Introduction to R for Beginners, Level II

Jeon Lee

Bio-Informatics Core Facility (BICF), UTSW

Basics of “R”

- Powerful programming language and environment for statistical computing
- Useful for very basic analysis as well as for hard core programming
- Very easy to make publication quality figures
- Run through command line or GUI
- Versions of R exist of Windows, MacOS, Linux and various other Unix flavors
- **Advantages:**
  - Free and open source
  - >2000 packages for handling biological data
  - Widely used, large user community (blogs like Seqanswers, StackOverflow…)
  - Extensive documentation with examples
  - Rstudio, a very user-friendly interface

Basic function

- List all the variables
  - ls()
- Examining a variable ‘x’
  - dim()
  - head()
  - tail()
  - class()
  - names()
- Removing variables
  - rm()
  - rm(list=ls()) # remove everything
- Get help
  - help() or ?
- Install packages in R environment
  - install.packages()

Objects

- R has five basic or “atomic” classes of objects

Example	Type
"a", "swc"	character
2, 15.5	numeric
2L (must add a L at end to denote integer)	integer
TRUE, FALSE	logical
1+4i	complex

- R objects can have attributes
  - names, dimnames
  - dimensions (e.g. matrices, arrays)
  - class
  - length
  - other user-defined attributes/metadata
  - Attributes of an object can be accessed using attributes()

Creating Vectors

- The c() function can be used to create vectors of objects

```
x <- c(0.5, 0.6)          ## numeric
x <- c(TRUE, FALSE)       ## logical
x <- c(T, F)              ## logical
x <- c("a", "b", "c")     ## character
x <- 9:29                 ## integer
x <- c(1+0i, 2+4i)        ## complex
```

- Using the vector() function

```
x <- vector("numeric", length = 10)
x
```

```
 [1] 0 0 0 0 0 0 0 0 0 0
```

Object coercion

- When different objects are mixed in a vector, coercion occurs so that every element in the vector is of the same class.

```
y <- c(1.7, "a")        ## character
y <- c(TRUE, 2)         ## numeric
y <- c("a", TRUE)       ## character
```

- Objects can be explicitly coerced from one class to another using the as.* functions, if available.

```
x <- 0:6
class(x)
```

```
[1] "integer"
```

```
as.numeric(x)
```

```
[1] 0 1 2 3 4 5 6
```

```
as.logical(x)
```

```
[1] FALSE TRUE TRUE TRUE TRUE TRUE TRUE
```

```
as.character(x)
```

```
[1] "0" "1" "2" "3" "4" "5" "6"
```

Matrix

- Matrices are vectors with a dimension attribute. The dimension attribute is itself an integer vector of length 2 (nrow, ncol)

```
m <- matrix(nrow = 2, ncol = 3)
m
```

```
     [,1] [,2] [,3]
[1,]   NA   NA   NA
[2,]   NA   NA   NA
```

```
dim(m)
```

```
[1] 2 3
```

```
attributes(m)
```

```
$dim
[1] 2 3
```

```
m <- matrix(1:6, nrow = 2, ncol = 3)
m
```

```
     [,1] [,2] [,3]
[1,]    1    3    5
[2,]    2    4    6
```

Matrix (cont.)

- Matrices can also be created directly from vectors by adding a dimension attribute.

```
m <- 1:10
m
```

```
 [1]  1  2  3  4  5  6  7  8  9 10
```

```
dim(m) <- c(2, 5)
m
```

```
     [,1] [,2] [,3] [,4] [,5]
[1,]    1    3    5    7    9
[2,]    2    4    6    8   10
```

- Matrices can be created by column-binding or row-binding with cbind() and rbind().

```
x <- 1:3
y <- 10:12
cbind(x, y)
```

```
     x  y
[1,] 1 10
[2,] 2 11
[3,] 3 12
```

```
rbind(x, y)
```

```
  [,1] [,2] [,3]
x    1    2    3
y   10   11   12
```

List

- Lists are a special type of vector that can contain elements of different classes.

```
x <- list(1, "a", TRUE, 1 + 4i)
x
```

```
[[1]]
[1] 1

[[2]]
[1] "a"

[[3]]
[1] TRUE

[[4]]
[1] 1+4i
```

