DESIGNSAFE-CI

A NATURAL HAZARDS ENGINEERING COMMUNITY

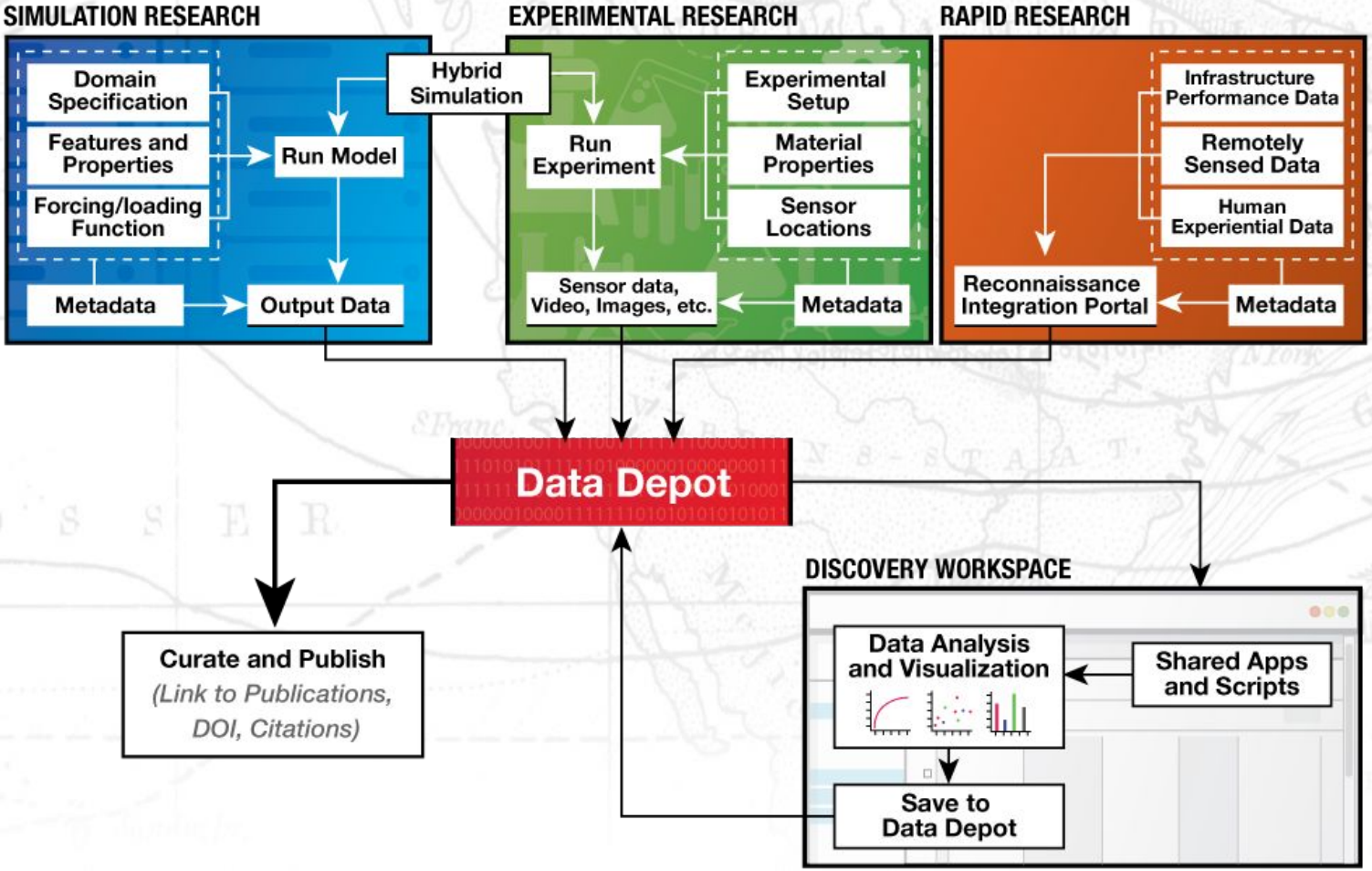
A New Cyberinfrastructure for the Natural Hazards Community

