Programación Básica y Gráficas en R

Code

1.Introducción

- En comparación con otros paquetes como Stata, R tiene un costo de entrada relativamente mayor.
- Sin embargo, este costo se cubre rápidamente debido a la versatilidad del programa para diversos procesos.
- Programación y gráficas son dos campos en los que R se desempeña relativamente bien.
- Estos dos campos son universos en sí mismos.
- No alcanzamos (ni es el objetivo del curso) cubrir todos los detalles asociados a estos temas.
- Pero introduciremos algunos aspectos básicos de cada campo.
- En primer lugar, veremos en esta lección los fundamentos de las funciones y los loops en R, que son la piedra angular en programación.
- Sin que esto implique que aprendamos a programar. Apenas meteremos el dedo gordo del pie en un oceáno de profundidad.
- Algo parecido en cuanto a gráficas.
- Veremos los fundamentos de `ggplot2` , acaso el paquete de R más versatil para hacer gráficas.
- Lectores interesados en aprender a programar (en forma) pueden remitirse al libro de Matloff (2011), “The Art of R Programming”.
- Para más en gráficas y `ggplot2` Whickham (2009), “ggplot2. Elegant Graphics for Data Analysis” es una opción interesante.

2. Funciones

- Como en la mayoría de lenguajes de programación, el corazón de programar son las funciones.
- Una función es un grupo de instrucciones que toma insumos, los usa para computar otros valores y devuelve un resultado.
- La típica función tiene la siguiente estructura:

```
function.name <- function(arguments)

{

computations on the arguments

some other code

}
```

- Así, las funciones suelen tener un nombre.
- Los argumentos usados como insumos.
- Un cuerpo, que es el código dentro de los corchetes {}. Es donde llevamos a cabo los cálculos.
- Y uno o más valores retornados.
- Empecemos con una función simple que nos devuelva el cuadrado de cualquier número:

Hide

Hide

```
funcion_cuadrado <- function(n)
{
  u=n*n
  return(u)
}
```

Hide

Hide

```
funcion_cuadrado(5)
```

```
[1] 25
```

- Listo: hemos programado nuestra primera función.
- Naturalmente, no teníamos que definirla porque R ya le tiene incorporada: `n^2`
- Probemos con una función más compleja.
- Una que nos permita contar el número de elementos impares que tiene un vector.
- Para ello, primero conozcamos el operador módulo `%%` .
- Este operador devuelve el residuo de la división de dos números. Por ejemplo:

Hide

Hide

```
38 %% 7
```

```
[1] 3
```

- Porque 7 cabe 5 veces en 38, y sobran 3. O por ejemplo,

Hide

Hide

```
100 %% 5
```

```
[1] 0
```

- Porque 100 es un múltiplo de 5.
- Entendiendo este operador podemos programar la función que cuente el número de impares en un vector.
- Porque un impar es un número que al dividirse por 2 deja un residuo de 1:

Hide

Hide

```
55 %% 2
```

```
[1] 1
```

- Así, nuestra función es:

Hide

Hide

```
numeroimpares <- function(x)
{
  k <- 0
  for (n in x)
  {
    if (n %% 2 ==1) k <- k+1
  }
  return(k)
}
```

- Vamos paso a paso:
- Primero, le dimos un nombre a la función.
- Establecimos que el insumo de la función es un vector llamado `x`.
- Luego abrimos el cuerpo del código.
- Este empieza estableciendo un escalar `k` en 0.
- Luego le pedimos que para cada elemento `n` del vector `x`, en orden desde el primero hasta el último, se use el operador módulo.
- Si el resultado es 1, se le agrega una unidad al valor de `k` que venimos arrastrando.
- Probemos la función con el vector `c(5, 6, 7, 8)`. En este caso, el vector tiene 2 impares.

Hide

Hide

```
a <- c(5, 6, 7, 8)
numeroimpares(a)
```

```
[1] 2
```

- Funciona bien. También podemos incluir directamente el vector:

Hide

Hide

```
numeroimpares(c(1, 5, 7, 9, 11))
```

```
[1] 5
```

Ejercicio

a. Cree una función que para cualquier escalar, devuelva el cuadrado del logaritmo natural de dicho escalar multiplicado por 10. Pruebe que la función sirve con el escalar 1 y con algún otro escalar.

b. Cree una función que cuente, para cualquier matriz, el número de elementos que son múltiplos de 3. Pruebe su función con una matriz 3x3 que tenga 4 múltiplos de 3.

3. Iteraciones y loops

- En la definición de funciones, los loops son un elemento esencial.
- Un loop es una forma de repetir un secuencia de instrucciones bajo ciertas condiciones.
- Permite automatizar partes de un código que necesitan repetición.
- En la función anterior usamos un loop: `for (n in x) {`
- Le pedimos a R que repitiera una acción para cada elemento en x.
- Significa que hace una iteración del loop para cada componente del vector `x`.
- En la primera iteración, `n=x[1]`. En la segunda, `n=x[2]`. Y así sucesivamente.
- Veamos otro ejemplo:

Hide

Hide

```
x <- c(5, 12, 13)
for (n in x)
  print(n^2)
```

```
[1] 25
[1] 144
[1] 169
```

- Acá le pedimos que a cada elemento del vector le saque el cuadrado.
- Otras declaraciones interesantes, con las que podemos generar loops, son `while` y `repeat`.

- Por ejemplo, consideremos el siguiente loop:

Hide

Hide

```
i <- 1
while (i<=10) i <- i+4
i
```

```
[1] 13
```

- Primero establecimos `i` igual 1.
- Y dijimos que siempre que i sea menor o igual a 10, se convierta en `i+4` .
- El ciclo llega hasta `i=9` , que se convierte en 13 y se detiene.
- Esto lo podemos lograr con otra declaración, incluyendo el operador `break` .

Hide

```
i <- 1
repeat
  {
  i <- i+4
  if (i>10) break
  }
i
```

```
[1] 13
```

4. Gráficas

- La visualización es clave en data science.
- Una imagen dice más que mil palabras.
- De hecho, una de las fortalezas de R es que permite hacer gráficas elegantes.
- Usaremos el paquete `ggplot2` , que es tal vez el más poderso y generalizado.

- Lo primero que debemos entender es la gramática de una gráfica.
- Los bloques de un gráfica son los siguientes:
    - Datos
    - Mapeo estético
    - Objeto geométrico
    - Transformaciones estadísticas
    - Escalas
    - Sistema de coordenadas
    - Ajustes de posición
    - Facetado

4.1 Gráficas básicas vs. `ggplot2`

- R tiene sus propias funciones incorporadas para hacer gráficas.
- Pero el paquete `ggplot2` es mucho más poderoso.
- Ilustremos esto con un ejemplo.
- Trabajaremos con la base `landdata-states.csv` que está en la carpeta de esta semana.
- Esta es una base sobre precios de la vivienda en USA.

Hide

Hide

```
housing <- read.csv("landdata-states.csv")
```

- Inspeccionemos la base:

Hide

Hide

```
str(housing)
```

```
'data.frame':   7803 obs. of  11 variables:
 $ State           : Factor w/ 51 levels "AK","AL","AR",..: 1 1 1 1 1 1 1 1 1 1 ...
 $ region          : Factor w/ 4 levels "Midwest","N. East",..: 4 4 4 4 4 4 4 4 4 4 .
..
 $ Date            : num  2010 2010 2010 2010 2008 ...
 $ Home.Value      : int  224952 225511 225820 224994 234590 233714 232999 232164 231
039 229395 ...
 $ Structure.Cost  : int  160599 160252 163791 161787 155400 157458 160092 162704 164
739 165424 ...
 $ Land.Value      : int  64352 65259 62029 63207 79190 76256 72906 69460 66299 63971
...
 $ Land.Share..Pct.: num  28.6 28.9 27.5 28.1 33.8 32.6 31.3 29.9 28.7 27.9 ...
 $ Home.Price.Index: num  1.48 1.48 1.49 1.48 1.54 ...
 $ Land.Price.Index: num  1.55 1.58 1.49 1.52 1.89 ...
 $ Year            : int  2010 2010 2009 2009 2007 2008 2008 2008 2008 2009 ...
 $ Qrtr            : int  1 2 3 4 4 1 2 3 4 1 ...
```

- Otra forma de ver los datos es por medio del comando head:

Hide

Hide

```
head(housing)
```

	State <fctr>	region <fctr>	Date <dbl>	Home.Value <int>	Structure.Cost <int>
1	AK	West	2010.25	224952	160599
2	AK	West	2010.50	225511	160252
3	AK	West	2009.75	225820	163791
4	AK	West	2010.00	224994	161787
5	AK	West	2008.00	234590	155400
6	AK	West	2008.25	233714	157458