Factor

- Factors are used to represent categorical data.
  - Factors are treated specially by modelling functions like lm() and glm().
  - The order of the levels can be set using the levels argument to factor().

```
x <- factor(c("yes", "yes", "no", "yes", "no"))
x
```

```
[1] yes yes no  yes no
Levels: no yes
```

```
attributes(x)
```

```
$levels
[1] "no"  "yes"

$class
[1] "factor"
```

```
table(x)
```

```
x
 no yes
  2   3
```

```
factor(c("yes", "yes", "no", "yes", "no"),
       levels = c("yes", "no"))
```

```
[1] yes yes no  yes no
Levels: yes no
```

Data frame

- Data frames are used to store tabular data
  - They are represented as a special type of list where every element of the list has to have the same length
  - Each element of the list can be thought of as a column and the length of each element of the list is the number of rows

```
x <- data.frame(foo = 1:4, bar = c(T, T, F, F))
x
```

```
  foo   bar
1   1  TRUE
2   2  TRUE
3   3 FALSE
4   4 FALSE
```

```
attributes(x)
```

```
$names
[1] "foo" "bar"

$row.names
[1] 1 2 3 4

$class
[1] "data.frame"
```

Data frame (cont.)

- Row and column names can be changed

```
row.names(x) <- paste("ID",1:4,sep = "")
x
```

```
      foo   bar
ID1   1  TRUE
ID2   2  TRUE
ID3   3 FALSE
ID4   4 FALSE
```

```
names(x) <- c("Number","Logic")
x
```

```
    Number  Logic
ID1      1   TRUE
ID2      2   TRUE
ID3      3  FALSE
ID4      4  FALSE
```

- Accessing the values

```
x$Number
```

```
[1] 1 2 3 4
```

```
x$Logic[1:2]
```

```
[1] TRUE TRUE
```

```
sapply(x,mean)
```

```
foo  bar
2.5  0.5
```

Subset

- Return subsets of vectors, matrices or data frames which meet conditions.
  - subset(x, subset, select, ...)
    - X : object to be subsetted.
    - subset: logical expression indicating elements or rows to keep: missing values are taken as false.
    - select: expression, indicating columns to select from a data frame.

```
s <- subset(x, Number>1, select=c(Logic))
s
```

```
      Logic
ID2    TRUE
ID3   FALSE
ID4   FALSE
```

```
subset(x, (Number>1)&(Logic==TRUE), select=c(Logic))
```

```
      Logic
ID2    TRUE
```

Merge

- Merge two data frames by common columns or row names

```
authors <- data.frame(surname = c("Tukey", "Venables", "Tierney", "Ripley", "McNeil"),
    nationality = c("US", "Australia", "US", "UK", "Australia"),
    deceased = c("yes", rep("no", 4)))
authors
```

	surname	nationality	deceased
1	Tukey	US	yes
2	Venables	Australia	no
3	Tierney	US	no
4	Ripley	UK	no
5	McNeil	Australia	no

```
books <- data.frame(
    name = c("Tukey", "Venables", "Tierney","Ripley", "Ripley", "McNeil", "R Core"),
    title = c("Exploratory Data Analysis", "Modern Applied Statistics","LISP-STAT",
          "Spatial Statistics", "Stochastic Simulation", "Interactive Data Analysis",
          "An Introduction to R"),
    other.author = c(NA, "Ripley", NA, NA, NA, NA,"Venables & Smith"))
```

Merge (cont.)

- By default the data frames are merged on the columns with names they both have, but separate specifications of the columns can be given by by.x and by.y.

```
merge(authors, books, by.x = "surname", by.y = "name")
```

	surname	nationality	deceased	title	other.author
1	McNeil	Australia	no	Interactive Data Analysis	<NA>
2	Ripley	UK	no	Spatial Statistics	<NA>
3	Ripley	UK	no	Stochastic Simulation	<NA>
4	Tierney	US	no	LISP-STAT	<NA>
5	Tukey	US	yes	Exploratory Data Analysis	<NA>
6	Venables	Australia	no	Modern Applied Statistics	Ripley

apply(X, MARGIN, FUN, ...)

- Returns a vector or array or list of values obtained by applying a function to margins of an array or matrix.

```
x <- cbind(x1 = 3, x2 = c(4:1, 3:6))
dimnames(x)[[1]] <- letters[1:8]
x
```

```
  x1 x2
a  3  4
b  3  3
c  3  2
d  3  1
e  3  3
f  3  4
g  3  5
h  3  6
```

```
apply(x, 2, mean)
```

```
 x1  x2
3.0 3.5
```

```
apply(x, 1, mean)
```

```
  a   b   c   d   e   f   g   h
3.5 3.0 2.5 2.0 3.0 3.5 4.0 4.5
```

lapply(X, FUN, ...)

