STAT 408: Week 2

Reproducible workflows: R Markdown, Git, GitHub

1/25/2022

Reproducibility

Reproducibility

*Research is **reproducible** when the exact results can be reproduced if given access to the original data, software, and code.*

Reproducibility checklist

Goal: Train new analysts whose only workflow is a reproducible one.

1. Are the tables and figures reproducible from the code and data?
2. Does the code actually do what you think it does?
3. In addition to what was done, is it clear *why* it was done?
4. Can the code by used for other data?
5. Can you extend the code to do other things?

How to produce reproducible research?

We need an environment where

- data, analysis, and reporting results are tightly connected, or better yet, inseparable
- the original data remain untouched
- all data manipulations and analyses are inherently documented
- documentation is human readable and syntax is minimal

Roadmap

1. Scriptability $\rightarrow$ R
2. Literate programming $\rightarrow$ R Markdown
3. Version control $\rightarrow$ Git + GitHub
   - Lots of mistakes along the way, need ability to revert!
   - Removes barriers to well-documented collaboration
   - Transparent commit history = accountability
   - Mastery takes time, earlier start the better = marketability

.R and .Rmd

R script files: .R

R demo - create your first R script file!

- make an object assignment (<-)
- then inspect it
- include a comment (#)
- do some basic arithmetic
- call an R function:

```
functionName(arg1 = val1, arg2 = val2, and so on)
```

R Markdown setup

Do this once:

```
install.packages("rmarkdown")
```

(or install from Packages tab)

R Markdown files: .Rmd

R Markdown overview

R Markdown demo - create your first R Markdown file!

- headers
- italics and bold
- R chunks

Git and GitHub

Basic Git workflow

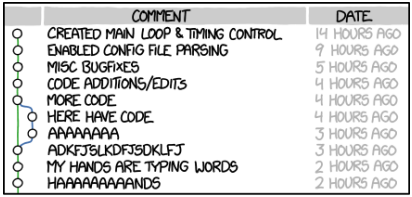
Setup

1. Create new repository ('repo') in Github
2. `clone` to a local directory

Useage

1. `pull` - **Always pull first!** The importance of this first step will become apparent when we start collaborating with others in the same repo.
2. Make your local changes.
3. `commit` - Include a short message to remind you what changes you just made. Advice: *Commit early and often!*
4. `push` - "Pushes" your changes to the central repo

More on commit message subject lines



- Use the imperative mood: complete the sentence, "If applied, this commit will. . . "
- Limit 50 characters
- Capitalize
- Do not end with a period

Examples: "Add link to textbook", "Update calendar for week 7"

Basic Git workflow with RStudio

1. Create new repository ('repo') in Github
2. In RStudio, File -> New project... Version control -> Git ->
   - Repository URL: [enter https link to repo]
   - Project directory name: [typical to use repo name here]
   - Create project as subdirectory of: [local path for repo directory]
3. "Git" tab buttons: Diff, Commit, Pull, Push

Your turn!

- If you don't have a GitHub account, create one now at **github.com**
- If you do, confirm you know your username and password by logging in at **github.com**
- Then, enter your name and GitHub username in D2L -> Course Resources -> Survey -> GitHub User Information

Prior to Thursday lab...

Read Happy Git with R chapters listed on course calendar and use them to...

1. create a GitHub account and enter your name and GitHub username in D2L -> Course Resources -> Survey -> GitHub User Information (*should have done today!*)
2. install Git
3. configure Git, GitHub, and RStudio

Data Structures in R

Reading data files

The ability to read datasets into R is an essential skill. For this class, most of the files will be on a webpage and can be directly downloaded using `read.csv` (or `read_csv` in the `tidyverse`).

Consider a dataset available at:
http://math.montana.edu/ahoegh/teaching/stat408/datasets/SeattleHousing.csv

```
Seattle <- read.csv(
  'http://math.montana.edu/ahoegh/teaching/stat408/datasets/SeattleHousing.csv',
  stringsAsFactors = F)
```

Viewing data files

A common function that we will use is `head()`, which shows the first few rows of a data frame.

```
head(Seattle, 3)
```

```
##     price bedrooms bathrooms sqft_living sqft_lot floors waterfront sqft_above
## 1 1350000        3      2.50        2753    65005      1          1       2165
## 2  228000        3      1.00        1190     9199      1          0       1190
## 3  289000        3      1.75        1260     8400      1          0       1260
##   sqft_basement zipcode     lat     long yr_sold mn_sold
## 1           588   98070 47.4041 -122.451    2015       3
## 2             0   98148 47.4258 -122.322    2014       9
## 3             0   98148 47.4366 -122.335    2014       8
```

