and reporting
  - Allow you and your staff to focus on the hard problems
  - Communicate effectively with users and management
  - Provide continuity of on-going analysis
- Deliver usage information about your systems in tabular and graphical forms, rolled up by system and by organization.
- Assumptions
  - The data is available, and already clean
  - The delivery vehicles include both PDF and Web

The Example

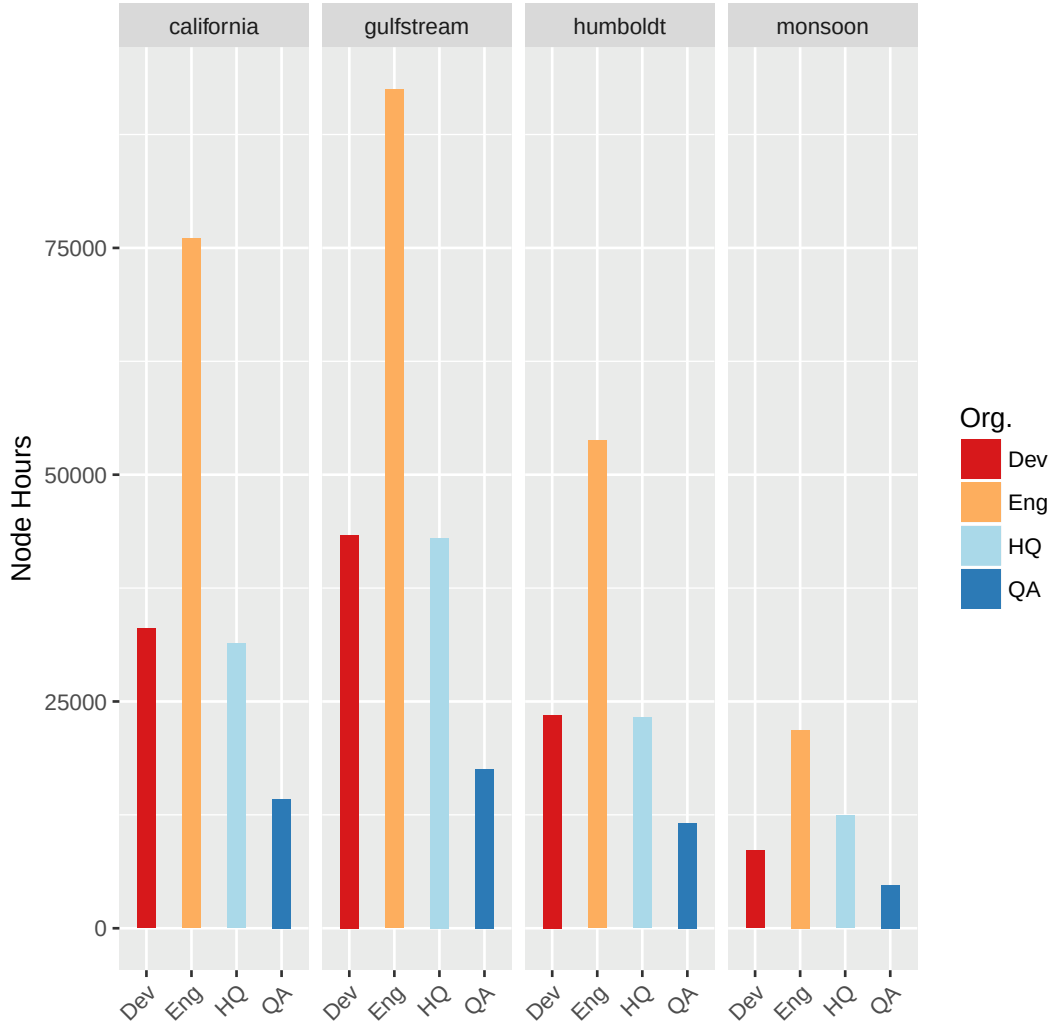
- Simulated usage data for compute clusters
  - Data generated using R, of course!
  - I did not try to simulate actual usage platforms.
- Four clusters named for ocean currents
- 20 users (generated by R)
- Four department organizations
- Data come from the future (December, 2017)

Our focus today is structure, not content!

Moving from this

```
      system    login size          start.time
1 california demosley    4 2017-12-13 18:40:49
2 california dyraibur    2 2017-12-08 18:09:14
3 california yndorado    4 2017-12-09 01:52:30
4 california jaalradw    8 2017-12-09 10:19:58
5 california niromero  128 2017-12-08 20:21:39
              end.time
1 2017-12-13 19:55:27
2 2017-12-08 19:37:45
3 2017-12-09 02:33:36
4 2017-12-09 11:02:23
5 2017-12-08 22:07:00
```

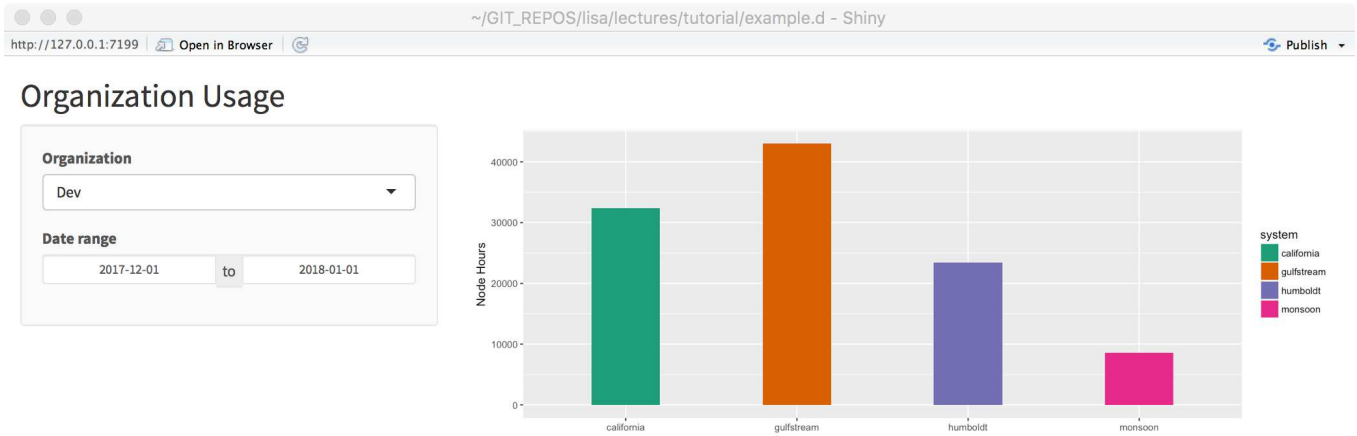
To print



Nam dui ligula, fringilla a, euismod sodales, sollicitudin vel, wisi. Morbi auctor lorem non justo. Nam lacus libero, pretium at, lobortis vitae, ultricies et, tellus. Donec aliquet, tortor sed accumsan bibendum, erat ligula aliquet magna, vitae ornare odio metus a mi. Morbi ac orci et nisl hendrerit mollis. Suspendisse

or Web

Screenshot of a simple R Shiny interface



Outline

R: A Language on a Mission

From John Chambers: in **Software for Data Analysis**

- **Mission**: *"Enable the best and most thorough exploration of data as possible"*
- **Prime Directive**: *"The computations and software for data analysis should be trustworthy: they should do what they claim, and be seen to do so."*