DESIGNSAFE-CI A NATURAL HAZARDS ENGINEERING COMMUNITY | TEXAS The University of Texas at Austin | TACC | RICE | Florida Tech

DesignSafe-ci Vision

- A CI that is an integral and dynamic part of research discovery
- Cloud-based tools that support the analysis, visualization, and integration of diverse data types
  - Key to unlocking the power of “big data”
- Support end-to-end research workflows and the full research lifecycle
- Enhance, amplify, and link the capabilities of the other NHERI components

DesignSafe: Enabling Research



Unleashing DesignSafe with Jupyter and R

Charlie Dey
charlie@tacc.utexas.edu

DESIGNSAFE-CI
A NATURAL HAZARDS ENGINEERING COMMUNITY

TEXAS
The University of Texas at Austin

TACC RICE Florida Tech

Agenda

- Introduction to Slack
- Introduction to the Jupyter Notebook
- Basics of R
- Playing with Dataframes
- Basic Plotting
- Data Analysis through ggplot2

What's Slack?

In a nutshell, Slack is a communication tool.

The Lingo:

- Teams
  - Slack is divided into teams, it's designated in the url you connect to.
    - https://designsafe-ci.slack.com/
- Channels
  - These are "Topics of Discussion"
  - Anyone can create a channel
  - Private conversations
    - even w/ yourself
- Threads:
  - You can group communications together and create subtopics

What's Slack?

But, it's a bit more than just a "Communication Tool"

- use slack to set up a reminders
  `/remind me in <time> to <message>`
- make a to do list
  /todo <task> at <due date or time>
- star messages for follow up
- team wide searches
- and animated GIFS!
  - /giphy

*after the presentation is over, we'll continue the conversation on slack, https://designsafe-ci.slack.com/

What are Jupyter Notebooks?

A web-based, interactive computing tool for capturing the whole computation process: developing, documenting, and executing code, as well as communicating the results.

How do Jupyter Notebooks Work?

An open notebook has exactly one interactive session connected to a kernel which will execute code sent by the user and communicate back results. This kernel remains active if the web browser window is closed, and reopening the same notebook from the dashboard will reconnect the web application to the same kernel.

What's this mean?

Notebooks are an interface to kernel, the kernel executes your code and outputs back to you through the notebook. The kernel is essentially our programming language we wish to interface with.

Jupyter Notebooks, Structure

- Code Cells
  - Code cells allow you to enter and run code
    Run a code cell using Shift-Enter

- Markdown Cells
  - Text can be added to Jupyter Notebooks using Markdown cells. Markdown is a popular markup language that is a superset of HTML.

Jupyter Notebooks, Structure

- Markdown Cells
  - You can add headings:
    - # Heading 1
Heading 2
Heading 2.1
Heading 2.2
  - You can add lists
    - 1. First ordered list item
2. Another item
· · * Unordered sub-list.
1. Actual numbers don't matter, just that it's a number
· · 1. Ordered sub-list
4. And another item.

Jupyter Notebooks, Structure

- Markdown Cells
  - pure HTML
    - <dl>
      <dt>Definition list</dt>
      <dd>Is something people use sometimes.</dd>

      <dt>Markdown in HTML</dt>
      <dd>Does *not* work **very** well. Use HTML tags.</dd>
      </dl>
  - And even, Latex!
    - $e^{i\pi} + 1 = 0$

Jupyter Notebooks, Workflow

Typically, you will work on a computational problem in pieces, organizing related ideas into cells and moving forward once previous parts work correctly. This is much more convenient for interactive exploration than breaking up a computation into scripts that must be executed together, as was previously necessary, especially if parts of them take a long time to run.

Jupyter Notebooks, Workflow

- Let a traditional paper lab notebook be your guide:
  - Each notebook keeps a historical (and dated) record of the analysis as it's being explored.
  - The notebook is not meant to be anything other than a place for experimentation and development.
  - Notebooks can be split when they get too long.
  - Notebooks can be split by topic, if it makes sense.