6 rows | 1-6 of 11 columns

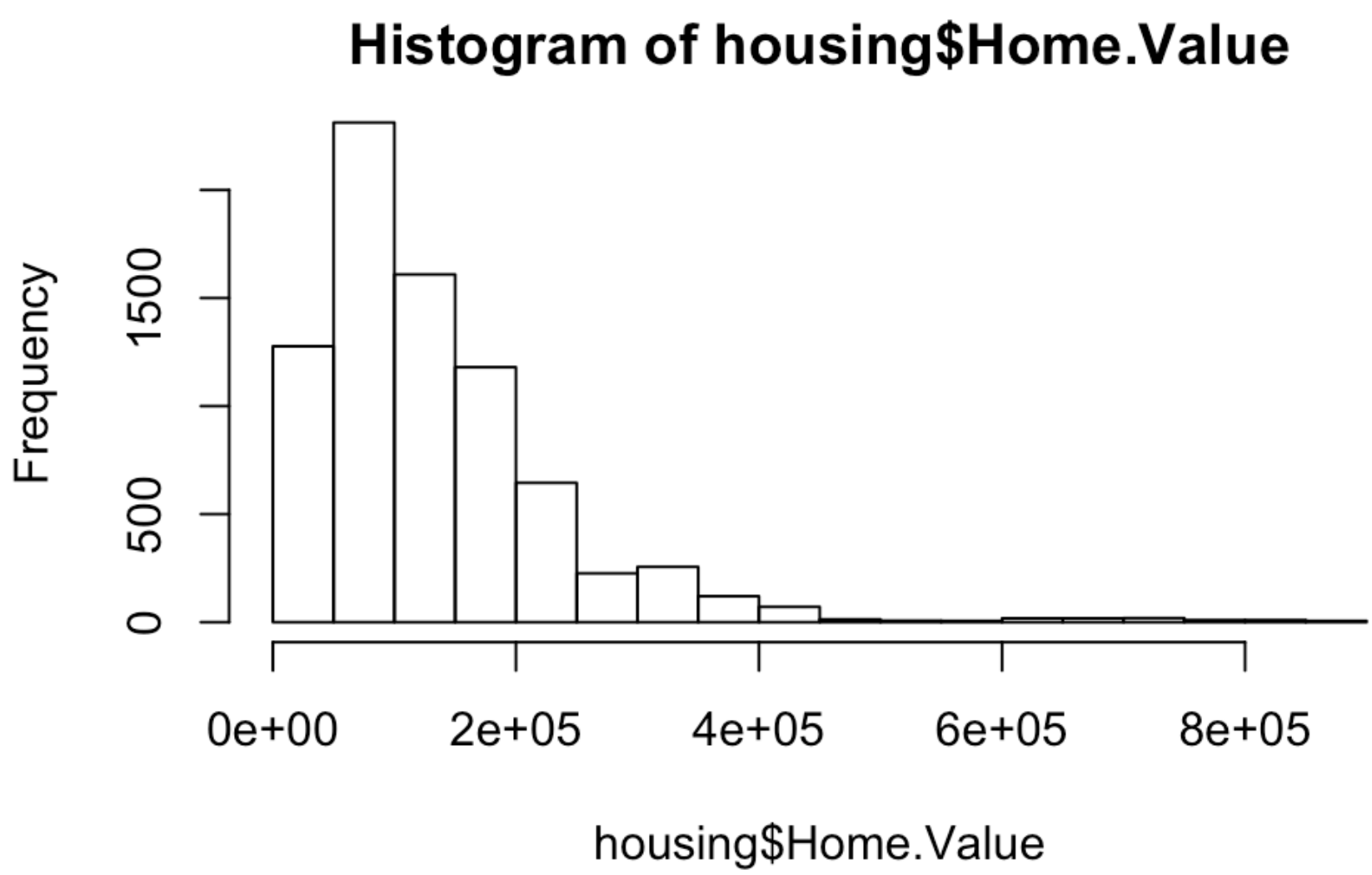
- Tenemos variables interesantes como el estado, el valor, la región, la antiguedad, entre otras.
- Las gráficas nos ayudan a entender estos datos.
- Comparemos gráficas tradicionales de R con las de `ggplot2`

- Empezemos con un histograma del precio de las viviendas. Con funciones básicas de R tenemos:

Hide

Hide

```
hist(housing$Home.Value)
```



- Ahora hagamos el histograma en `ggplot2` :

Hide

Hide

```
install.packages("ggplot2")
```

```
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed

  0     0    0     0    0     0      0      0 --:--:-- --:--:-- --:--:--     0
  0     0    0     0    0     0      0      0 --:--:--  0:00:01 --:--:--     0
 44 2694k   44 1199k    0     0   601k      0  0:00:04  0:00:01  0:00:03  601k
 86 2694k   86 2319k    0     0   741k      0  0:00:03  0:00:03 --:--:--  741k
100 2694k  100 2694k    0     0   860k      0  0:00:03  0:00:03 --:--:--  860k
tar: Failed to set default locale
```

```
The downloaded binary packages are in
    /var/folders/cw/_9zpljws58jbbv49mcb86l6c0000gn/T//Rtmp4wL8Gz/downloaded_packages
```

Hide

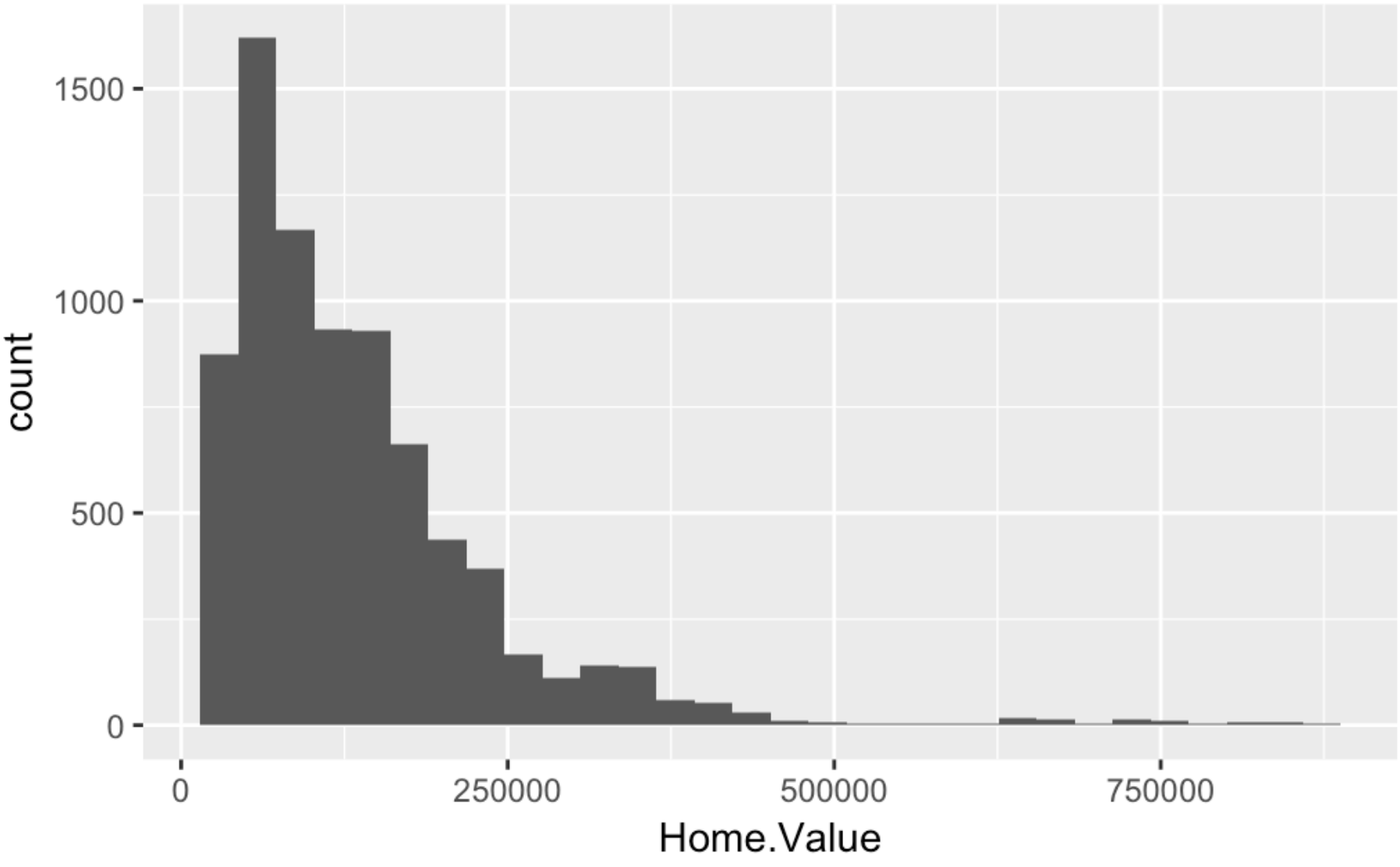
Hide

```
library(ggplot2)
```

```
package 'ggplot2' was built under R version 3.2.5
```

Hide

```
ggplot(housing, aes(x = Home.Value))+geom_histogram()
```



- Para gráficas sencillas, como este histograma, la sintaxis de `ggplot2` es más compleja.
- Pero para gráficas complicadas, es más fácil con `ggplot2`.
- Veamos esto con una gráfica de la serie de tiempo del valor de las casas.
- En la versión simple:

Hide

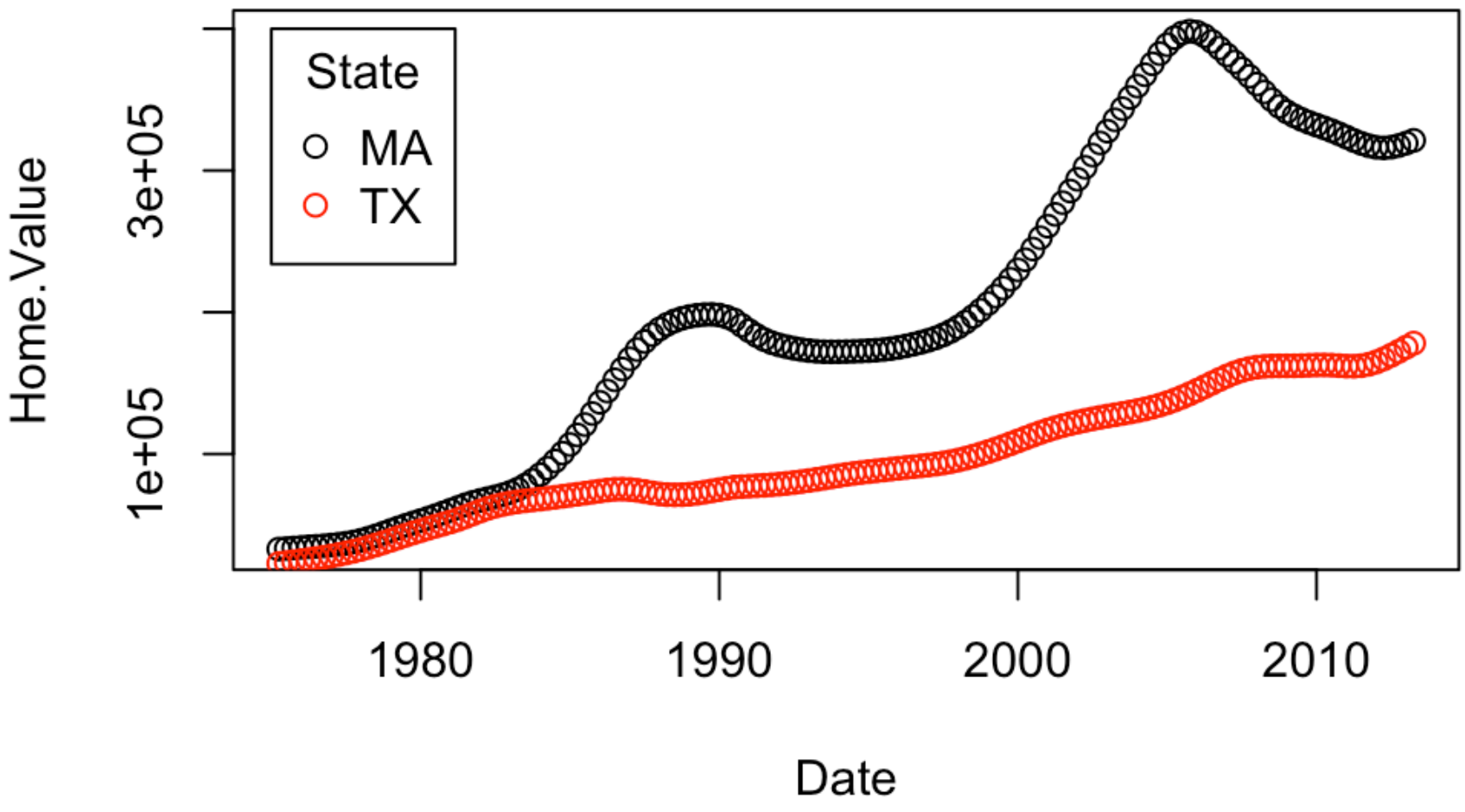
Hide

```
plot(Home.Value ~ Date, data=subset(housing, State=="MA"))
points(Home.Value ~ Date, col="red", data=subset(housing, State=="TX"))
```

Hide

Hide

```
legend(1975, 400000,
       c("MA", "TX"), title="State",
       col=c("black", "red"),
       pch=c(1, 1))
```

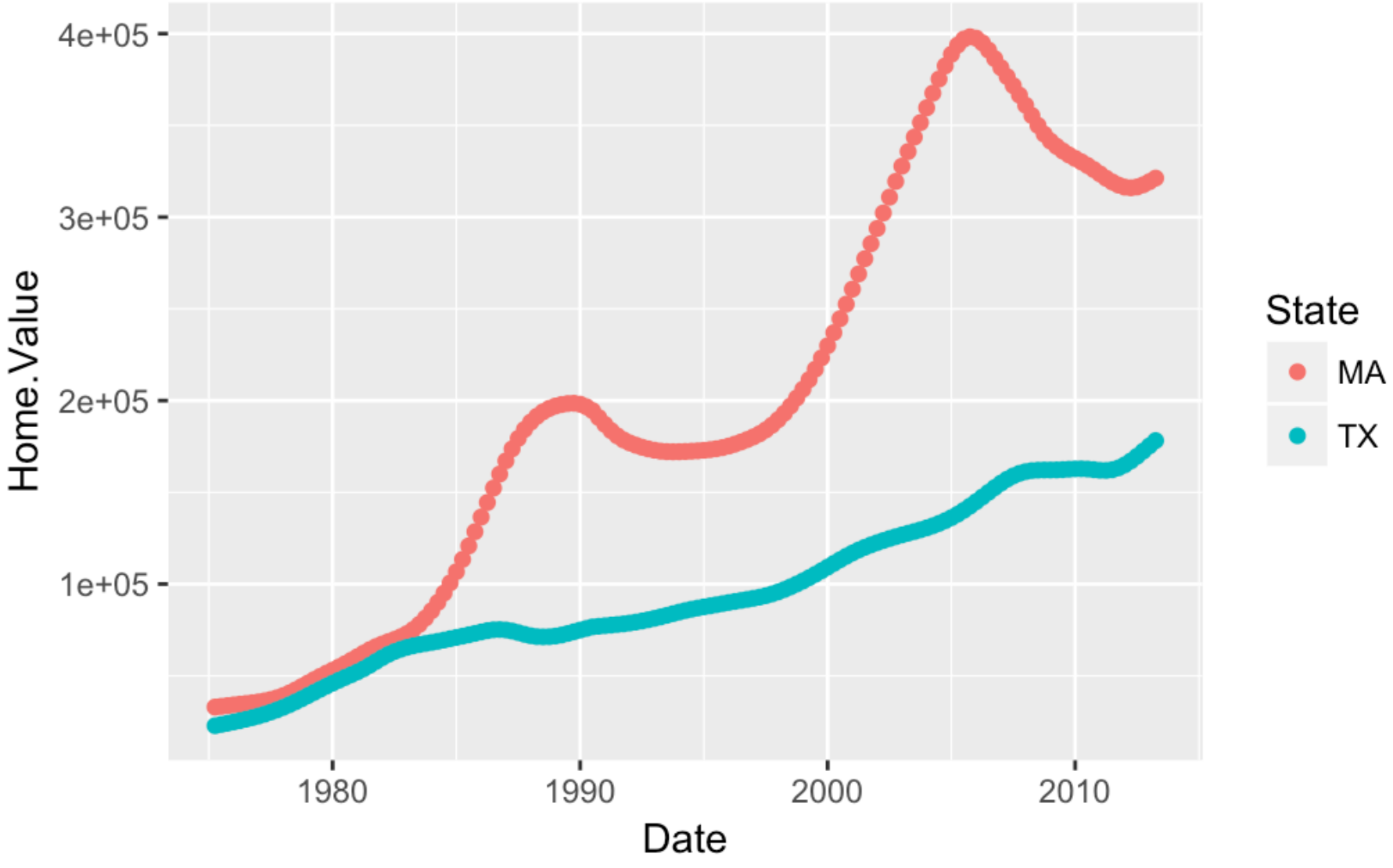


- La primera línea grafica la serie para MA, la segunda para TX, y la tercera es la leyenda.
- Un tanto complejo el proceso.
- Veamos como sería en `ggplot2` .

Hide

Hide

```
ggplot(subset(housing, State %in% c("MA", "TX")), aes(x=Date, y=Home.Value, color=Sta
te))+geom_point()
```



- En este caso, claramente, es mucho más fácil hacer la gráfica en ggplot.
- Entendamos este lenguaje.

4.2 Objetos geométricos y estética

Mapeo estético

- En `ggplot2` por estético entendemos lo que podemos ver.
- Por ejemplo, posición, color, relleno, forma, tipo de línea, tamaño.
- Cada objeto geométrico acepta solo un subconjunto de opciones estéticas.
- El mapeo estético se establece con la función `aes()`.