- lapply returns a list of the same length as X, each element of which is the result of applying FUN to the corresponding element of X.
- lapply always returns a list, regardless of the class of the input.

```
x <- list(a = 1:5, b = rnorm(10))
lapply(x, mean)
```

```
$a
[1] 3

$b
[1] 0.2174559
```

lapply(X, FUN, ...)

- Example: using lapply with an anonymous function to extract the first column of each matrix.

```
x <- list(a = matrix(1:4, 2, 2), b = matrix(1:6, 3, 2))
lapply(x, function(elt) mean(elt[,1]))
```

```
$a
[1] 1.5

$b
[1] 2
```

sapply(X, FUN, ...)

- sapply will try to simplify the result of lapply if possible.
- If the result is a list where every element is length 1, then a vector is returned
- If it can't figure things out, a list is returned.

```
x <- list(a = 1:4, b = rnorm(10), c = rnorm(20, 1), d = rnorm(100, 5))
lapply(x, mean)
```

```
$a
[1] 2.5

$b
[1] 0.02414874

$c
[1] 1.359283

$d
[1] 5.135528
```

```
sapply(x, mean)
```

```
       a        b        c        d
2.500000 0.024148 1.359283 5.135528
```

Reading and exploring data

- read.table() function is the most important and commonly used function to import simple data files into R.
  - header argument specifies whether or not you have specified column names in your data file.

```
data <- read.table("gene_expression.txt", sep = "\t", header = T)
#try read.csv() also
dim(data)
```

```
[1] 34  4
```

```
head(data)  # Display first 6 rows of the data frame, cf. tail(data)
```

	genesymbol	condition1	condition2	condition3
1	P2RY2	12	55	110
2	SOX2	34	33	150
3	OLIG2	13	45	144
4	FOXA1	16	48	138
5	ESR1	16	88	128
6	NANOG	17	49	149

Reading and exploring data (cont.)

- Calculating basic stats
  - In many cases, functions perform column-wise calculation

```
colSums(data[ ,2:4])  # cf. rowSums()
```

```
condition1 condition2 condition3
       652        2147        5129
```

```
data_meaninfo <- apply(data[ , 2: 4], 2, mean) # cf. sapply(data[ ,2:4],median)
data_meaninfo
```

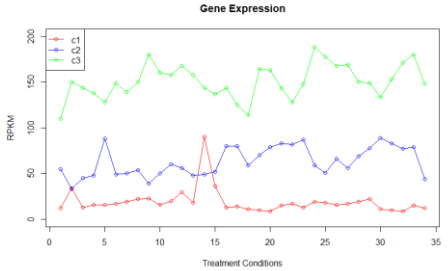
```
condition1   condition2   condition3
  19.17647     63.14706    150.85294
```

```
sapply(data[ ,2:4],sd)
```

```
condition1 condition2 condition3
  14.04855   16.07960   18.61432
```

Line plot

- A line chart is a graph that connects a series of points by drawing line segments.
  - type takes the value "p" to draw only the points, "l" to draw only the lines and "o" to draw both points and lines.

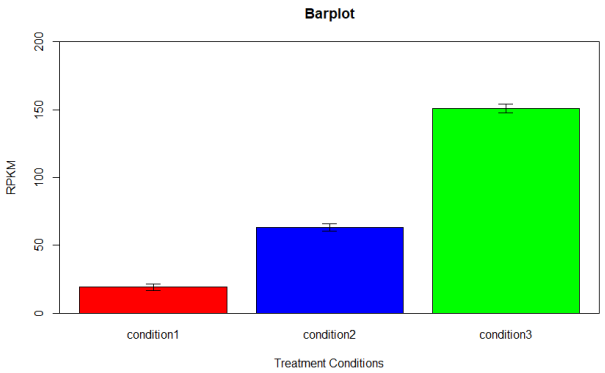



```
# Give the chart file a name
png("GE_linePlot.png")

# Plot the line chart
plot(data$condition1, type = "o", col = "red", ylim = c(0, 200),
    xlab = "Treatment Conditions", ylab = "RPKM", main = "Gene Expression")
lines(data$condition2, type = "o", col = "blue")
lines(data$condition3, type = "o", col = "green")
legend ("topleft",legend=c("c1", "c2", "c3"), col = c("red", "blue", "green"),
     lty=1,pch=1)

# Save the file
dev.off()
```