Jupyter Notebooks, Shortcuts

- **`Shift-Enter`: run cell**
  - Execute the current cell, show output (if any), and jump to the next cell below. If `Shift-Enter`is invoked on the last cell, a new code cell will also be created. Note that in the notebook, typing `Enter` on its own *never* forces execution, but rather just inserts a new line in the current cell. `Shift-Enter` is equivalent to clicking the `Cell | Run` menu item.

Jupyter Notebooks, Shortcuts

- `Ctrl-Enter`: **run cell in-place**
  - Execute the current cell as if it were in “terminal mode”, where any output is shown, but the cursor *remains* in the current cell. The cell’s entire contents are selected after execution, so you can just start typing and only the new input will be in the cell. This is convenient for doing quick experiments in place, or for querying things like filesystem content, without needing to create additional cells that you may not want to be saved in the notebook.

Jupyter Notebooks, Shortcuts

- **`Alt-Enter`: run cell, insert below**
  - Executes the current cell, shows the output, and inserts a *new* cell between the current cell and the cell below (if one exists). (shortcut for the sequence `Shift-Enter`,`Ctrl-m a`. (`Ctrl-m a` adds a new cell above the current one.))
- **`Esc` and `Enter`: Command mode and edit mode**
  - In command mode, you can easily navigate around the notebook using keyboard shortcuts. In edit mode, you can edit text in cells.

What is R

- R is a language and environment for statistical computing and graphics.
- Began as an experiment in using Lisp implementors to build a testbed statistical environment.

What is R

- an effective data handling and storage facility
- a suite of operators for calculations on arrays, in particular matrices
- integrated collection of data analysis tools
- graphical engine for plotting
- a simple programming language which includes conditionals, loops, user-defined recursive functions and input and output facilities.
- C, C++ and Fortran code can be linked and called at run time. (Advanced users can write C code to manipulate R objects directly.)

What is R

Three basic classes of data objects or *modes*:

- numeric
- character
- logical.

Numerics

```
> n <- 48
> n
[1] 48
```

Specials:

- Inf is a special number which denotes infinity (1 / 0 = Inf).
- NaN indicates an undefined value, Not a Number, due to a disallowed mathematical operation, i.e. taking the square root of a negative number would result in a NaN.

```
> h = sqrt(-2)
Warning message:
In sqrt(-2) : NaNs produced
> h
[1] NaN
```

Characters

```
> c = "Hello World!"
> c
[1] "Hello World!"
```

Logicals

```
> y = 7 * 6
> x = 42
> y == x
[1] TRUE
```

Data Objects Extended

R has a wide variety of data types including:

- scalars
- vectors (numerical, character, logical)
- matrices
- data frames
- lists

Vectors

```
a <- c(1,2,5.3,6,-2,4) # numeric vector
b <- c("one","two","three") # character vector
c <- c(TRUE,TRUE,TRUE,FALSE,TRUE,FALSE) #logical vector
```

To make a reference:

```
> a[6]
[1] 4
> a[4]
[1] 6
> a[2]
[1] 2
> a[c(6, 4, 2)]
[1] 4 6 2
>
```

Matrices

```
# generates 5 x 4 numeric matrix
y<-matrix(1:20, nrow=5,ncol=4)

# another example
cells <- c(1,26,24,68)
rnames <- c("R1", "R2")
cnames <- c("C1", "C2")
mymatrix <- matrix(cells, nrow=2, ncol=2, byrow=TRUE,
  dimnames=list(rnames, cnames))
```

All columns in a matrix must have the same mode(numeric, character, etc.) and the same length. The general format is

```
mymatrix <- matrix(vector, nrow=r, ncol=c, byrow=FALSE,
     dimnames=list(char_vector_rownames, char_vector_colnames))
```

Matrices

Referring to the elements:

```
x[1,1] # element at 1,1
x[,4] # 4th column of matrix
x[3,] # 3rd row of matrix
x[2:4,1:3] # rows 2,3,4 of columns 1,2,3
```

n-Dimensional Arrays

An n-Dimensional Array is basically a set of stacked matrices.

```
> a <- matrix(8, 2, 3)          # Creates a 2 x 3 matrix populated with 8's.
> b <- matrix(9, 2, 3)          # Creates a 2 x 3 matrix populated with 9's.
> array(c(a, b), c(2, 3, 2)) # Creates a 2 x 3 x 2 array with the first
   , , 1                        #  level [,,1] populated with matrix a (8's),
                                #  and the second level [,,2] populated
        [,1] [,2] [,3]          #  with matrix b (9's).
   [1,]    8    8    8
   [2,]    8    8    8         # Use square brackets to extract values.  For
                                #  example, [1, 2, 2] extracts the second
   , , 2                        #  value in the first row of the second level.
                                # You may also use the <- operator to
        [,1] [,2] [,3]          #  replace values.
   [1,]    9    9    9
   [2,]    9    9    9
```