Reproducibility Data analysis that is not reproducible is neither good science nor good engineering. Reproducibility has the benefit that *when* you package your results in a way that they can be reproduced, you've simplified and automated *your* future task of analyzing the next pile of data.

R

- R is a statistics and data-oriented domain-specific language
- R is a functional language that also has objects
- R promotes trustworthy, reproducible, repeatable analysis

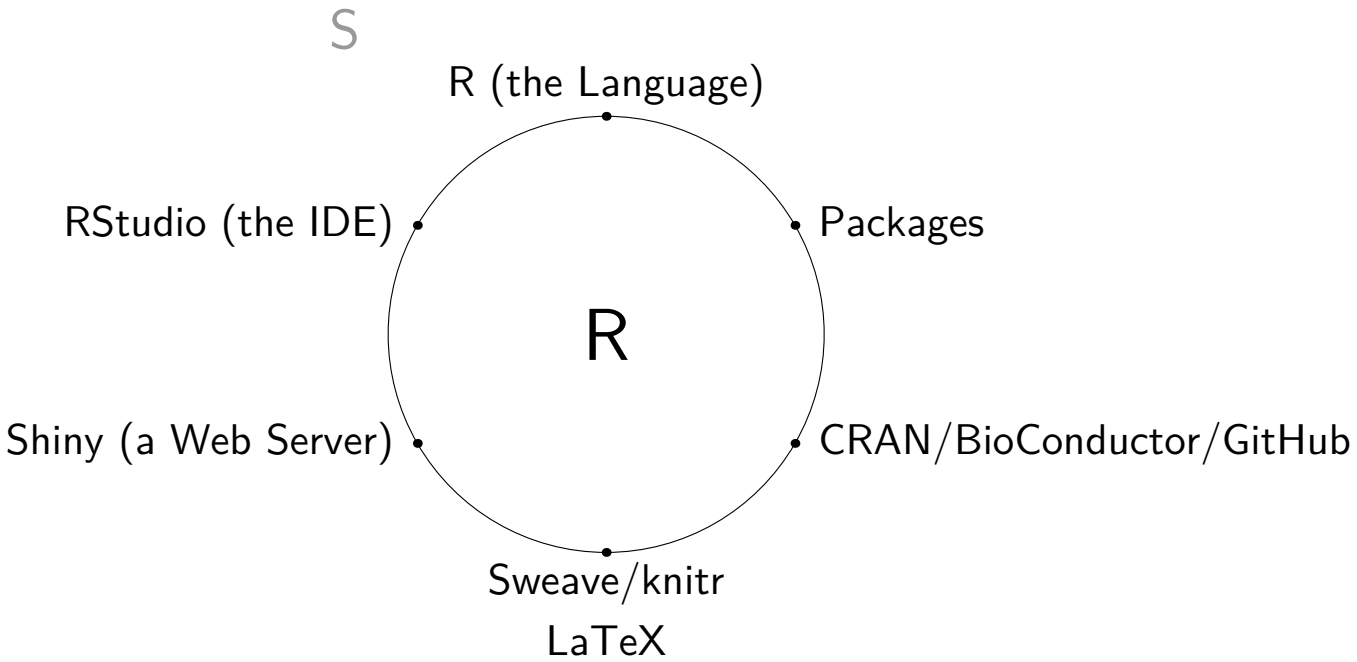
- R originated with S in the 1980's.
- R has evolved considerably and accrued both *power* and *inconsistencies*
- R can encapsulate *almost* the entire analysis workflow
  - Can be used as either an exploratory environment or a delivery platform
  - Connects to most common data sources: files, databases, URLs
  - Delivers results via scripts, documents, or web
  - Text, graphics, links, and tables can be interwoven
  - I admit to stooping to `perl` occasionally just to clean up data!

"There's more than one way to do it"

Key Features

- Data separate from computation
  - In a spreadsheet, the code and the data are all intertwingled
  - R encourages *reproducible* analyses by writing clean code for all aspects of analysis
  - Look and feel of plots or text can be controlled by writing your own functions.
- Data parallel
  - Vector operations are the norm
- Functional
  - It's common to parameterize one function with other functions
- Data-oriented data types
  - Vector
  - Factor
  - Dates
  - NA
  - Data Frame
- Packages

The R Ecosystem



The R Ecosystem . . .

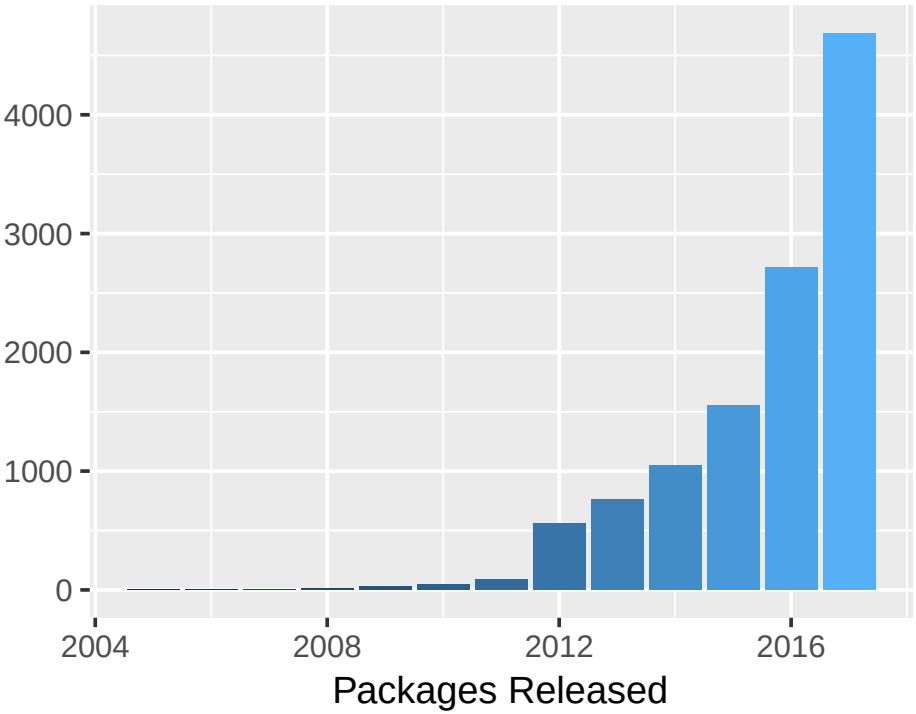
R	Programming language and environment for data analysis and statistical computing
CRAN	Comprehensive R Archive Network. Primary distribution point for R and CRAN-approved packages Other packages sites include BioConductor, R-Forge, and GitHub
R Studio	Alternative GUI for programming R, managing R projects, and R packages
Shiny	An R-friendly web server from the R Studio folks