Bar plot + user-defined function

- barplot() creates a bar plot with vertical or horizontal bars.
- source() makes it possible to read R code from a file

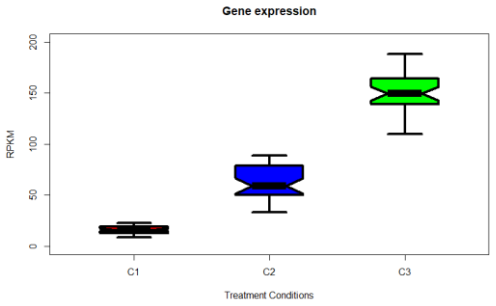
```
# Selecting some columns from the data frame
df <- data[, 2: 4]
# Read a user-defined function to plot error bars
source("ErrorBar_function.R")
# Calculate column means
my_mean <- apply(df, 2, mean)
# Calculate sem (a user defined function)
my_sem <- apply(df, 2, sem)
# Plot a bar chart with different colors
barx <- barplot(my_mean, names.arg = names(df),ylim = c(0, 200),
          xlab = "Treatment Conditions",ylab = "RPKM",
          col = c("red", "blue", "green"),main="Barplot")
box()
error.bar(barx, my_mean, my_sem)
```



Box plot

- boxplot() produces box-and-whisker plot(s) of the grouped values.
  - IQR: interquantile range, “middle fifty (Q3-Q1)” range.
  - outliers are defined as data points, which are greater than (0.5×IQR+Q3) or lower than (Q1-0.5×IQR).

```
# Basic boxplot
boxplot(data$condition1, data$condition2, data$condition3,
            ylim=c(0,200))
# Plot the boxplot
boxplot(data$condition1, data$condition2, data$condition3,
    col = c("red", "blue", "green"),
    names = c("C1", "C2", "C3"),
    pin=c(5,5),notch = TRUE,
    outline = FALSE, lwd = 4, lty = 1,
    cex.axis = 1, ylim = c(0, 200),
    boxwex = 0.5,main = "Gene expression",
    xlab = "Treatment Conditions", ylab = "RPKM")
```

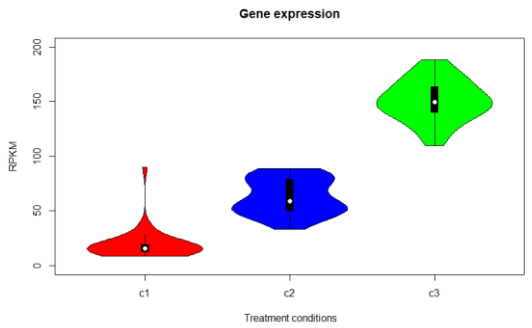



```
#pin: plot dimensions (width, height) in inches
#cex.axis: magnification of axis annotation
#outline: logic whether or not to plot outliers
#lwd: line width of the box
#boxwex: width of the box
#lty: line type
#cex: num. by which plotting text should be scaled
#pch: symbol type (circle or square etc)
```

Violin plot

- boxplot() produces box-and-whisker plot(s) of the grouped values.

```
library(vioplot)
vioplot(data$condition1, data$condition2, data$condition3, col = c("red"), horizontal = F)
title("Gene expression")

# Add more violins to the plot by setting add=TRUE
# Open a plot first
plot(1, 1, xlim = c(0.5, 3.5), ylim = c(0, 200), type = "n", xlab = " Treatment conditions",
      ylab = "RPKM", main="Gene expression", axes=FALSE)
## Draw the axes with labels and then plot the violin plots one after another
axis(side = 1, at = 1: 3, labels = c("c1", "c2", "c3"))
axis(side = 2)
vioplot(data$condition1, col = "red", at = 1, add = T)
vioplot(data$condition2, col = "blue", at = 2, add = T)
vioplot(data$condition3, col = "green", at = 3, add = T)
```



What Is The Grammar Of Graphics?