Lists

Lists are vectors that allow mixtures of different classes. They are most useful if created with name-value pairs so that they can be accessible by name.

```
person = list(firstName="Sue", lastName="Smith", age=20, married="F")
> person
$firstName
[1] "Sue"

$lastName
[1] "Smith"

$age
[1] 20

$married
[1] "F"

> person$lastName
[1] "Smith"
```

Data Frames

Dataframes are to matrices what lists are to vectors.
They are two dimensional tables where the values within a column need to be the same type, but values between columns do not. *Data read from an external file will be put into a dataframe.

```
 firstName. lastName. age
1     Homer,  Simpson,  36
2     Marge,  Simpson,  34
3      Bart,  Simpson,  20
4      Lisa,  Simpson,   8
5    Maggie,  Simpson,   1
```

Data Frames

A data frame is more general than a matrix, in that different columns can have different modes (numeric, character, factor, etc.).

```
>d <- c(1,2,3,4)
>e <- c("red", "white", "red", NA)
>f <- c(TRUE,TRUE,TRUE,FALSE)
>mydata <- data.frame(d,e,f)
>names(mydata) <- c("ID","Color","Passed") # variable names
```

Referring to the elements.

```
>myframe[3:5] # columns 3,4,5 of data frame
>myframe[c("ID","Age")] # columns ID and Age from data frame
>myframe$X1 # variable x1 in the data frame
```

Factors

Tell R that a variable is nominal by making it a factor. The factor stores the nominal values as a vector of integers in the range [1... k] (where k is the number of unique values in the nominal variable), and an internal vector of character strings (the original values) mapped to these integers.

```
# variable gender with 20 "male" entries and
# 30 "female" entries
gender <- c(rep("male",20), rep("female", 30))
gender <- factor(gender)
# stores gender as 20 1s and 30 2s and associates
# 1=female, 2=male internally (alphabetically)
# R now treats gender as a nominal variable
summary(gender)
```

Ordered Factors

An ordered factor is used to represent an **ordinal variable**.

```
# variable rating coded as "large", "medium", "small'
rating <- ordered(rating)
# recodes rating to 1,2,3 and associates
# 1=large, 2=medium, 3=small internally
# R now treats rating as ordinal
```

Useful Functions for Data Types

```
length(object) # number of elements or components
str(object)    # structure of an object
class(object)  # class or type of an object
names(object)  # names

c(object,object,...)       # combine objects into a vector
cbind(object, object, ...) # combine objects as columns
rbind(object, object, ...) # combine objects as rows

object     # prints the object

ls()       # list current objects
rm(object) # delete an object

newobject <- edit(object) # edit copy and save as newobject
fix(object)               # edit in place
```

Control Structures

if...else

```
> i <- 4
> i
[1] 4
> if (i < 2)
+ {
+ print (i)
+ }else {
+ print (i * i)
+ }
[1] 16
```

Control Structures

for loop

```
> for (a in 1:3) {
+ print (a*a)
+ }
[1] 1
[1] 4
[1] 9
```

Control Structures

while loop

```
> i=1
> while (i<4) {
+ print(i*2)
+ i = i + 1
+ }
[1] 2
[1] 4
[1] 6
```

Control Structures

Creating a vector:

```
x <- array()                # Initializes an empty data structure.
for (i in 1:10) {           # Loops through every value from 1 to 10,
replacing
  if (is.integer(i/2)) { #  the even values in `x' with i+5.
    x[i] <- i + 5
  }
}                           # Enclose multiple commands in {}.
```