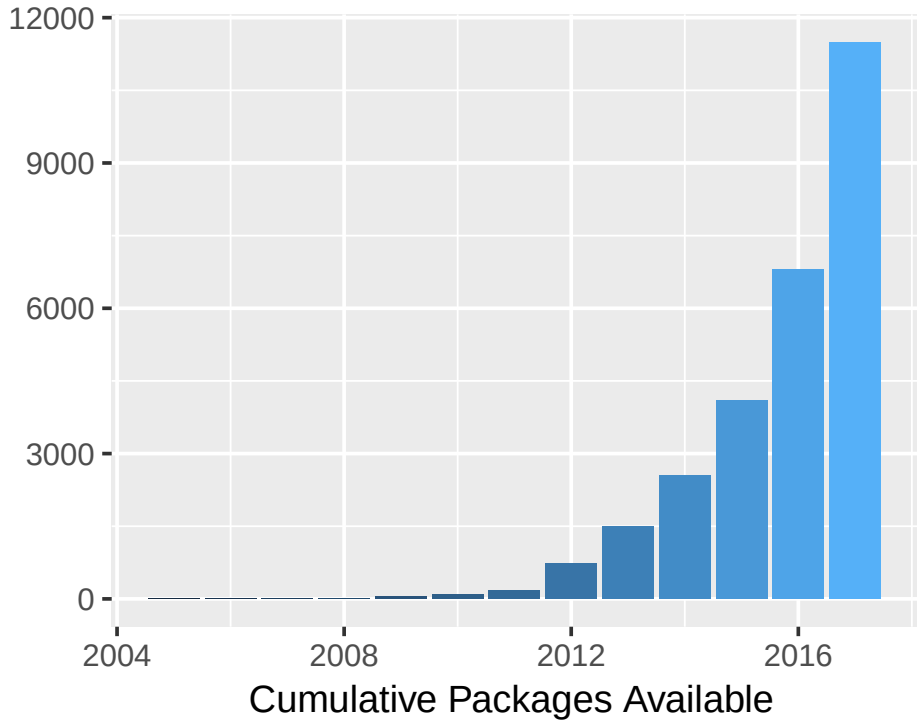
The R Ecosystem . . .

- R works with LaTeX for document integration via `Sweave`, `knitr`
  - This slide deck used `Sweave` to format the examples
  - RStudio to program the examples
  - Packages like `xtable` and `brew` ease the R+LaTeX path.
- Graphics packages export to `eps`, `png`, `pdf`, and others
- RStudio added `RMarkdown` to the document production path and `git` integration for easier project management
- The R Journal[1] Online Publication
- The R Manuals page[2]
- Online sources: blogs, stack overflow, and `CRAN` mailing lists.