Objetos geométricos

- Los objetos geométricos son las marcas que ponemos en una gráfica. Por ejemplo:
    - Puntos (`geom_point`)
    - Líneas (`geom_line`)
    - Boxplots (`geom_boxplot`)
- Una gráfica debe tener al menos un `geom` . Puede tener más de uno.
- Con el operador `+` se pueden añadir más `geom` .
- Para un listado de todos los objetos geométricos disponibles usar:

Hide

Hide

```
help.search("geom_", package="ggplot2")
```

- Veamos algunos ejemplos

Puntos (Scatterplot)

- Con la funcion `aes` y los objetos geométricos podemos hacer gráficas en `ggplot2` .
- Como mínimo necesitamos mapear x y y. Luego hay algunas opciones:

Hide

Hide

```
ggplot(housing, aes(y=Land.Value, x=Home.Value, col="red"))+geom_point()
```

- Tenemos demasiadas observaciones como para ver algo claro.
- Acotemos el análisis únicamente al primer trimestre de 2002

Hide

Hide

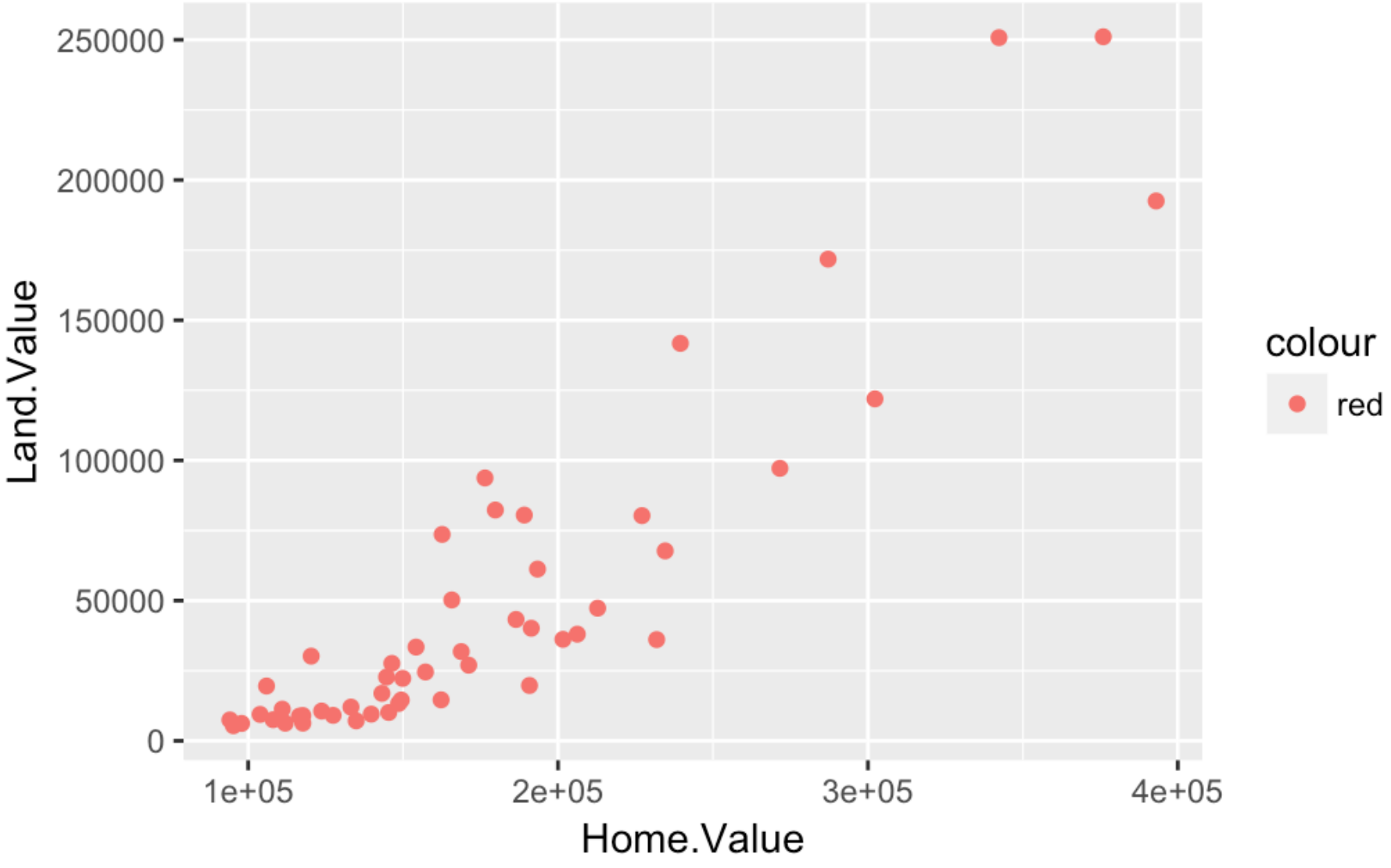
```
housing2002Q1 <- subset(housing, Date==2002.25)
```

- Y construyamos el scatterplot solo para este periodo:

Hide

Hide

```
ggplot(housing2002Q1, aes(y=Land.Value, x=Home.Value, col="red"))+geom_point()
```

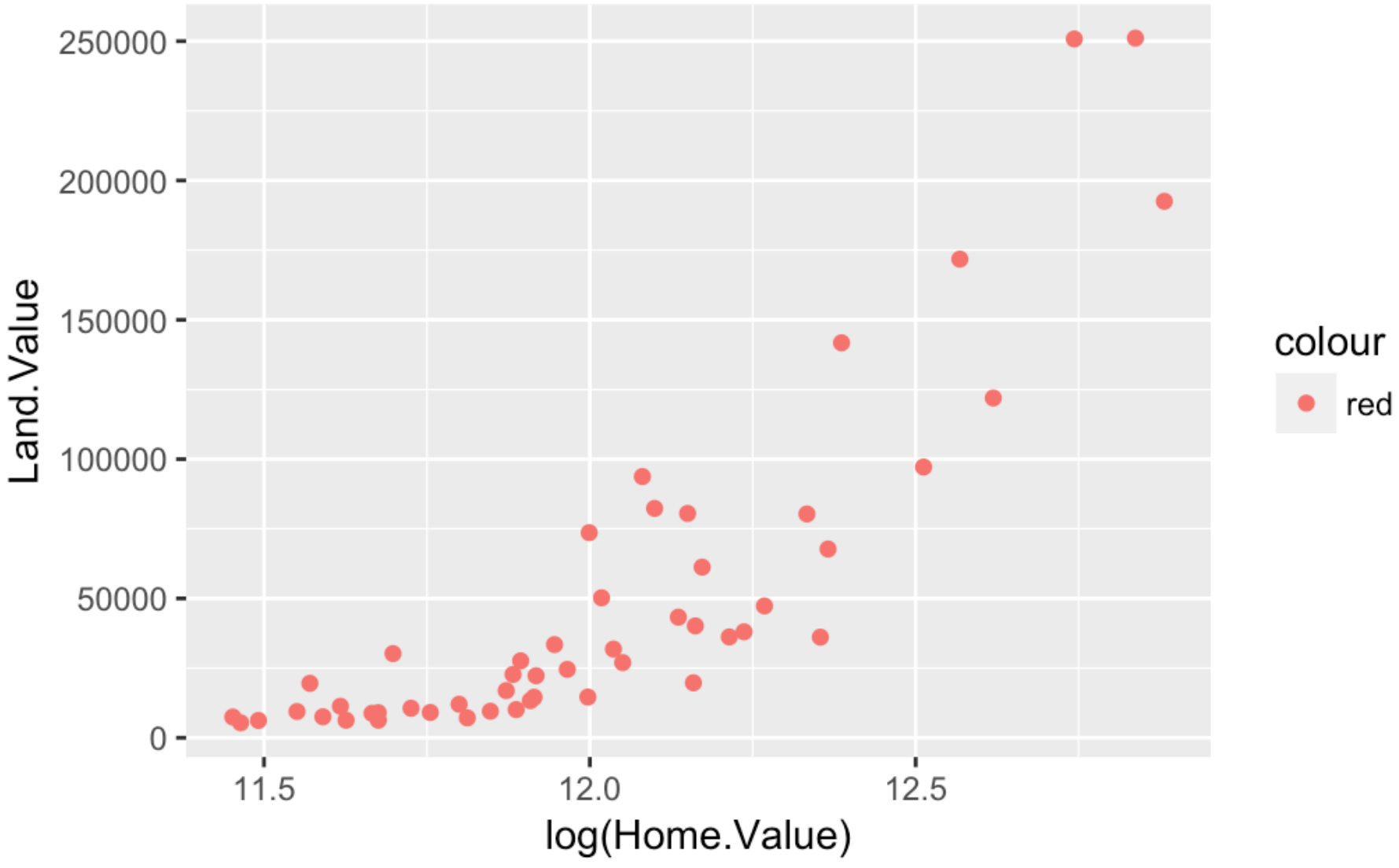


- Mucho más claro.
- Podemos hacer transformaciones a variables y graficarlas:

Hide

Hide

```
ggplot(housing2002Q1, aes(y=Land.Value, x=log(Home.Value), col="red"))+geom_point()
```



- Nótese la escala en el eje x.