Viewing data files

Other useful functions to examine a data file: `tail()`, `names()`, `dim()`

```
tail(Seattle, 2)
```

```
##      price bedrooms bathrooms sqft_living sqft_lot floors waterfront sqft_above
## 868 399950        2      1.00         710     1157      2          0        710
## 869 224000        3      1.75        1500    11968      1          0       1500
##     sqft_basement zipcode     lat     long yr_sold mn_sold
## 868             0   98102 47.6413 -122.329    2014       6
## 869             0   98010 47.3095 -122.002    2014       6
```

```
names(Seattle)
```

```
##  [1] "price"         "bedrooms"      "bathrooms"     "sqft_living"
##  [5] "sqft_lot"      "floors"        "waterfront"    "sqft_above"
##  [9] "sqft_basement" "zipcode"       "lat"           "long"
## [13] "yr_sold"       "mn_sold"
```

```
dim(Seattle)
```

```
## [1] 869  14
```

Data structure overview

R has four common types of data structures:

- Vectors
- Matrices (and Arrays)
- Lists
- Data Frames

Data structure overview

The base data structures in R can be organized by dimensionality and whether they are homogenous.

Dimension	Homogenous	Heterogenous
1d	Vector	List
2d	Matrix	Data Frame
no d	Array	

Vector types

There are four common types of vectors: logical, integer, double (or numeric), and character. The `c()` function is used for combining elements into a vector

```
dbl <- c(1,2.5,pi)
int <- c(1L,4L,10L)
log <- c(TRUE,FALSE,F,T)
char <- c('this is','a character string')
```

Vector types

The type of vector can be identified using the `typeof()` (or `class()`) function. Note that only a single data type is allowed.

```
  typeof(dbl)
```

```
## [1] "double"
```

```
  comb <- c(char,dbl)
  typeof(comb)
```

```
## [1] "character"
```

```
  comb
```

```
## [1] "this is"              "a character string" "1"
## [4] "2.5"                  "3.14159265358979"
```

Exercise: Vectors

Create a vector with your first, middle, and last names.

Solution: Vectors

Create a vector with your first, middle, and last names.

```
my.names <- c("Stacey","Allayne","Hancock")
my.names
```

```
## [1] "Stacey"  "Allayne" "Hancock"
```

Data frame overview

- the most common way of storing data in R
- like a matrix with rows-and-column structure; however, unlike a matrix each column may have a different type
- in a technical sense, a data frame is a list of equal-length vectors

```
df <- data.frame(x = 1:3, y = c('a','b','c'))
kable(df)
```

x	y
1	a
2	b
3	c

Subsetting

Vector subsetting: by indices

Subsetting allows you to extract certain elements from a data frame or vector (or matrix, array, list). We take subsets of vectors, matrices, and arrays by using square brackets of the appropriate dimension: `[ ]`, `[ , ]`, `[ , ,]`

```
num.vec <- seq(from = 1, to = 9, by = 1); num.vec
```

```
## [1] 1 2 3 4 5 6 7 8 9
```

```
num.vec[1:3]
```

```
## [1] 1 2 3
```

```
num.vec[c(1,5,8)]
```

```
## [1] 1 5 8
```

```
num.vec[-5]
```

```
## [1] 1 2 3 4 6 7 8 9
```

Vector subsetting: by logical

Subsetting also works with logical values or expressions.

```
num.vec[num.vec > 5]
```

```
## [1] 6 7 8 9
```

```
num.vec[num.vec != 6]
```

```
## [1] 1 2 3 4 5 7 8 9
```

```
num.vec[rep(c(TRUE,FALSE,TRUE),each=3)]
```

```
## [1] 1 2 3 7 8 9
```

Data Frame Subsetting: by indices

The same ideas apply to data frames, but the indices now constitute rows and columns of the data frame.

```
df <- data.frame(x=1:3, y=3:1, z=c('a','b','c'))
df[,1]
```

```
## [1] 1 2 3
```

```
df[-1,c(2:3)]
```

```
##   y z
## 2 2 b
## 3 1 c
```

Data Frame Subsetting: by $ or subset()

There are also a couple built in functions in R for subsetting data frames.

```
df$x
```

```
## [1] 1 2 3
```

```
new.df <- subset(df, x >1); new.df
```

```
##   x y z
## 2 2 2 b
## 3 3 1 c
```

Exercise: Subsetting

Adding to the .Rmd file you created earlier:

1. Read in the `Seattle` data set:

```
Seattle <- read.csv(
  'http://math.montana.edu/ahoegh/teaching/stat408/datasets/SeattleHo
  stringsAsFactors = F)
```

1. Create a new data frame that only includes houses worth more than $1,000,000.
2. (bonus) From this new data frame, what is the average living square footage of houses. Hint columns in a data.frame can be indexed by `Seattle$sqft_living`.