[1] http://journal.r-project.org
[2] https://cran.r-project.org/manuals.html

CRAN-R



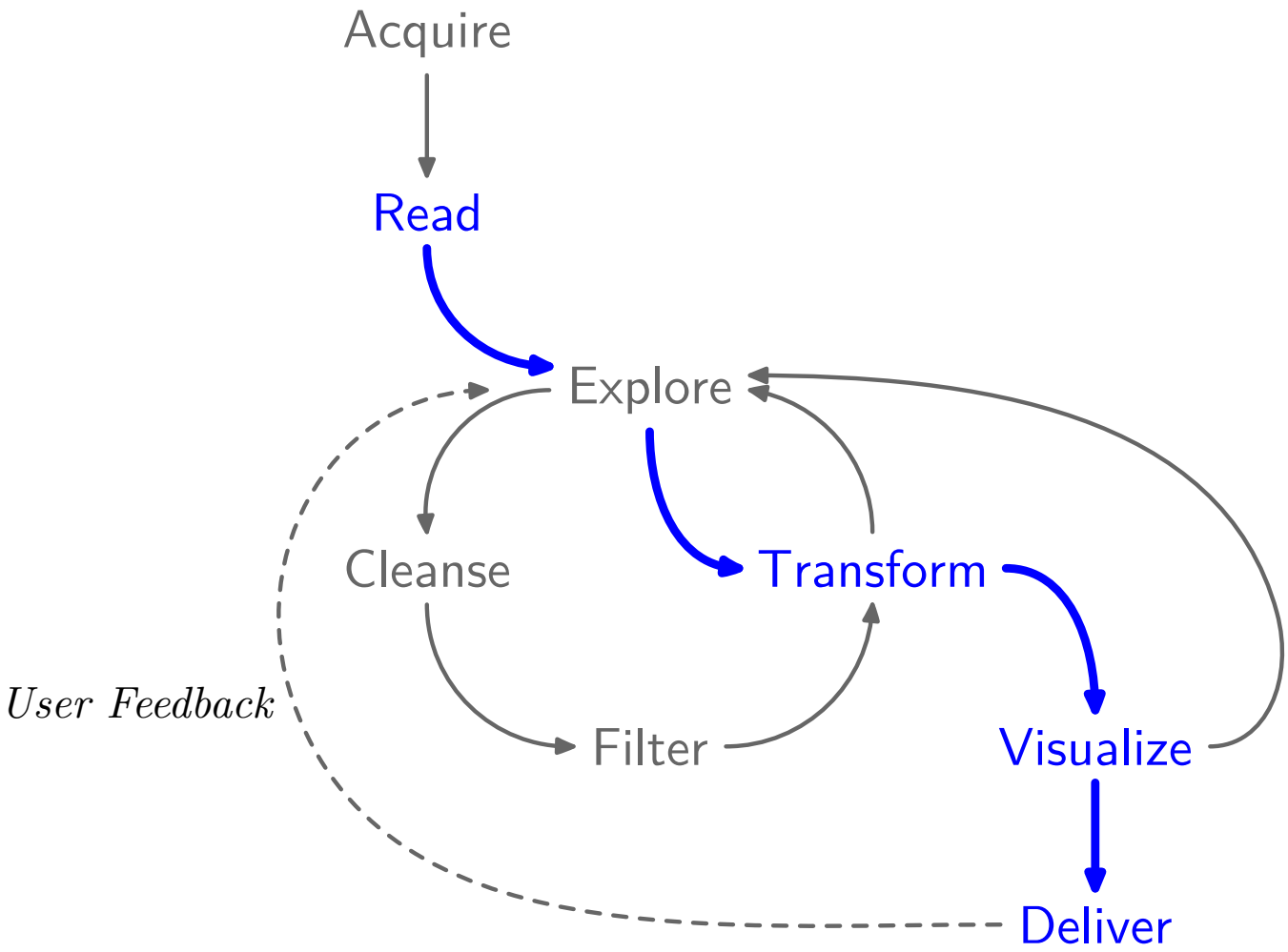


Data as of September 25, 2017

CRAN Task Views[3] allow you to browse packages by topic.

[3] http://cran.r-project.org/web/views/

The Data Science Workflow



Why not Python?

- Python is a well-known general-purpose language
  - Object-oriented
  - Large, powerful libraries for data analysis
  - Strong community
  - Comparable features
- In combination, R and Python (or PERL) form a powerful combination when you have to do a lot of data manipulation, especially text!

Outline

Acquire the Data (Non Trivial)

- **Not always trivial!**
- If *you* own the data
  - Are the right configuration parameters turned on?
  - Are the right processes running?
  - Is data being lost due to log rotation or other cleanups?
- If *someone else* owns the data
  - Are they collecting the data you need?
  - Are they paying attention to the data they are collecting?
  - Are there organizational/political/security boundaries?
  - Can you get the data in a timely manner?
  - Can you get the data repeatedly?

A look inside the database. . .

```
      system    login size          start.time
1 california demosley    4 2017-12-13 18:40:49
2 california dyraibur    2 2017-12-08 18:09:14
3 california yndorado    4 2017-12-09 01:52:30
4 california jaalradw    8 2017-12-09 10:19:58
5 california niromero  128 2017-12-08 20:21:39
              end.time
1 2017-12-13 19:55:27
2 2017-12-08 19:37:45
3 2017-12-09 02:33:36
4 2017-12-09 11:02:23
5 2017-12-08 22:07:00
```

Accessing the database

R code

```
1  fetch.usage.data <- function(end.date) {
2    query <- sprintf("SELECT * FROM usage WHERE end_time <='%s'", end.date)
3    df <- pg.fetch(query)
4
5    # Rename columns from postgres name to internal names.
6    colnames(df) <- c("system", "login", "size", "start.time", "end.time")
7    df
8  }
```

Helper functions:

- `pg.connect()` hides the details of setting up the DB connection.
- `pg.fetch()` executes a select query, based o

The helpers

R code

```
1  library(RPostgreSQL)
2  pg.fetch <- function(sql) {
3    conn <- pg.connect()
4    rs <- dbSendQuery(conn, sql)
5    df <- dbFetch(rs)
6    dbClearResult(rs)
7    dbDisconnect(conn)
8    df
9  }
10 pg.connect <- function(
11   dbname   = getOption("lisa17.database.name", Sys.getenv("PGDATABASE")),
12   username = getOption("lisa17.database.user",  Sys.getenv("PGUSER")),
13   hostname = getOption("lisa17.database.hostname", Sys.getenv("PGHOST")),
14   password = getOption("lisa17.database.password", Sys.getenv("PGPASSWORD")),
15   port = getOption("lisa17.database.port", Sys.getenv("PGPORT"))) {
16        dbConnect("PostgreSQL", dbname=dbname, host=hostname,
17                                user=username, port=port)
18 }
```

- Note the liberal use of named/default arguments.

Reading from Files

- You can read a CSV file directly into R using `read.csv()`[4]
  - The `read.csv()` function is a special case of `read.table()`
  - Both functions take a slew of useful arguments
- `read.csv()` creates and returns a `data.frame`
- The example code assigns the result to a variable named `users`
- Most R functions that read or write files allow the file to be any I/O connection, including files, pipes, and URLS.

[4] http://stat.ethz.ch/R-manual/R-devel/library/utils/html/read.table.html

Reading the user information

For today, the user information is in a flat text file.

R code

```
read.user.file <- function(filename="users.csv") {
  # The CSV files are installed as part of the package, and so the next line
  # creates the appropriate path.
  path <- path.to.package.file(filename)
  read.csv(path,
           colClasses=c("character", "factor", "factor"))
}
```

```
> users <- read.user.file()
> head(users, 3);
```

```
          fullname org    login
1   Domingo, Alexus  HQ aldoming
2    Locke, Mikayla  QA  milocke
3 Johnson, Alisiana Eng aljohnso
```

R survival kit

`data frame` A tabular data structure that works much like a table in a relational database

`tibble` Recent addition for data wrangling — a special form of data frame.

`factor` The R data type used to efficiently represent and manipulate discrete *categorical* data. For example, our users, system names, and organizations are all represented using factors.

`POSIXct` Basic timestamp data type. The start and end time of jobs are represented as `POSIXct` data values.

`c()` Primitive vector constructor

`=` or `<-` Both are assignment operations. The `=` is usually used for parameter definition.

The pipe operator %>%

R code

```
c <- f(x) %>%
        g(12) %>%
            h(47)
```

Can be used to replace sequential assignments, or nested calls

R code

```
b <- g( f(x) , 12)
c <- h(b, 47)
```

- Pipe appears in Ruby, Elixir, and is similar to the "cascading ." in Javascript
- Recent add-on to R via the `magritteR` package
- Used the `tidyverse/dplyr` world of R programming
- This slide set uses `dplyr` and pipes

Outline

Dplyr

- `dplyr` is a highly useful package for managing data transforms via chains of filters.
  - It's predecessor was `plyr`.
- Chunk tables based on some criteria, perform an action on each subset, and then recombine the result ino a new table
- Some useful `dplyr` functions

function	purpose
`group_by`	creates sub-tables
`filter`	selects rows based on criteria
`summarize`	computes new columns
`mutate`	modifies the table by adding columns, etc.
`do`	call an arbitrary function
`n()`	inside summarize, returns the number of items

Required data transformations

- Again, we've got clean data to start with, so no cleaning needed!
- The raw data contains only the `system`, `login`, and basic stats.
- Next we'll add the user's `organization` and compute the actual node hours used.
- The user's information is read from a flat file, so that we can see the `join` operation.
  - In operation, this data would also normally come from other database queries
- Finally, in some cases, we will add the system utilization as a percent of total available.