Líneas (línea de predicción)

- Podemos tener más de un objeto geométrico en una misma gráfica.
- Por ejemplo, podemos graficar la recta de regresión a la gráfica que ya tenemos.
- Primero, corramos la regresión de `Land.Value` contra `Home.Value` y pronostiquemos:

Hide

Hide

```
housing2002Q1$pred.LV <- predict(lm(Land.Value ~ log(Home.Value), data = housing2002Q
1))
```

- Definamos el plano:

Hide

Hide

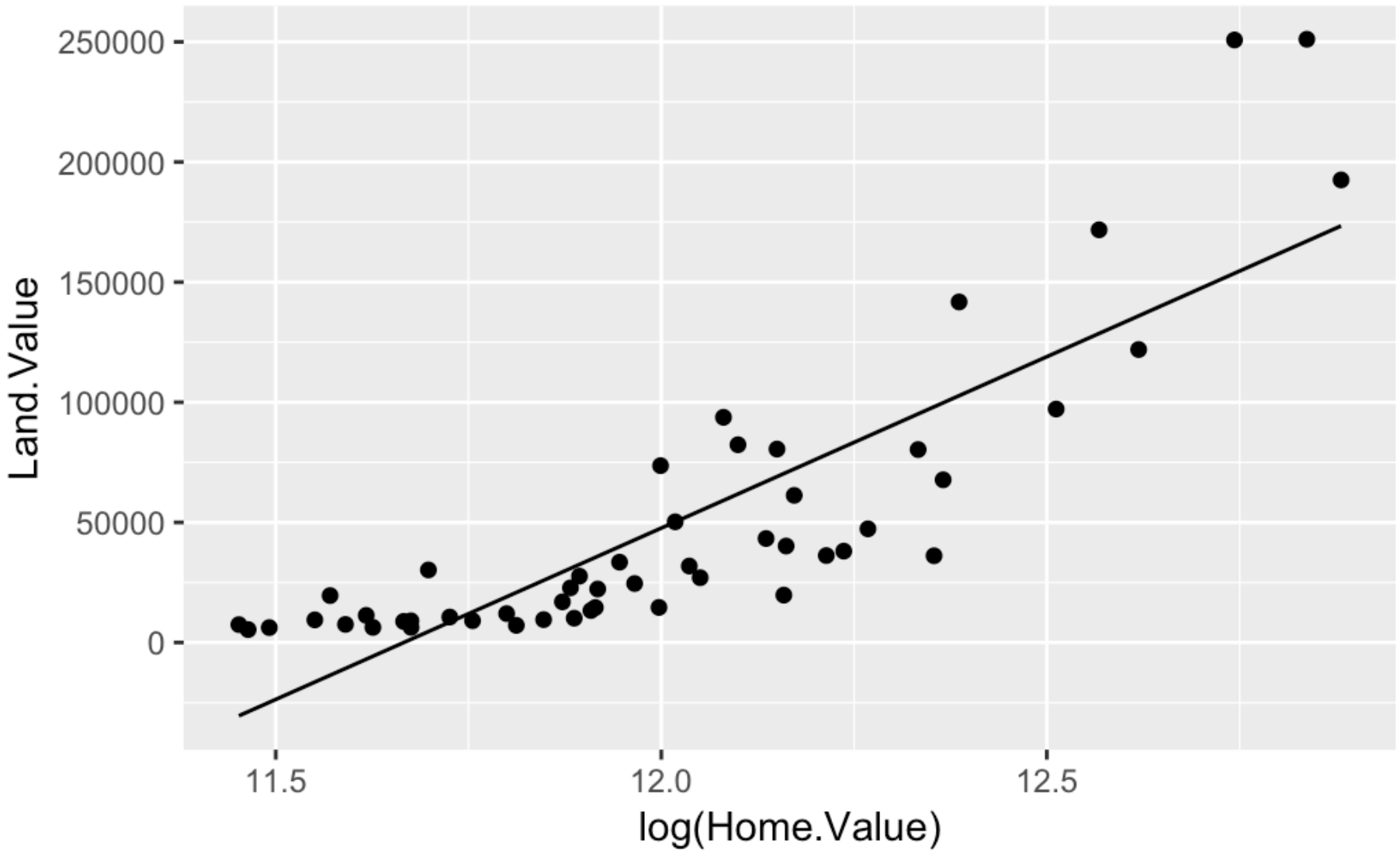
```
p1 <- ggplot(housing2002Q1, aes(x=log(Home.Value), y=Land.Value))
```

- Grafiquemos en dicho plano las observaciones y la recta de regresión

Hide

Hide

```
p1 + geom_point()+geom_line(aes(y=pred.LV))
```

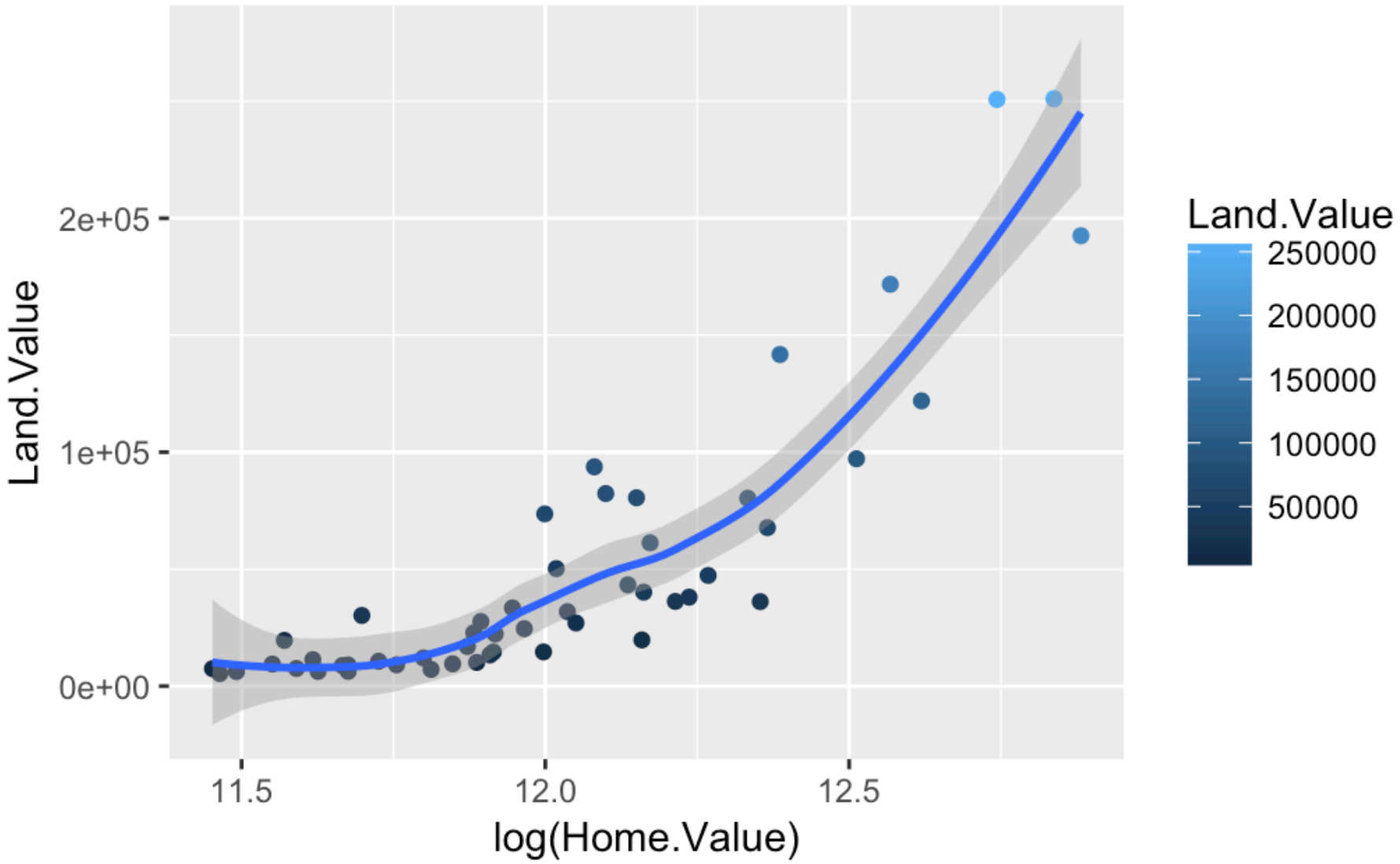


- Hay otro tipo de ajustes más sofisticados (no-paramétricos):