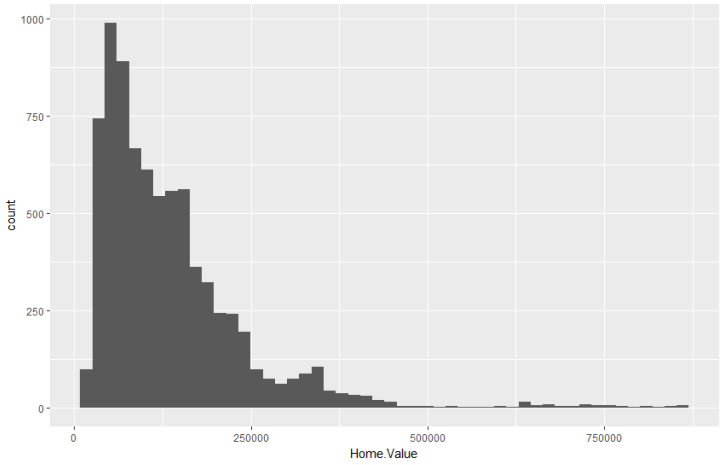
- The basic idea: independently specify plot building blocks and combine them to create just about any kind of graphical display you want. Building blocks of a graph include:
  - data: in data.frame form
  - aesthetic: map variables into properties people can perceive visually ... axis, position, color, shape, line type?
  - geom: specifics of what people see ... points? lines?
  - scale: map data values into “computer” values
  - stat: summarization/transformation of data
  - facet: juxtapose related mini-plots of data subsets

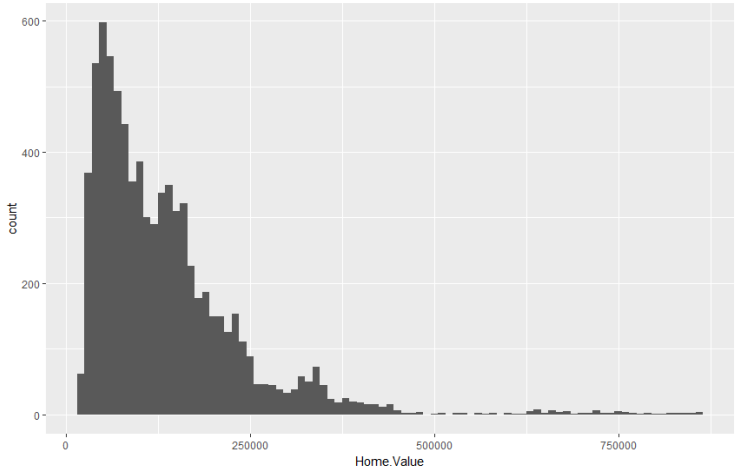
  - for more information, refer to “Data Visualization with ggplot2” provided (ggplot2-cheatsheet.pdf)

Histogram

- Histogram is an one variable plot
- ggplot() + geom_histogram()

```
housing <- read.csv("landdata-states.csv")
library(ggplot2)
ggplot(housing, aes(x = Home.Value)) +
    geom_histogram(bins=50)
```

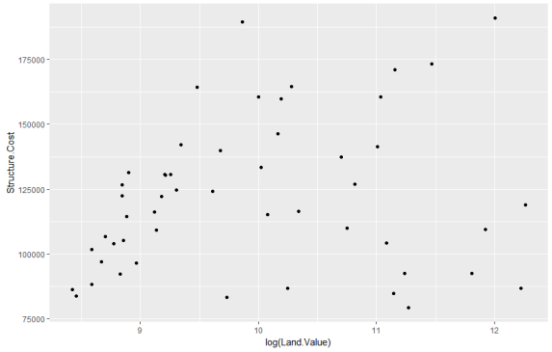





```
p <- ggplot(housing, aes(x = Home.Value))
p + geom_histogram(binwidth=10000)
```

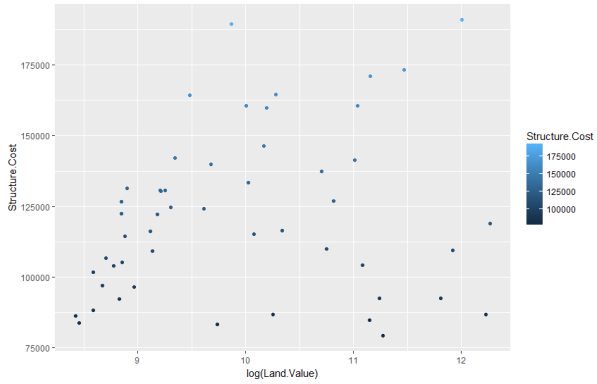
Points (Scatterplot)

- Scatter plot takes two variables.