Transformation Helpers

R code

```
# Note the optional argument for users.
add.users.and.usage <- function(data.df, user.df=read.user.file()) {
  left_join(data.df, user.df, by="login") %>%
    mutate(usage= size * difftime(end.time, start.time, units="hours"))
}
```

Read

```
> start.date <- as.POSIXct("2017-12-01")                    1
> end.date <- start.date + months(1)                        2
> data <- fetch.usage.data(end.date)                        3
> head(data, 2)                                             4

     system    login size          start.time
1 california demosley    4 2017-12-13 18:40:49
2 california dyraibur    2 2017-12-08 18:09:14
            end.time
1 2017-12-13 19:55:27
2 2017-12-08 19:37:45

> users <- read.user.file();                                1
> head(users, 2)                                            2

        fullname org    login
1 Domingo, Alexus  HQ aldoming
2  Locke, Mikayla  QA  milocke
```

Transform

```
> final.data <- data %>% add.users.and.usage(users)                    1
> head(final.data, 3)                                                  2

     system    login size          start.time
1 california demosley    4 2017-12-13 18:40:49
2 california dyraibur    2 2017-12-08 18:09:14
3 california yndorado    4 2017-12-09 01:52:30
            end.time        fullname org          usage
1 2017-12-13 19:55:27 Mosley, Destiny Dev 4.975556 hours
2 2017-12-08 19:37:45  Raiburn, Dylan Dev 2.950556 hours
3 2017-12-09 02:33:36  Dorado, Yngwie  HQ 2.740000 hours
```

Explore

```
> # how many jobs per system?                                          1
> # We're now calling functions in the `dplyr` package!               2
> final.data %>% group_by(system) %>% summarize(jobs = n())           3
```

```
# A tibble: 4 x 2
      system  jobs
       <chr> <int>
1 california  2763
2 gulfstream  3418
3   humboldt  1998
4    monsoon   899
```

Explore

```
> # how many jobs per user?
> # We're now calling functions in the `dplyr` package!
> final.data %>%
+   group_by(login) %>%
+     summarize(jobs = n()) %>%
+       head(n=4)
```

```
# A tibble: 4 x 2
     login  jobs
     <chr> <int>
1 aldoming   474
2 aljohnso   462
3 alsotode   454
4 brdaniel   435
```

Explore

```
> final.data %>%                                          1
+   group_by(system, org) %>%                             2
+       summarize(jobs = n()) %>% head(n=3)               3

# A tibble: 3 x 3
# Groups:   system [1]
      system    org  jobs
       <chr> <fctr> <int>
1 california    Dev   574
2 california    Eng  1374
3 california     HQ   533
```

Usage functions

- There's a pattern here, and so it is best to wrap it up in code

R code

```
1 rollup.usage <- function(df) {
2   df %>%
3     group_by(system, org, login) %>%
4       summarize(usage=sum(usage))
5 }
6
```

Usage vs. Utlization

- But we really want to know the system utilization
- For that, we need the size of the system, which is defined in a package option

```
> # Grab the option, and show only the system and size columns
> getOption("lisa17.systems")[, c("system", "nodes")]
```

```
      system nodes
1   humboldt   256
2    monsoon   128
3 gulfstream   288
4 california   256
```

R code

```
add.utilization <- function(df, start.date, end.date) {
  sys.data <- getOption("lisa17.systems")[,c("system", "nodes")]
  # Convert factors to strings, to avoid warnings
  sys.data$system <- as.character(sys.data$system)
  df$system <- as.character(df$system)
  df %>%
    left_join(sys.data, by="system") %>%
    mutate(utilization=as.numeric(usage)/
                        (nodes * hours.in.range(start.date, end.date)))
}
```

Final

```
> rollup.usage(report.data) %>%                                          1
+     add.utilization(start.date, end.date) %>% head(n=3)                2

# A tibble: 3 x 6
# Groups:   system, org [1]
   system    org    login          usage nodes utilization
    <chr> <fctr>    <chr>         <time> <dbl>       <dbl>
1 monsoon    Dev chlockwo 1983.794 hours   128  0.02083117
2 monsoon    Dev demosley 2622.746 hours   128  0.02754060
3 monsoon    Dev dyraibur 2118.753 hours   128  0.02224833
```

Outline

Intro

dplyr uses pipes to transform data

R code

```r
df = data %>%
          f() %>%
             g() %>% h()
```

ggplot builds up visualizations by adding elements

R code

```r
viz <- ggplot(data=df) +
        geom & aesthetics +
        scales +
        layout +
        style + ...

# viz is a recipe (object), not yet rendered.
viz <-  viz + theme...

# print(viz) or ggsave(viz) then renders the graphic
#    (and other functions will also render....)
```

GGPLOT: A grammar of graphics

- ggplot2 implements a "Grammar of Graphics[5]"
- Each layer in the chart has a `geom()` , an aesthetic `aes()`, a `stat()`, a `scale()` and other attributes
- ggplot2 overloads the + operator in order to add new plot elements
- ggplot2 is a *deep* and *powerful* package that we can only touch upon here!