Exercise: Subsetting - Solutions

1. Create a new data frame that only includes houses worth more than $1,000,000.

```
expensive.houses <- subset(Seattle, price > 1000000) # or
expensive.houses <- Seattle[Seattle$price > 1000000,]
```

2. (bonus) From this new data frame what is the average living square footage of houses. Hint columns in a data.frame can be indexed by `Seattle$sqft_living`

```
mean(expensive.houses$sqft_living)
```

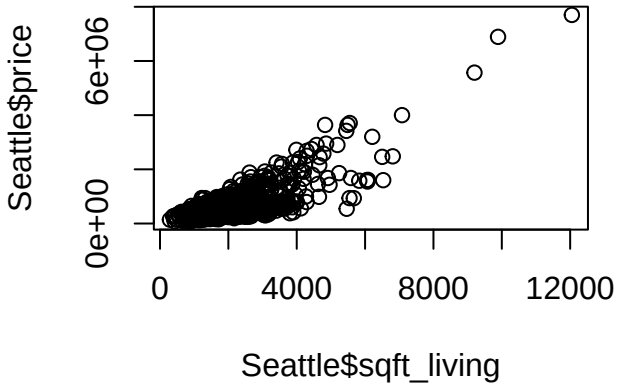
```
## [1] 3890.065
```

Base R Graphics

Scatterplot

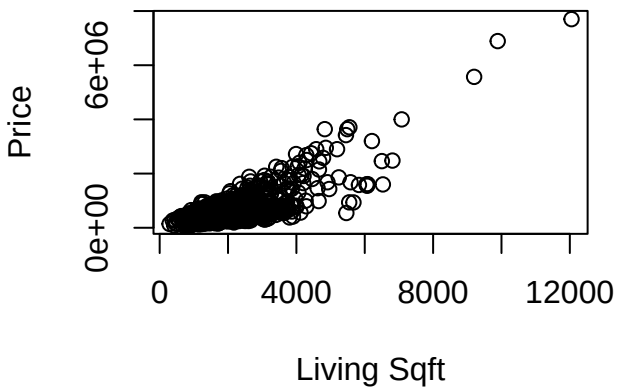
Later in the course, we will spend considerable time on graphics. For now, let's consider some of the basic functionality in R.

```
plot(Seattle$price ~ Seattle$sqft_living)
```



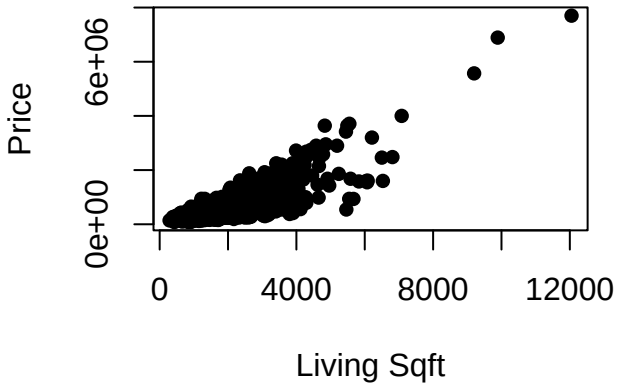
Add labels: xlab, ylab


```
plot(Seattle$price ~ Seattle$sqft_living,
     ylab='Price', xlab='Living Sqft')
```



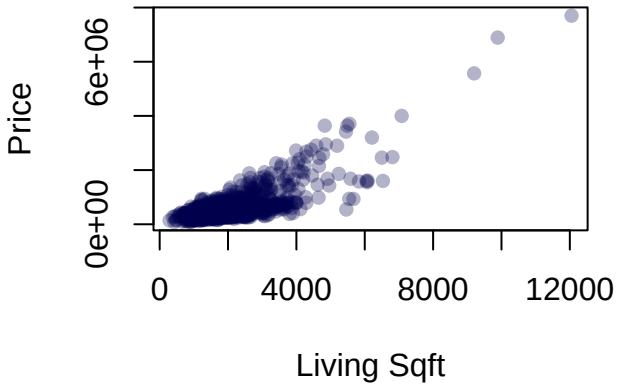
Specify point character: pch


```
plot(Seattle$price ~ Seattle$sqft_living,
     ylab='Price', xlab='Living Sqft', pch=16)
```