Fibonacci Sequence

```
len <- 10
fibvals <- numeric(len)
fibvals[1] <- 1
fibvals[2] <- 1
for (i in 3:len) {
   fibvals[i] <- fibvals[i-1]+fibvals[i-2]
}
```

Pascal's Triangle

```
pascalTriangle <- function(h) {
  for(i in 0:(h-1)) {
    s <- ""
    for(k in 0:(h-i)) s <- paste(s, "  ", sep="")
    for(j in 0:i) {
      s <- paste(s, sprintf("%3d ", choose(i, j)), sep="")
    }
    print(s)
  }
}
```

Importing Data

```
# first row contains variable names, comma is separator
# assign the variable id to row names


mydata <- read.table("/home/jupyter/mydata/<name of file>.csv",
header=TRUE, sep=",", row.names="id")
```

Querying Data

```
# RODBC Example
# import 2 tables (Crime and Punishment) from a DBMS
# into R data frames (and call them crimedat and pundat)

library(RODBC)
myconn <-odbcConnect("mydsn", uid="Rob", pwd="aardvark")
crimedat <- sqlFetch(myconn, "Crime")
pundat <- sqlQuery(myconn, "select * from Punishment")
close(myconn)
```

http://www.mayin.org/ajayshah/KB/R/

Example Code

```
# Goal: To read in a simple data file, and look around it's contents.

# Suppose you have a file "x.data" which looks like this:
#        1997,3.1,4
#        1998,7.2,19
#        1999,1.7,2
#        2000,1.1,13
# To read it in --

A <- read.table("x.data", sep=",",
                col.names=c("year", "my1", "my2"))
nrow(A)                                 # Count the rows in A

summary(A$year)                         # The column "year" in data frame A
                                        # is accessed as A$year


A$newcol <- A$my1 + A$my2               # Makes a new column in A
newvar <- A$my1 - A$my2                 # Makes a new R object "newvar"
A$my1 <- NULL                           # Removes the column "my1"

# You might find these useful, to "look around" a dataset --
str(A)
summary(A)
library(Hmisc)          # This requires that you've installed the Hmisc package
contents(A)
describe(A)
```

http://www.mayin.org/ajayshah/KB/R/

Example Code

```
# Goal: Joint distributions, marginal distributions, useful
tables.

# First let me invent some fake data
set.seed(102)  # This yields a good illustration.
x <- sample(1:3, 15, replace=TRUE)
education <- factor(x, labels=c("None", "School", "College"))
x <- sample(1:2, 15, replace=TRUE)
gender <- factor(x, labels=c("Male", "Female"))
age <- runif(15, min=20,max=60)

D <- data.frame(age, gender, education)
rm(x,age,gender,education)
print(D)

# Table about education
table(D$education)

# Table about education and gender --
table(D$gender, D$education)
# Joint distribution of education and gender --
table(D$gender, D$education)/nrow(D)

# Add in the marginal distributions also
addmargins(table(D$gender, D$education))
addmargins(table(D$gender, D$education))/nrow(D)
```

```
# Generate a good LaTeX table out of it --
library(xtable)
xtable(addmargins(table(D$gender, D$education))/nrow(D),
       digits=c(0,2,2,2,2))            # You have to do | and
\hline manually.

# Study age by education category
by(D$age, D$gender, mean)
by(D$age, D$gender, sd)
by(D$age, D$gender, summary)

# Two-way table showing average age depending on education &
gender
a <- matrix(by(D$age, list(D$gender, D$education), mean), nrow=2)
rownames(a) <- levels(D$gender)
colnames(a) <- levels(D$education)
print(a)
# or, of course,
print(xtable(a))
```

http://www.mayin.org/ajayshah/KB/R/

Example Code

```
# Goal: Show the efficiency of the mean when compared with the median
#       using a large simulation where both estimators are applied on
#       a sample of U(0,1) uniformly distributed random numbers.

one.simulation <- function(N=100) {     # N defaults to 100 if not supplied
  x <- runif(N)
  return(c(mean(x), median(x)))
}

# Simulation --
results <- replicate(100000, one.simulation(20)) # Gives back a 2x100000 matrix

# Two kernel densities --
k1 <- density(results[1,])              # results[1,] is the 1st row
k2 <- density(results[2,])

# A pretty picture --
xrange <- range(k1$x, k2$x)
plot(k1$x, k1$y, xlim=xrange, type="l", xlab="Estimated value", ylab="")
grid()
lines(k2$x, k2$y, col="red")
abline(v=.5)
legend(x="topleft", bty="n",
       lty=c(1,1),
       col=c("black", "red"),
       legend=c("Mean", "Median"))
```

And... Plotting Graphs!