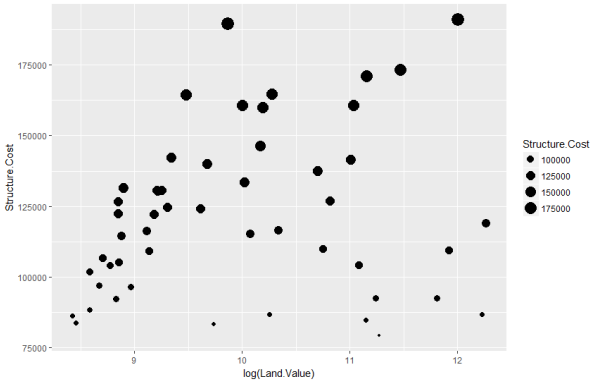



```
hp2001Q1 <- subset(housing, Date == 2001.25)
p <- ggplot(hp2001Q1, aes(y = Structure.Cost, x = log(Land.Value)))
p + geom_point()
```

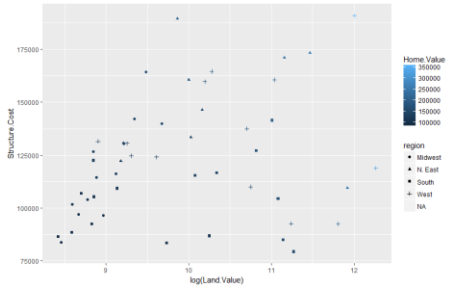
```
p + geom_point(aes(color = Structure.Cost))
```




```
p + geom_point(aes(size = Structure.Cost))
```



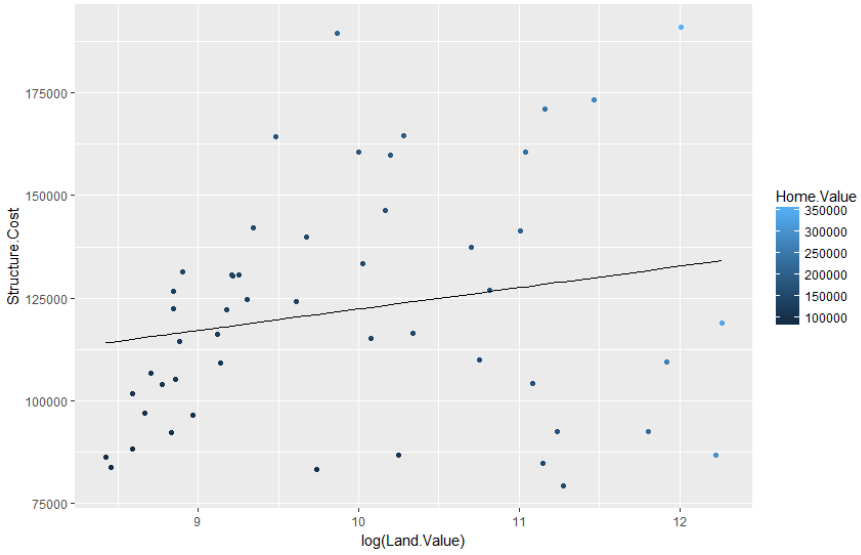

```
p + geom_point(aes(color=Home.Value, shape = region))
```



Lines (Prediction Line)

- A plot constructed with ggplot can have more than one geom.

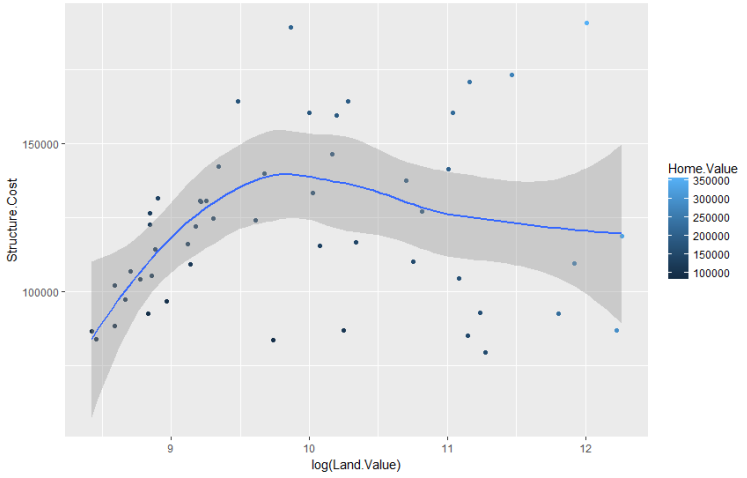
```
hp2001Q1$pred.SC <- predict(lm(Structure.Cost ~ log(Land.Value),
                               data = hp2001Q1))
p <- ggplot(hp2001Q1, aes(x = log(Land.Value), y = Structure.Cost))
p + geom_point(aes(color = Home.Value)) + geom_line(aes(y = pred.SC))
```