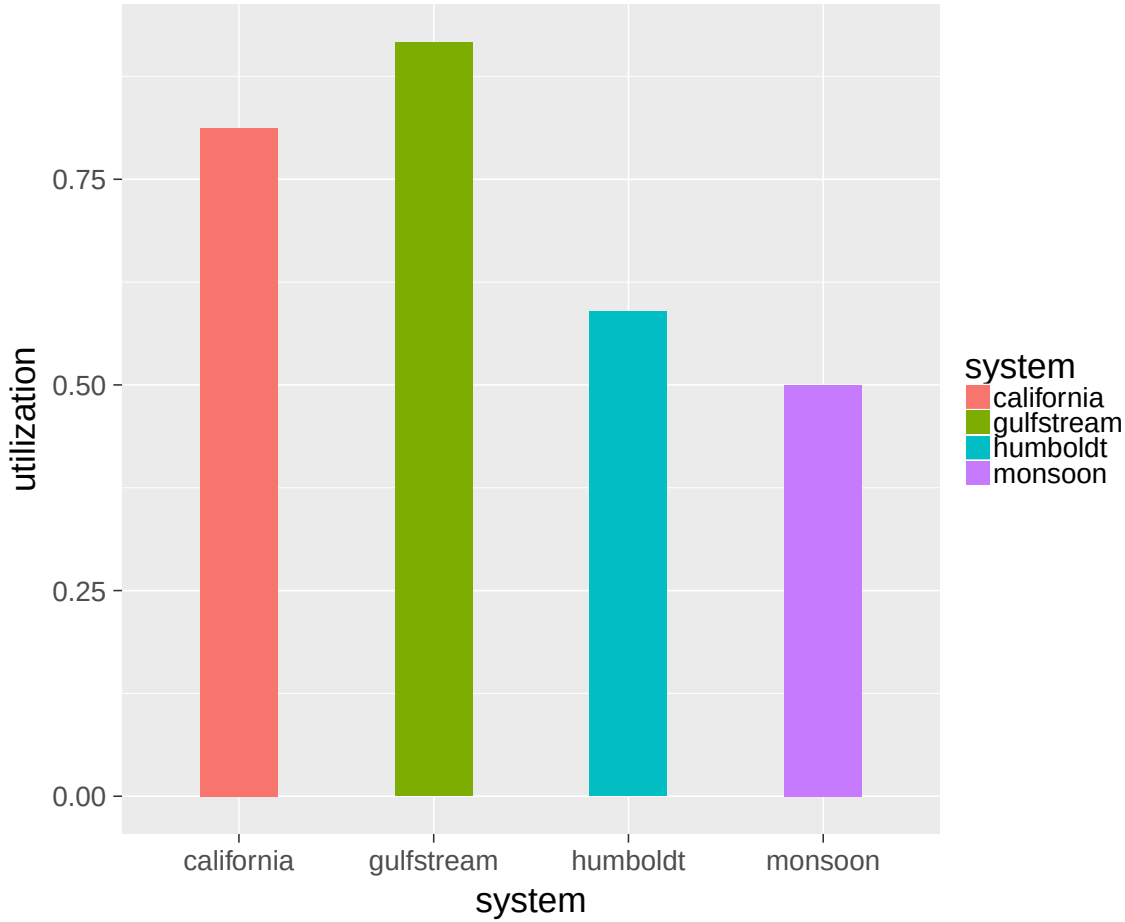
R code

```
#' @param d is a data.frame with columns `system` and `utilization`
#' @return a ggplot object
bare.barchart <- function(d) {
  ggplot(data=d) +
    geom_bar(aes(x=system, y=utilization, fill=system),
             width=.4,
             stat="identity")
}
```

[5] http://www.amazon.com/The-Grammar-Graphics-Statistics-Computing/dp/0387245448

Generating a bare chart

```
> report.data %>%                                          1
+    rollup.system.usage(start.date, end.date) %>%        2
+       bare.barchart()                                   3
```



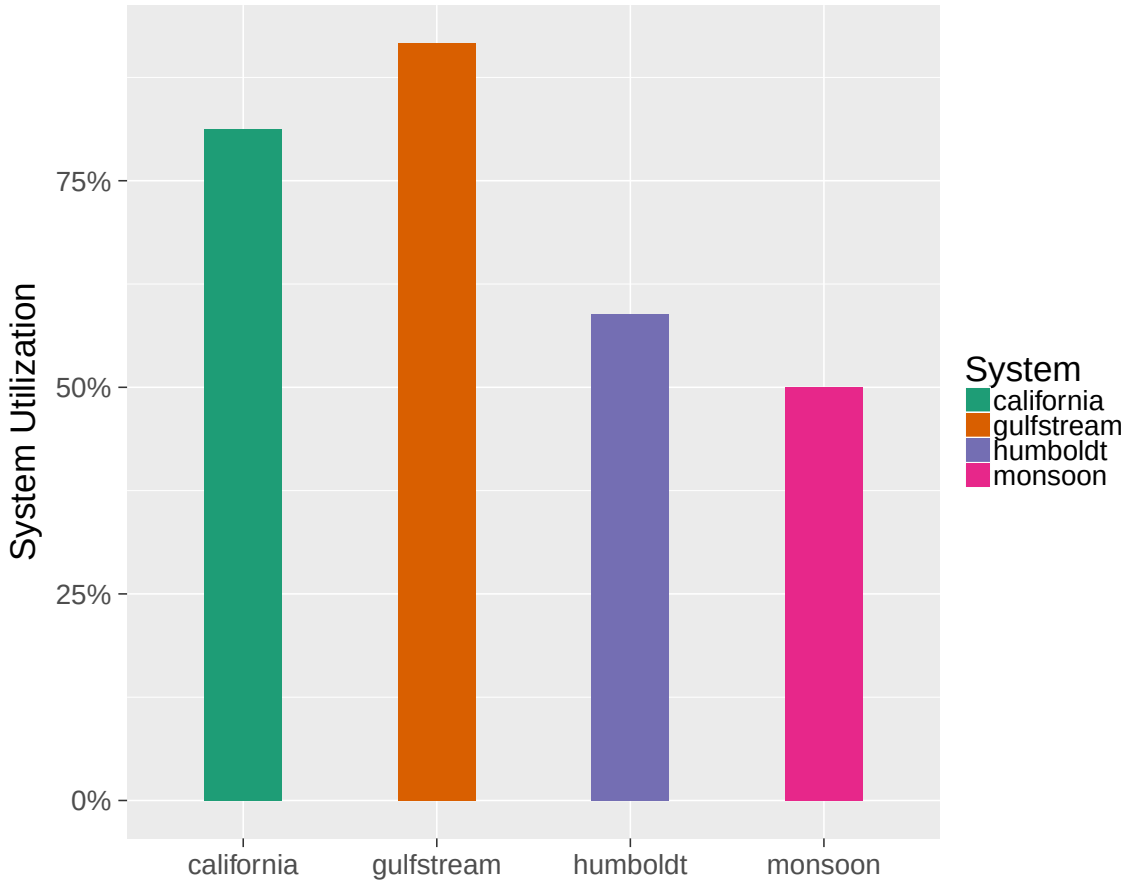
Make the chart nicer

R code

```
#' @param d is a data.frame with columns `system` and `utilization`
#' @return a ggplot object
#'
#' The call to `bare.barchart` is for the purposes of the course
system.barchart <- function(d) {
  bare.barchart(d) +
    scale_fill_brewer(name="System", palette="Dark2") +
    scale_y_continuous(labels=scales::percent) +
    xlab("") +
    ylab("System Utilization")
}

org.barchart <- function(d) {
  ggplot(data=d) +
    geom_bar(aes(x=org, y=as.numeric(usage), fill=org),
             width=.4, stat="identity") +
    scale_fill_brewer(name="Org", palette="RdYlBu") +
    xlab("") +
    ylab("Node Hours")
}
```