Let's plot a simple sin curve from 0 to 2*pi

We'll need a sequence of numbers for x

and we'll need to calculate sin(x)

```
x=seq(0,2*pi,0.01)
y=sin(x)
```

Plotting

And then… plot

```
plot(x,y,type='l', main="Sin Curve")
```

Basic Data Analysis Plotting

Let's look at some seismic data.

```
w1 <- read.csv(file="/home/jupyter/mydata/training_data.csv",sep=",",head=TRUE)
```

Basic Data Analysis Plotting

Column 1: PGA (g) of ground motion
Column 2: Column curvature ductility (dimensionless)
Column 3: Expansion bearing longitudinal displacement
Column 4: Expansion bearing transverse displacement

These are results from the simulations by Bryant G. Nielson (Georgia Tech, 2005). These are results of nonlinear finite element analysis of 96 randomly generated bridge samples hit with 96 earthquake ground motions. The peak ground acceleration along with the peak of three of the component responses have been reported here.

```
w1
names(w1)
str(w1)
summary(w1)
```

Basic Data Analysis Plotting

Column 1: PGA (g) of ground motion
Column 2: Column curvature ductility (dimensionless)
Column 3: Expansion bearing longitudinal displacement
Column 4: Expansion bearing transverse displacement

Let's add 4 more columns:
log(PGA)
log(Col.res)
log(exp_brg_long)
log(fxd_brg_long)

```
w1$ln1 = log(w1$PGA)
w1$lnCR = log(w1$Col.res)
w1$lnExp = log(w1$exp_brg_long)
w1$lnFxd = log(w1$fxd_brg_long)
```

Basic Data Analysis Plotting

A **strip chart** is the most basic type of plot available. It plots the data in order along a line with each data point represented as a box. Here we provide examples using the w1 data frame mentioned at the top of this page, and the one column of the data is w1$vals.

To create a strip chart of this data use the stripchart command:

```
stripchart(w1$exp_brg_long))
```

Basic Data Analysis Plotting

If you would prefer to see which points are repeated you can specify that repeated points be stacked:

```
stripchart(w1$exp_brg_long), method="stack")
```

Basic Data Analysis Plotting

A variation on this is to have the boxes moved up and down so that there is more separation between them:

```
stripchart(w1$exp_brg_long), method="jitter")
```

Basic Data Analysis Plotting

If you do not want the boxes plotting in the horizontal direction you can plot them in the vertical direction:

```
stripchart(w1$exp_brg_long),vertical=TRUE)
stripchart(w1$exp_brg_long,vertical=TRUE,method="jitter")
```

Basic Data Analysis Plotting

If you wish to label your plots, we can do that directly through the stripchart command

```
stripchart(w1$exp_brg_long,method="stack",
           main='Expansion bearing longitudinal displacement',
           xlab='Expansion bearing longitudinal displacement')
```

Basic Data Analysis Plotting

Histogram is very common plot. It plots the frequencies that data appears within certain ranges. Let's check out another column of data, w1$PGA (g) of ground motion.

To plot a histogram of the data use the “hist” command:

```
hist(w1$PGA)
hist(w1$PGA,main="PGA (g) of ground motion",xlab="PGA")
```

Basic Data Analysis Plotting

You can specify the number of breaks to use using the breaks option.

```
hist(w1$PGA,main="PGA (g) of ground motion",xlab="PGA", breaks = 8)
```

Basic Data Analysis Plotting

A **scatter plot** provides a graphical view of the relationship between two sets of numbers. Let's look at the relationship between two columns of our seismic data we looked at previously

```
plot(w1$PGA,w1$fxd_brg_long)
```

Basic Data Analysis Plotting

Let's label our final plot...

```
plot(tree$STBM,tree$LFBM,
       main="Seismic Activity of 96 Randomly Generated Bridge Samples from OpenSees",
       xlab="PGA (g) of ground motion",
       ylab="Expansion bearing longitudinal displacement")
plot(w1$PGA,w1$fxd_brg_long)
```

Plotting with ggplot2

Let's look at what we can do with ggplot2.