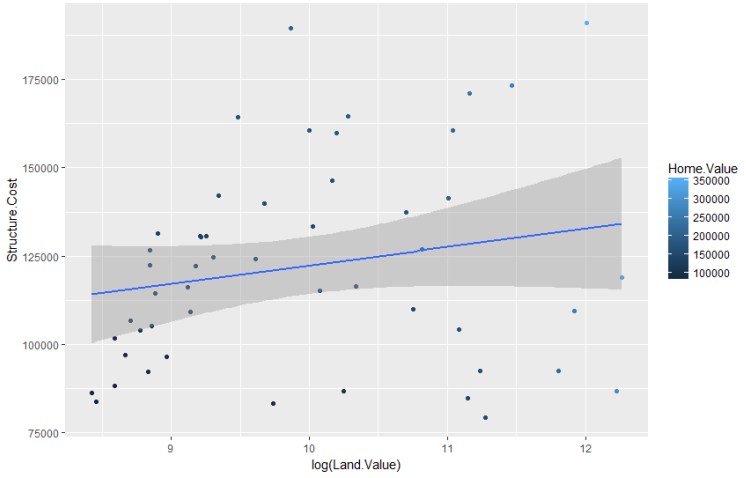


Smoothers

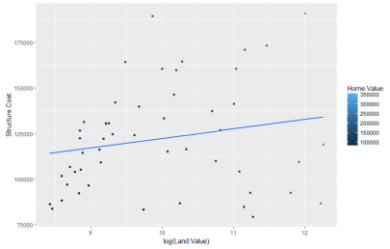
- The default for geom_smooth is a loess fit.

```
p + geom_point(aes(color = Home.Value)) + geom_smooth()
p + geom_point(aes(color = Home.Value)) + geom_smooth(method="lm")
```





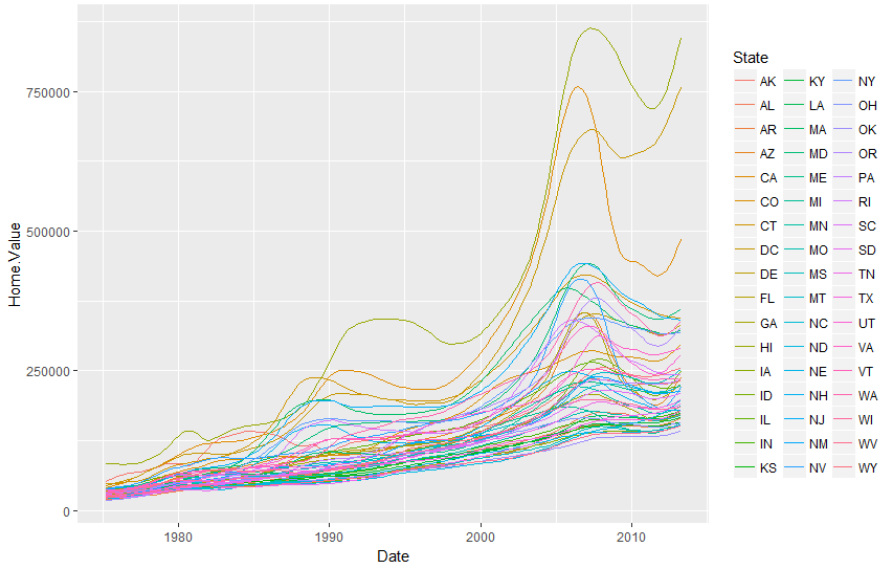

```
p + geom_point(aes(color = Home.Value)) +
    geom_smooth(method="lm", se = FALSE)
```