Nicer chart

```
> report.data %>%                                                    1
+         rollup.system.usage(start.date, end.date) %>%              2
+             system.barchart()                                      3
```

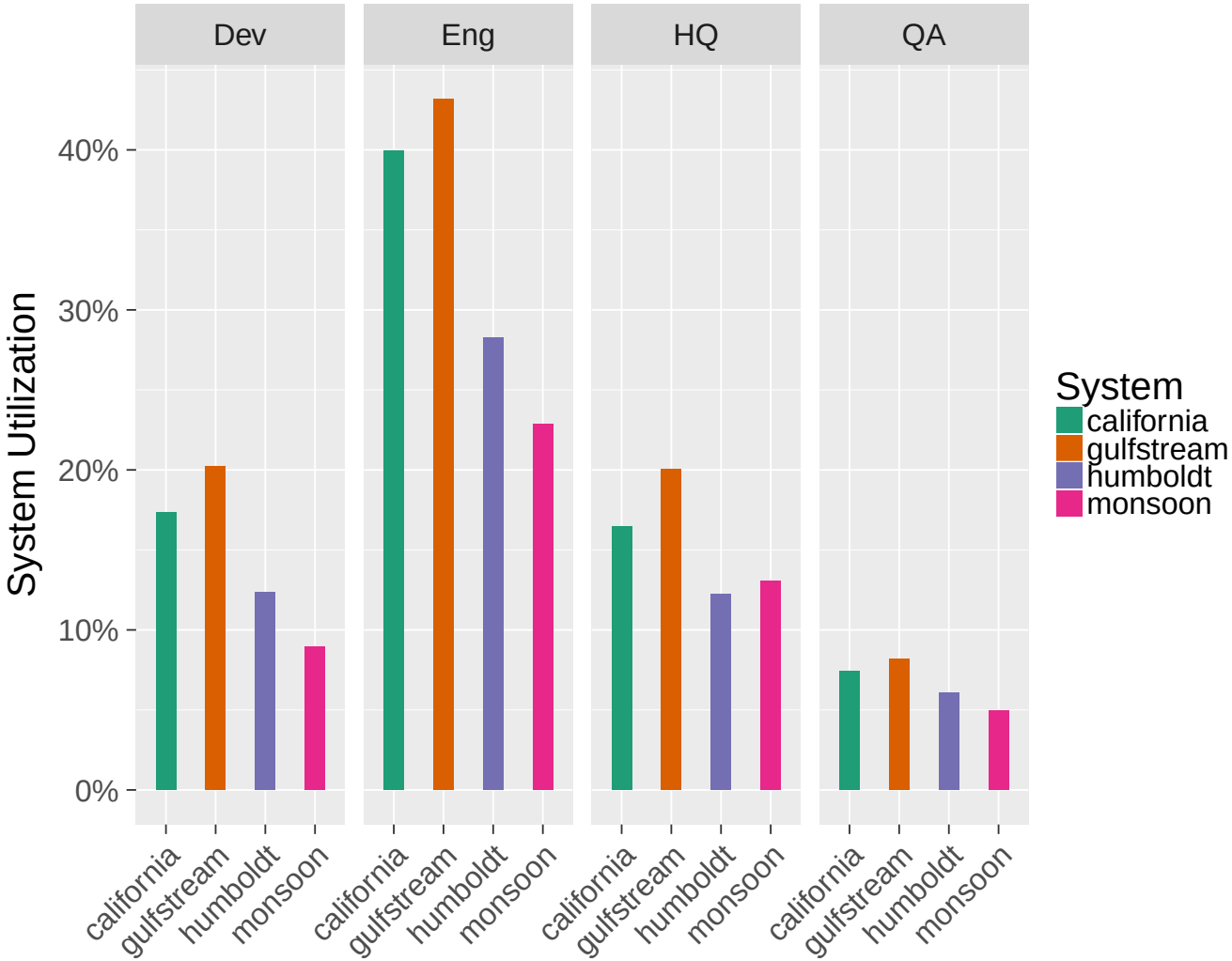


Using facets to organize data

```
> report.data %>%                                                    1
+     rollup.system.usage(start.date, end.date) %>%                  2
+         system.barchart() +                                        3
+            facet_grid(. ~ org) +                                   4
+                theme(axis.text.x = element_text(angle = 45, hjust = 1))   5
```

Note how one can adapt a plot before it is rendered

Result


Dev
Eng
HQ
QA
System Utilization
40%
30%
20%
10%
0%
california
gulfstream
humboldt
monsoon
System
california
gulfstream
humboldt
monsoon