- ggplot2 is an implementation of “grammar of graphics.”
- basic components of a graphic are broken into distinct parts
- these parts get added together and get our plot.

```
library(ggplot2)
```

Plotting with ggplot2

We'll also need dplyr which is the next iteration of plyr, tools focused on working with data frames (hence the d in the name).

- Identify the most important data manipulation tools needed for data analysis and make them easy to use from R.
- Provide fast performance for in-memory data by writing key pieces in C++.
- Use the same interface to work with data no matter where it's stored, whether in a data frame, a data table or database.

```
library(dplyr)
```

Plotting with ggplot2

First thing we need to to create our ggplot object.

Everything we do will build off of this object. The bare minimum for this is the data and aes(), or the aesthetics layers - where we define our x and y axis.

This creates our blank slate that contains our data and defines what we want on the x and y axis.

```
myplot <- ggplot(w1, aes(x=lnRPG, y=lnExp))
myplot
```

Plotting with ggplot2

Let's plot points; we can add "geometries"to the ggplot object.

Notice that we add new “things” to a ggplot object by adding new functions. Basically it takes the output from the first function as the input to the second.

To add points and create the plot, we need to add that to our object.

```
myplot + geom_point()
```

Plotting with ggplot2

We can also use that syntax to create a new object

```
myscatter <- myplot + geom_point()
myscatter
```

Plotting with ggplot2

and "add" some labels and a title

```
myascatter <- myscatter +
  ggtitle("PGA (g) of ground motion vs Expansion bearing transverse displacement") +
  xlab("PGA(g) of ground motion") + ylab("Expansion bearing transverse displacement")
myscatter
```

Plotting with ggplot2

add a bit of interest...

```
myscatter <- myscatter + geom_point(aes(color=(Col.res > median(w1$Col.res)),
shape=(Col.res > median(w1$Col.res)), size=5))
```

Plotting with ggplot2

add some more interest

Lets add a loess line with 95% confidence intervals

```
myscatter_loess <- myscatter + geom_smooth()
myscatter_loess
```

Plotting with ggplot2

add some more interest

add a simple regression line...

```
myscatter_lm <- myscatter + geom_smooth(method="lm")
myscatter_lm
```

Plotting with ggplot2

add some more interest

let's group our data...

```
myscatter_lm_group <- myscatter +
  geom_smooth(method="lm", aes(group=(Col.res > median(w1$Col.res))))
myscatter_lm_group
```

Plotting with ggplot2

add some more interest

finally, add some color to our regression lines

```
myscatter_lm_color <- myscatter +
  geom_smooth(method="lm", aes(color=(Col.res > median(w1$Col.res)), fill=(Col.res >
median(w1$Col.res))))
myscatter_lm_color
```

Summary

R can be very useful in understanding data.

There is a story behind every dataset, our goal is to dig through that data to find and highlight the stories that may not be seen at first glance.

- Basic R
- Using Dataframes
- Plotting our data
- Finding the story

And we're just grazing the surface.

DesignSafe: Thanks

Ellen Rathje, Tim Cockerill, Jamie Padgett, Dan Stanzione, Steve Mock, Josue Coronel, Joe Stubbs, Craig Jansen, Matt Stelmaszek, Hedda Prochaska, Joonyee Chuah, Sushreyo Misra

Questions?

For additional questions, feel free to contact us

- Slack:
  https://designsafe-ci.slack.com/
- Email:
  training@designsafe-ci.org
- or fill out a ticket:
  https://www.designsafe-ci.org/help/tickets

DesignSafe: References

- http://www.r-tutor.com/r-introduction
- http://www.cyclismo.org/tutorial/R/plotting.html
- http://usgs-r.github.io/gsIntroR/H_Visualize.html#customizing-plots-with-themes