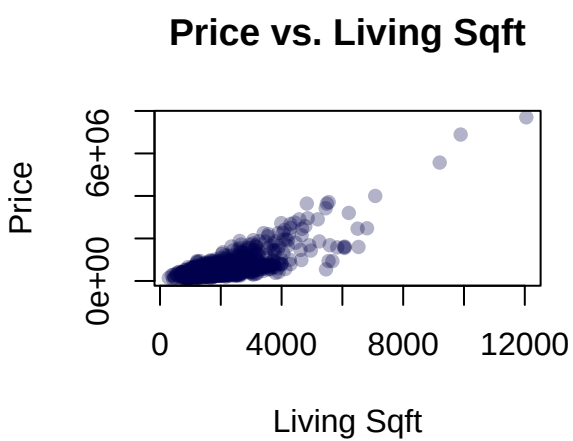
Specify color: `col`


```
plot(Seattle$price ~ Seattle$sqft_living,
     ylab='Price', xlab='Living Sqft', pch=16, col=rgb(0,0,.3,.3))
```



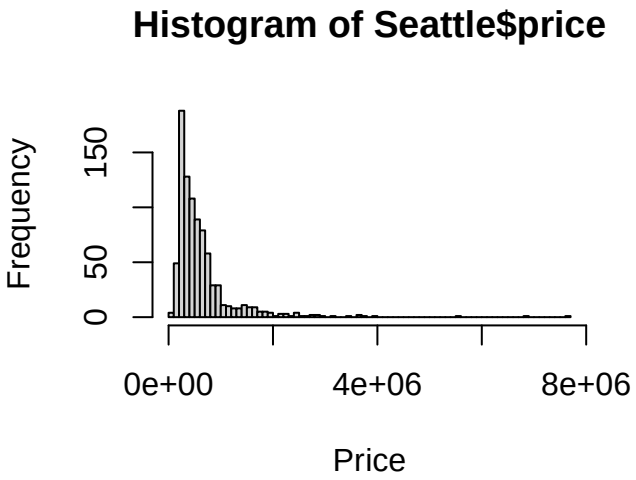
Add title: `main`


```
plot(Seattle$price~Seattle$sqft_living,
     ylab='Price', xlab='Living Sqft', pch=16, col=rgb(0,0,.3,.3),
     main='Price vs. Living Sqft')
```



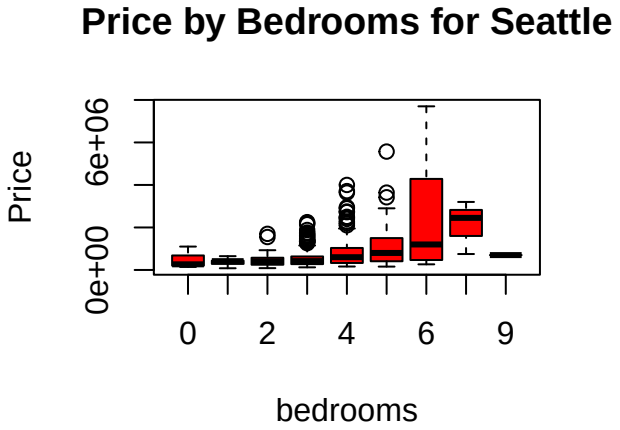
Histogram


```
hist(Seattle$price, xlab='Price', breaks='FD')
```



Boxplot

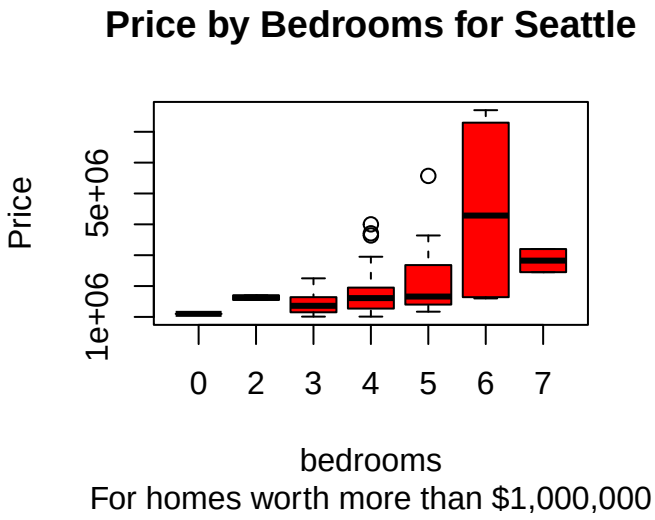

```
boxplot(Seattle$price ~ Seattle$bedrooms,
        ylab='Price', xlab='bedrooms', col='red',
        main='Price by Bedrooms for Seattle')
```



Exercise: Basic Plot

- Using only the subset of homes worth more than a million dollars, create a graphic.

Solution: Basic Plot



For homes worth more than $1,000,000

Solution: Basic Plot — Code


```
boxplot(expensive.houses$price ~ expensive.houses$bedrooms,
        ylab='Price', xlab='bedrooms', col='red',
        main='Price by Bedrooms for Seattle',
        sub='For homes worth more than $1,000,000')
```