Users: Ordering by usage

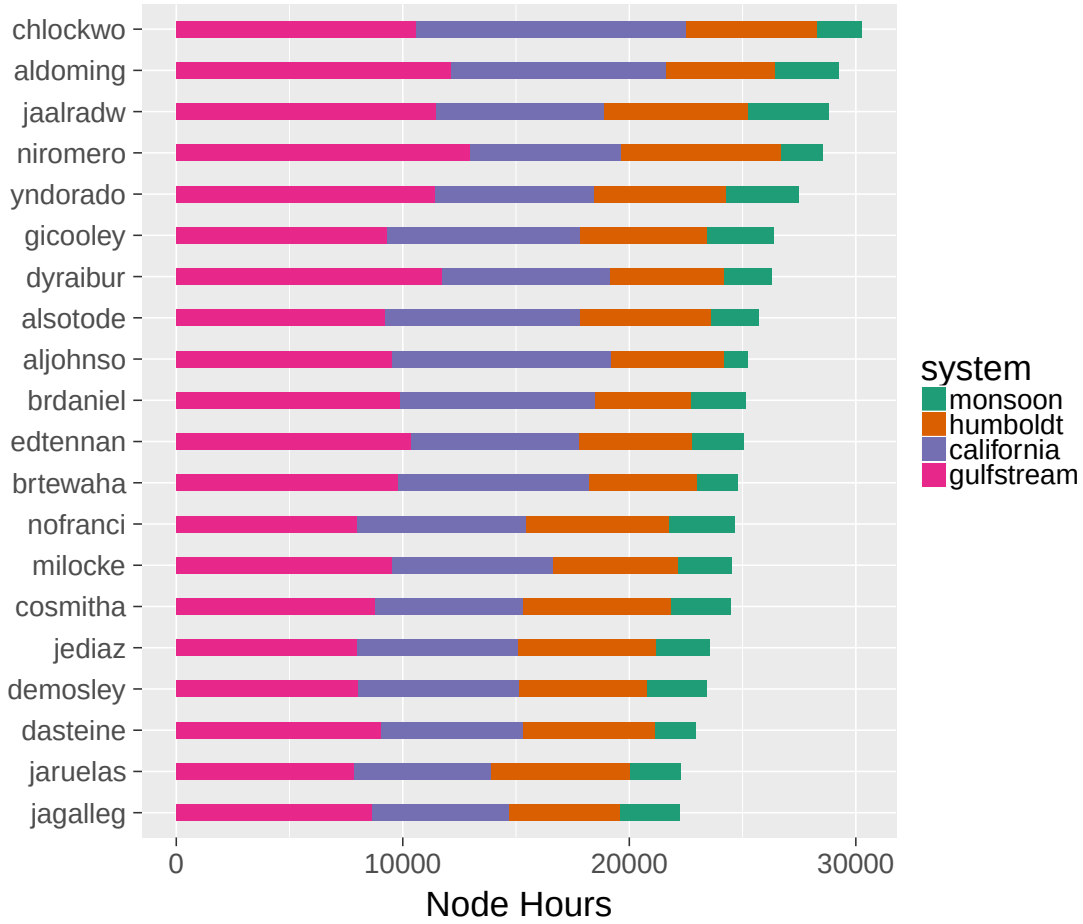
This is harder; ggplot sometimes has to be convince to change its default orders. Here we change the order of the levels in a factor

R code

```
user.barchart <- function(d) {
  # Order the user names by usage
  temp.df <- d %>% group_by(login) %>% summarize(usage=sum(usage))
  new.levels <- reorder(temp.df$login, temp.df$usage)
  d$login <- factor(d$login, levels=levels(new.levels), ordered=TRUE)

  ggplot(data=d) +
    geom_bar(aes(x=login, y=as.numeric(usage), fill=system),
             position="stack",
             width=.4,
             stat="identity") +
    scale_fill_brewer(palette="Dark2") +
    coord_flip() +
    xlab("") +
    ylab("Node Hours")
}
```

Result

```
> report.data %>%                                    1
+       rollup.usage() %>%                           2
+           user.barchart()                          3
```



Outline

Sweave, Brew, and xtable

`Sweave` transforms a file of R and TeX into a LaTeXdocument

`brew` is a templating package, like other templating packages.

`xtable` renders tabular data into HTML or LaTeX

- In the final few minutes, we're going to scrub up a mock report, like one that you might want to produce.
- We'll use a lot of placeholder text, and focus on the report.

Command Line Script!

R code

```
#!/usr/bin/env Rscript
library(lisa17ws)
library(brew)

# start.date should be a command-line argument...
report.start.date <- as.POSIXct("2017-12-01")
report.period = sprintf("%s, %d", month.name[month(report.start.date)],
                                  year(report.start.date))
report.orgs = getOption("lisa17.orgs")

# Create the primary document
brew(file="../templates/monthly.tmpl", output="report.Rnw")
Sweave("report.Rnw")

# Create sub-documents for each organization
for (report.org in sort(report.orgs)) {
    output.file <- sprintf("%s.Rnw", report.org)
    brew(file="../templates/org.tmpl", output=output.file)
    Sweave(output.file)
}

system("latexmk -pdfps report.tex")
```

Essential monthly template

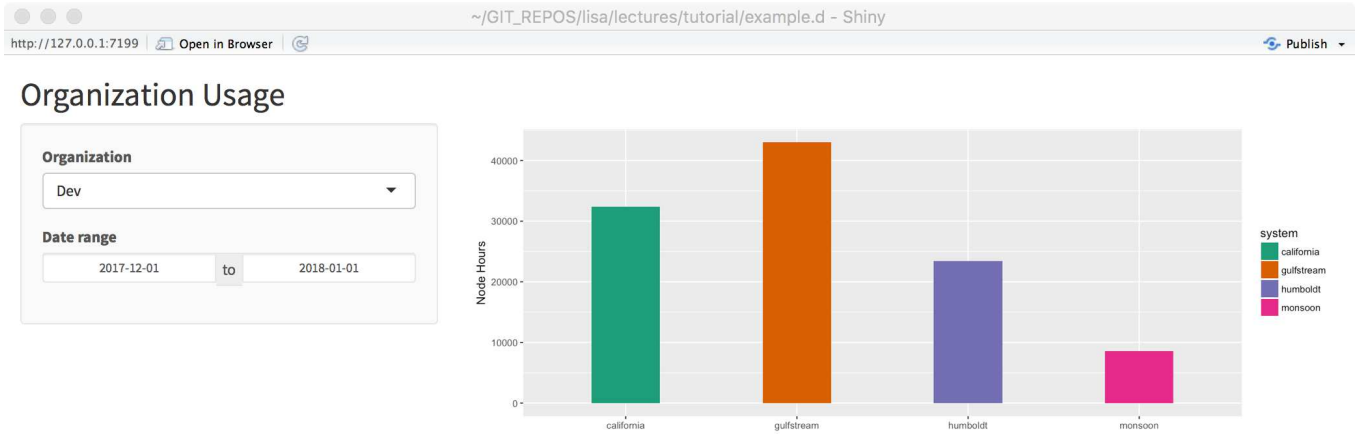
Brew Template

```
\begin{center}
<<overall,echo=FALSE,fig=TRUE,split=TRUE,eps=TRUE,pdf=FALSE>>=
rollup.usage(report.data) %>%
      org.barchart() + facet_grid( . ~ system) +
           theme(axis.text.x = element_text(angle = 45, hjust = 1))
@
\end{center}

\lipsum[2]

\begin{center}
<<tab, echo=FALSE, results=tex>>=
df <- rollup.usage(report.data) %>% group_by(login) %>%
                                        summarize(usage=sum(usage))
df$usage = formatC(df$usage, digits=2, format="f", big.mark=",")
print(xtable(df))
@
\end{center}
```

R Shiny

- Web Server integrated with R-Studio
- Can run as standalone
- Components are UI and Server
- Calls back into R, just like Sweave
- Following example uses the same data analysis and display code as always, just wrapped into a different script.

Shiny Screenshot



Questions?

Thanks!

Remember to fill out your evaluation!