Hide

Hide

```
p1 + geom_point(aes(color=Land.Value))+geom_smooth()
```



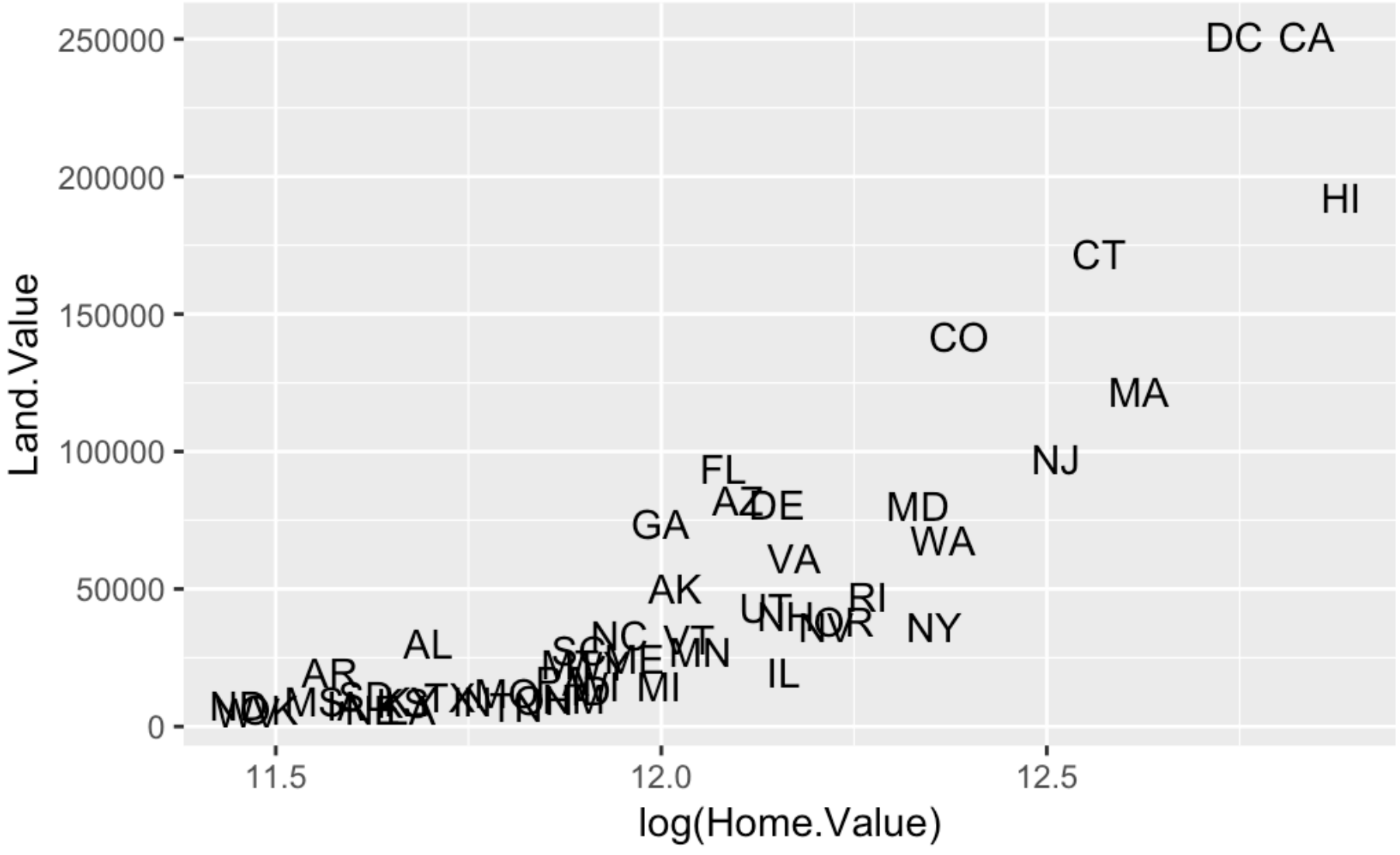
Texto (Label Points)

- Cada `geom` permite algunos mapeos particulares.
- Por ejemplo, `geom_text` acepta `labels`.

Hide

Hide

```
p1+geom_text(aes(label=State))
```

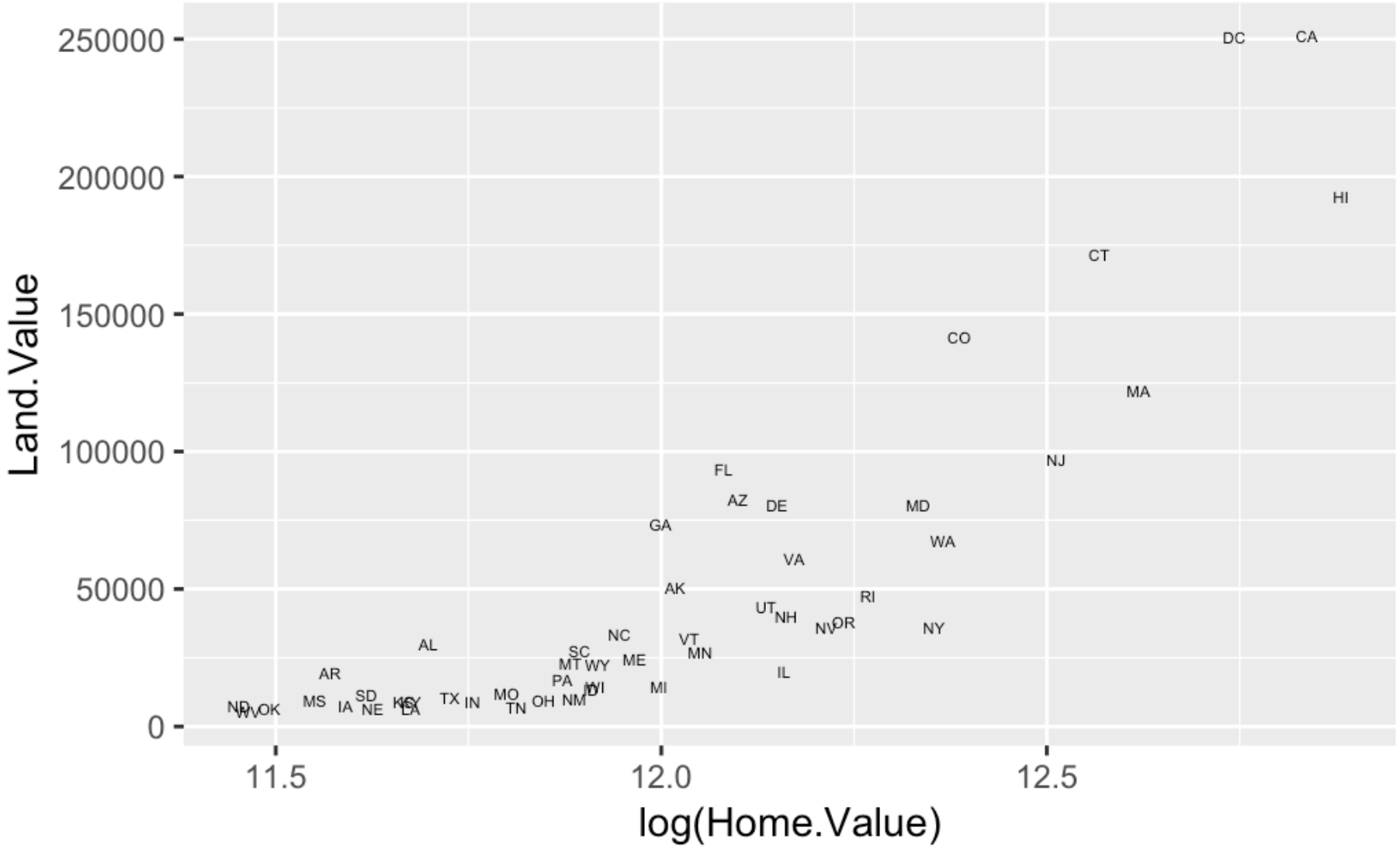


- El tamaño de los labels no nos ayuda mucho en este caso.