Faceting

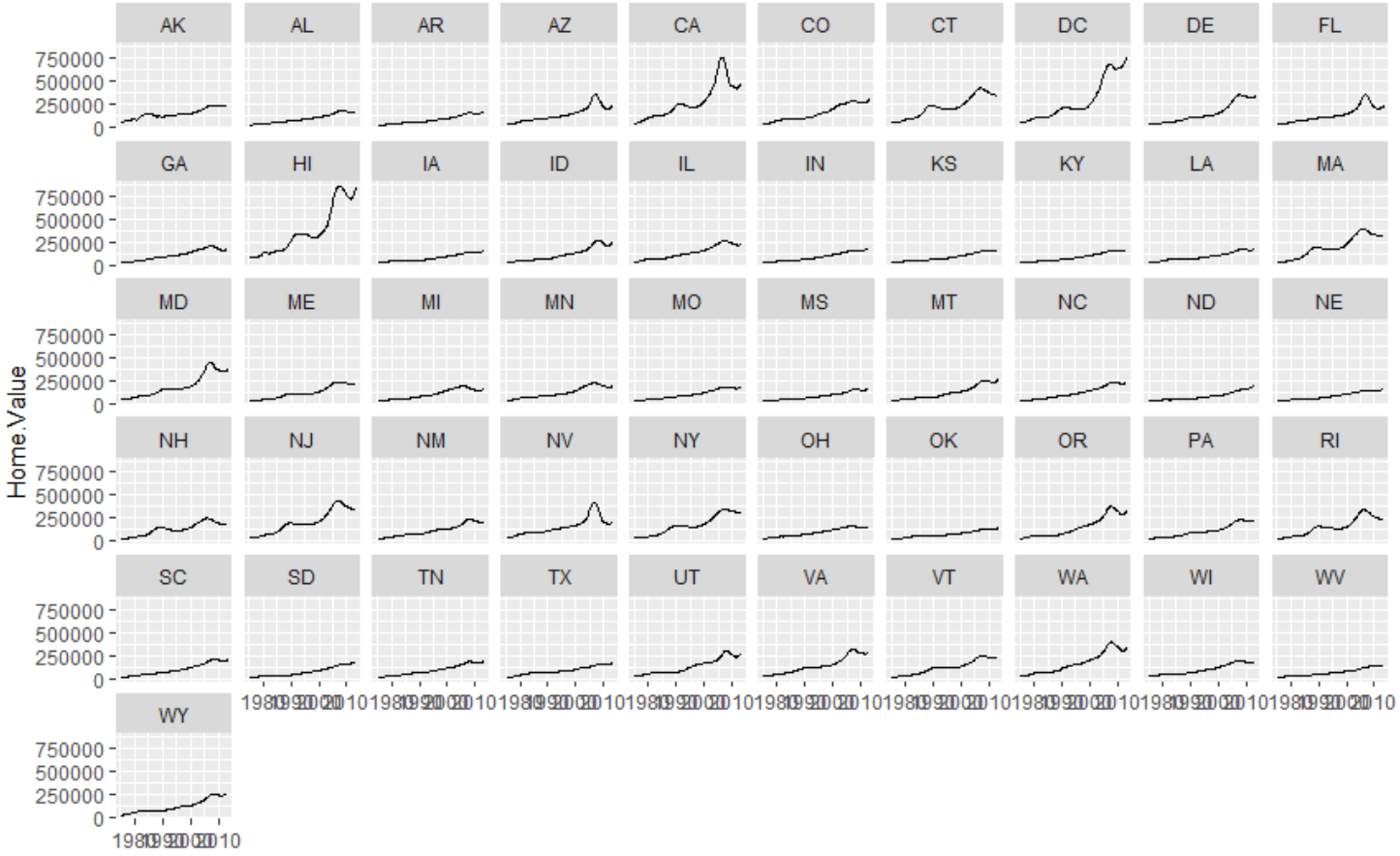
- The idea is to create separate graphs for subsets of data
  - facet_wrap(): define subsets as the levels of a single grouping variable
- The trend in housing prices in each state

```
p <- ggplot(housing, aes(x = Date, y = Home.Value))
p + geom_line(aes(color = State))
```



Faceting (cont.)

p + geom_line() + facet_wrap(~State, ncol = 10)



Box plot

- The boxplot compactly displays the distribution of a continuous variable.
  - Keep in mind that the x-axis should assigned to a factor variable

```
p <- ggplot(housing, aes(x = region, y = Home.Value))
p + geom_boxplot(aes(fill=region))
```