Hide

Hide

```
p1+geom_text(aes(label=State), size=1.5)
```



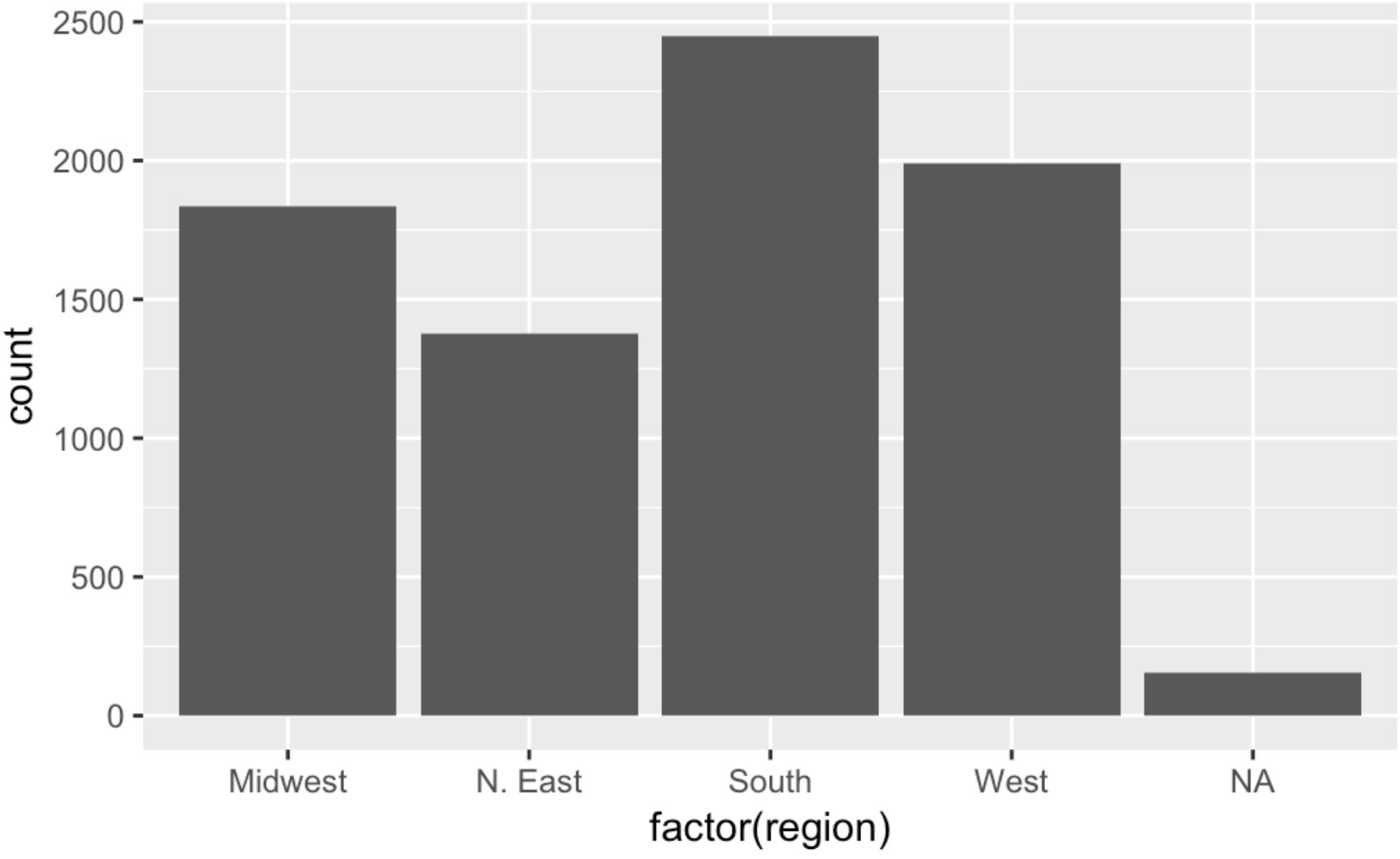
Barras

- Es fácil hacer gráficas de barras:

Hide

Hide

```
ggplot(housing, aes(factor(region)))+geom_bar()
```

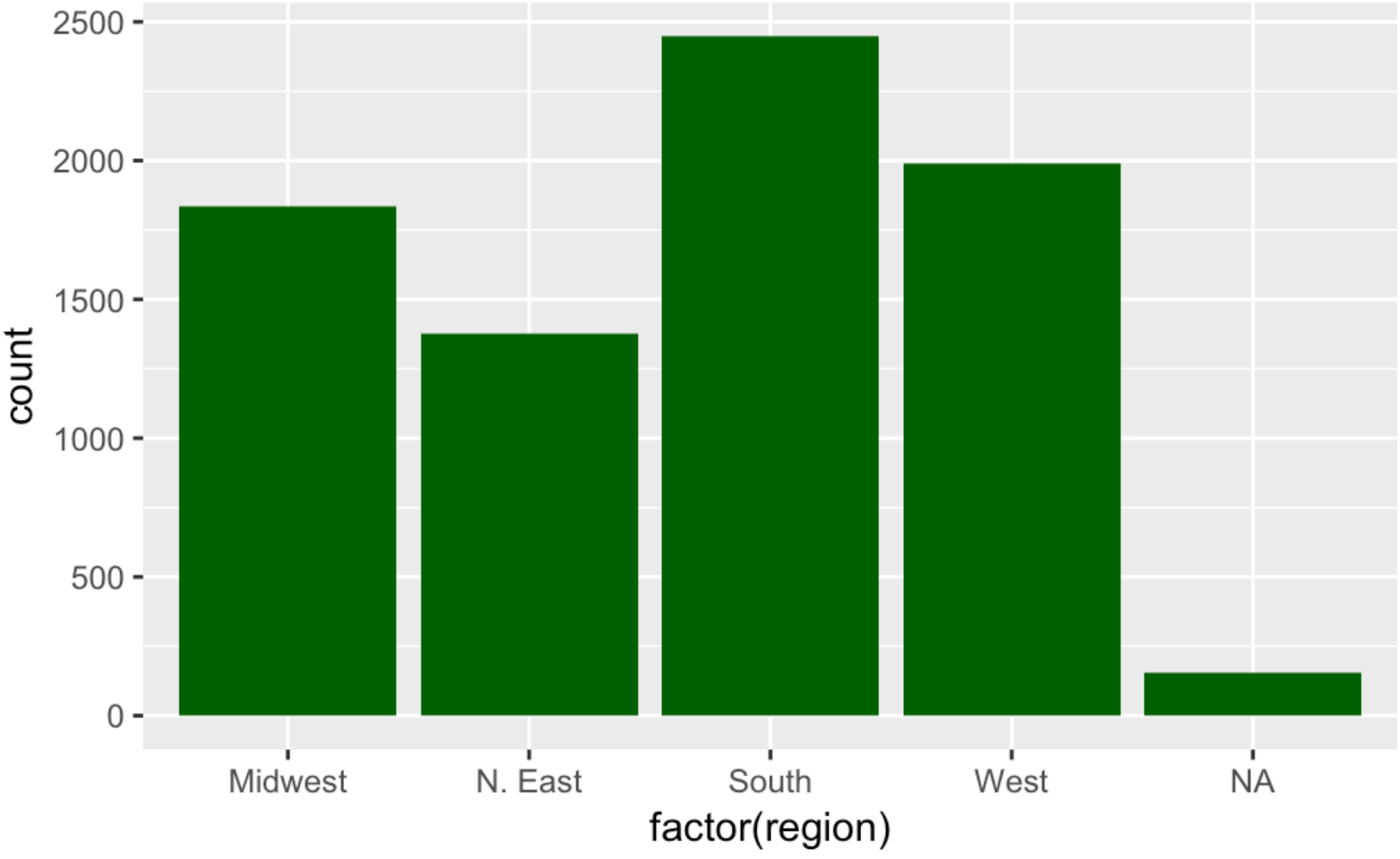


- Le podemos poner algo de color:

Hide

Hide

```
ggplot(housing, aes(factor(region)))+geom_bar(fill="darkgreen")
```

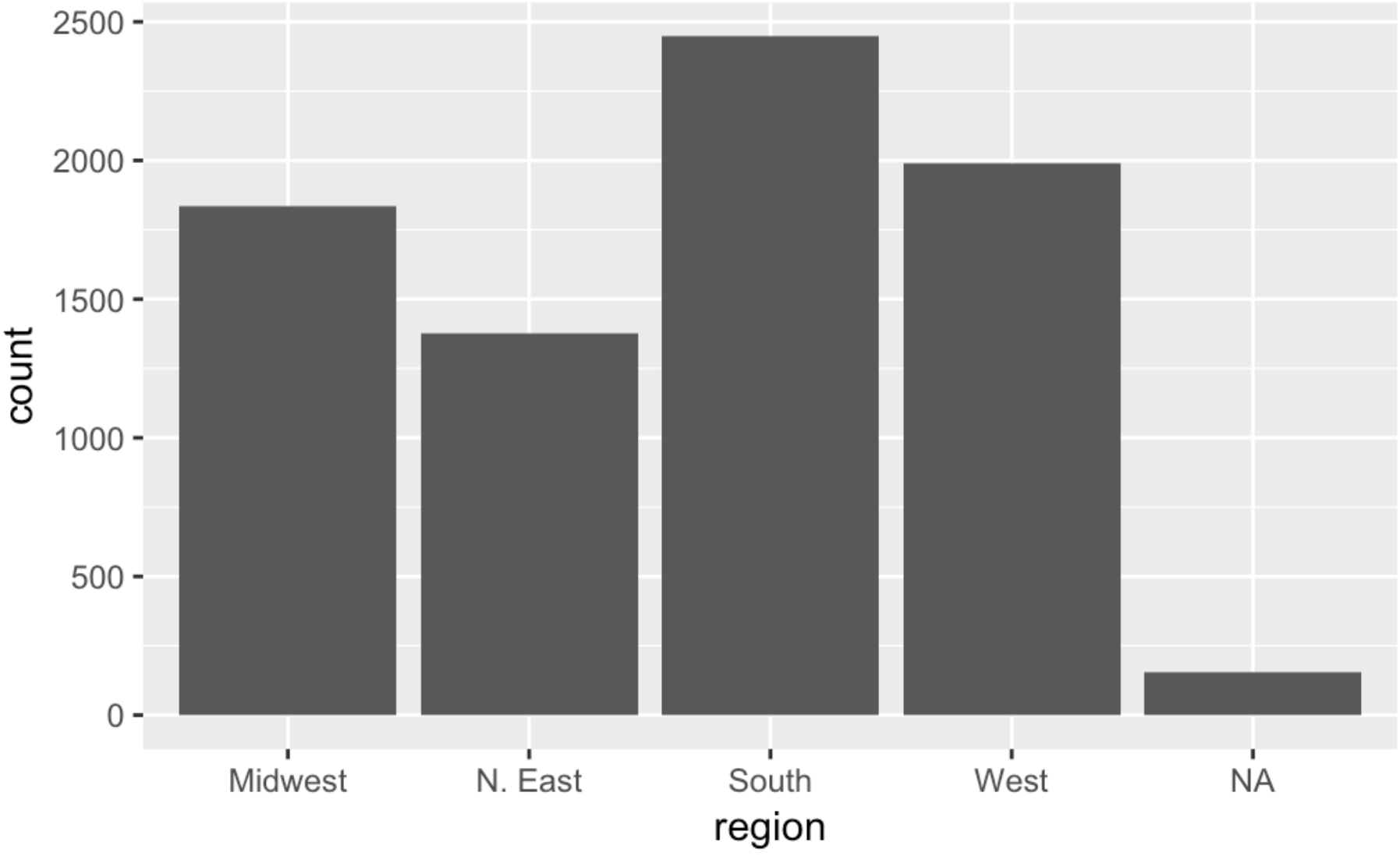


- Hay otra función, `qplot` , que es útil para este tipo de gráficas.
- La sintaxis es distinta:

Hide

Hide

```
qplot(region, data=housing, geom="bar")
```

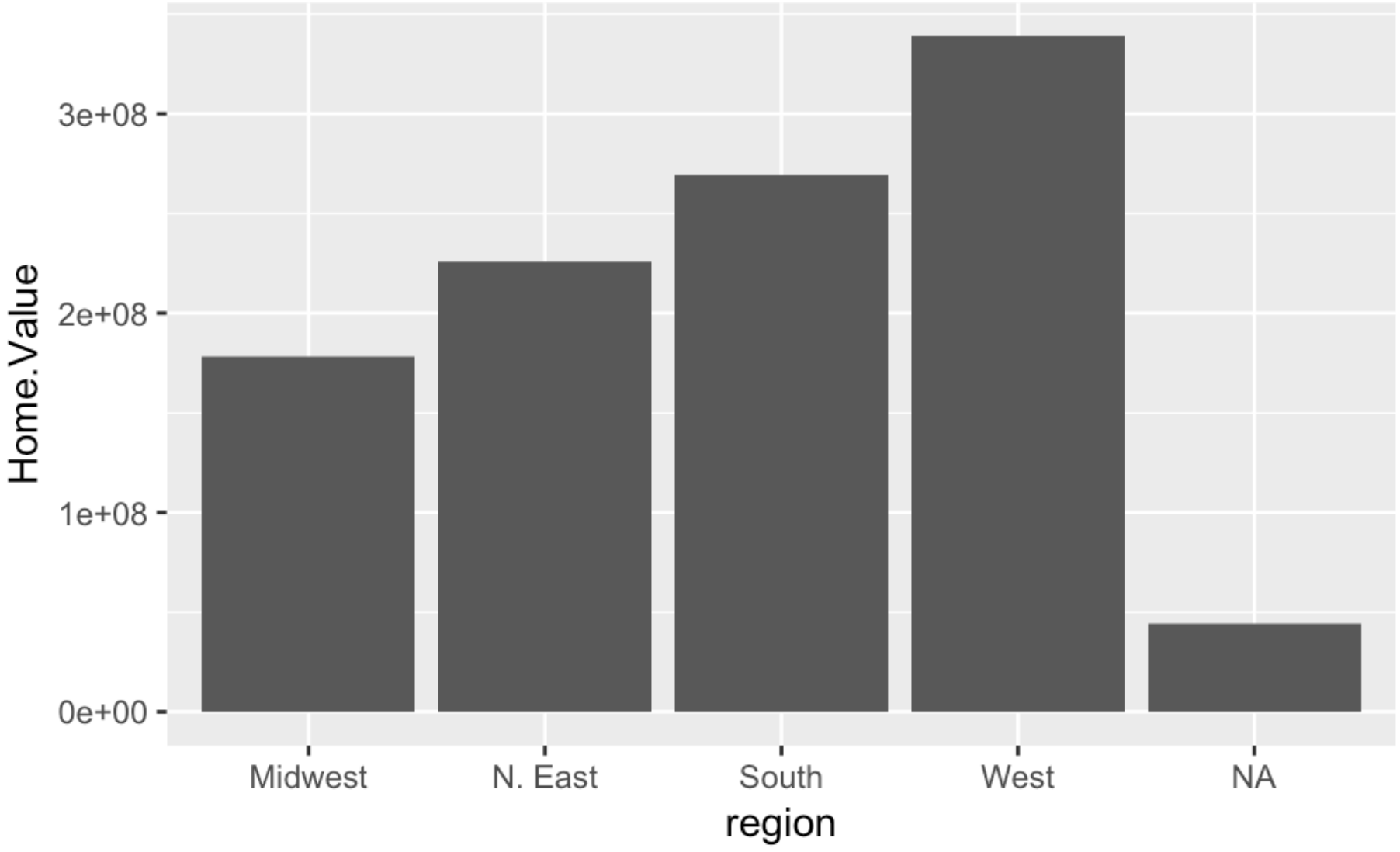


- Esta función es más intuitiva para ciertas variantes.
- Por ejemplo, si queremos ver la media de `Home.Value` por región.

Hide

Hide

```
qplot(region, data=housing, geom="bar", weight=Home.Value, ylab="Home.Value")
```

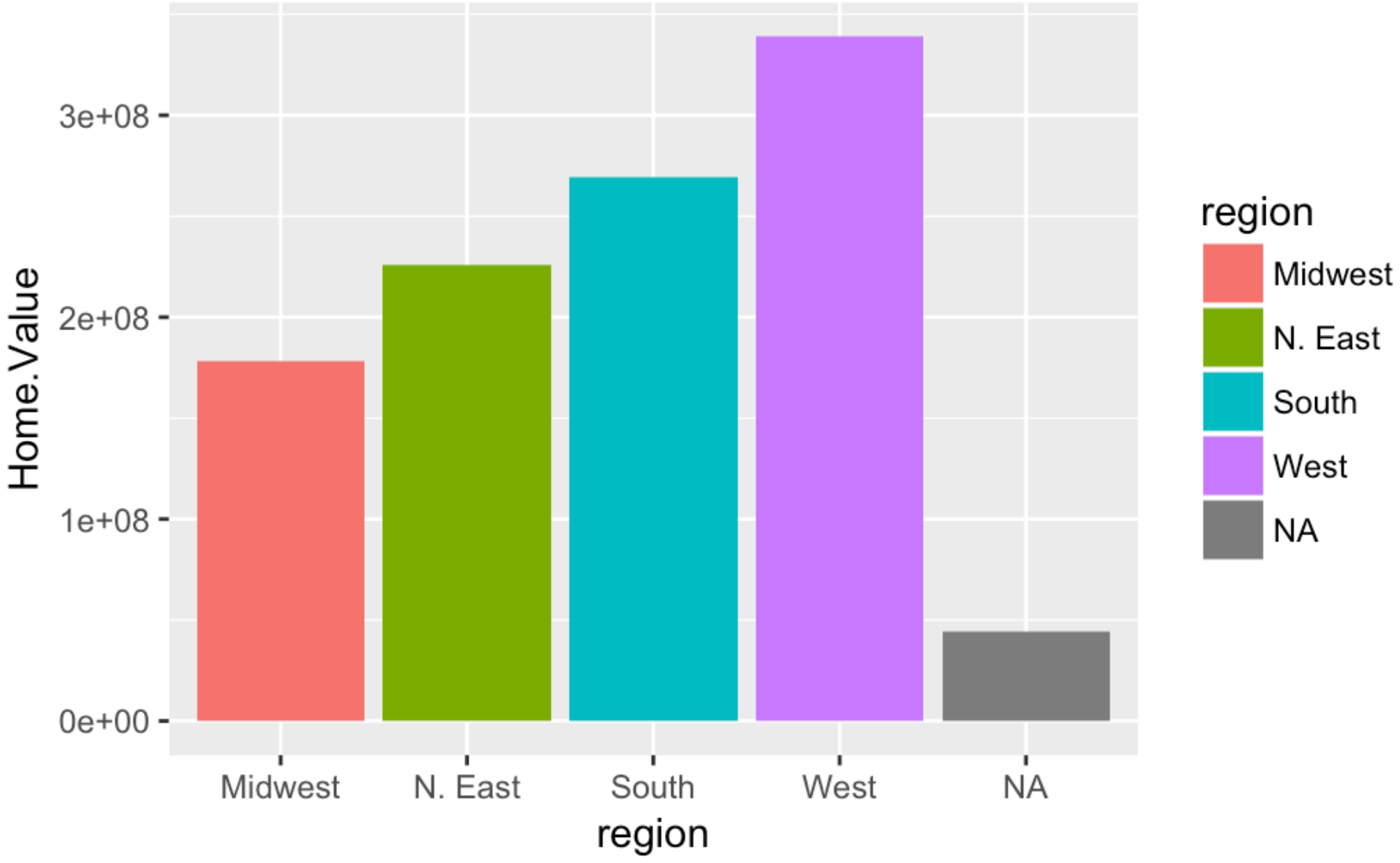


- Y le podemos poner un poco de color:

Hide

Hide

```
qplot(region, data=housing, geom="bar", weight=Home.Value, ylab="Home.Value", fill=re
gion)
```


3e+08
2e+08
1e+08
0e+00
Home.Value
Midwest
N. East
South
West
NA
region
region
Midwest
N. East
South
West
NA