Working with Geospatial Data in R

The `raster` package

Data frames aren't a great way to store spatial data

```
> head(preds)
        lon      lat predicted_price
1 -123.3168 44.52539        258936.2
2 -123.3168 44.52740        257258.4
3 -123.3168 44.52940        255543.1
4 -123.3168 44.53141        253791.0
5 -123.3168 44.53342        252002.4
6 -123.3168 44.53542        250178.7
```

- No CRS information
- Inefficient storage
- Inefficient display

A better structure for raster data

- data matrix + information on grid + CRS

258936.2	256579.2	254147.2	251593.8	…
257258.4	255082.5	252848.8	250499.2	
255543.1	253557.9	251537.5	249410.6	
253791.0	252004.4	250211.4	248326.8	
…				

The `raster` package

- `sp` provides some `raster` data classes:
  - `SpatialGrid`, `SpatialPixels`, `SpatialGridDataFrame`, `SpatialPixelsDataFrame`
- But `raster` is better:
  - easier import of rasters
  - large rasters aren't read into memory
  - provides functions for raster type operations
- Also uses S4 and when appropriate provides same functions

`raster` provides print methods for `sp` objects

```
> library(sp)
> countries_spdf

An object of class "SpatialPolygonsDataFrame"
Slot "data":
                      name iso_a3 population          gdp
region
1              Afghanistan    AFG   28400000     22270.00
2                   Angola    AGO   12799293    110300.00
3                  Albania    ALB    3639453     21810.00

...
```

VERY long output!

```
Slot "proj4string":
CRS arguments:
 +proj=longlat +datum=WGS84 +no_defs +ellps=WGS84 +towgs84=0,0,0
```

raster provides print methods for sp objects

```
> library(raster)
> countries_spdf

class       : SpatialPolygonsDataFrame
features    : 177
extent      : -180, 180, -90, 83.64513  (xmin, xmax, ymin, ymax)
coord. ref. : +proj=longlat +datum=WGS84 +no_defs +ellps=WGS84 …
variables   : 6
names       :        name, iso_a3, population,         gdp,
min values  : Afghanistan,    -99,        140,       16.00,
max values  :    Zimbabwe,    ZWE, 1338612970, 15094000.00, …
```

Compact and useful output

Let’s practice!

Color

A perceptual color space: HCL

- **Trichromatic** - we perceive color as three-dimensional

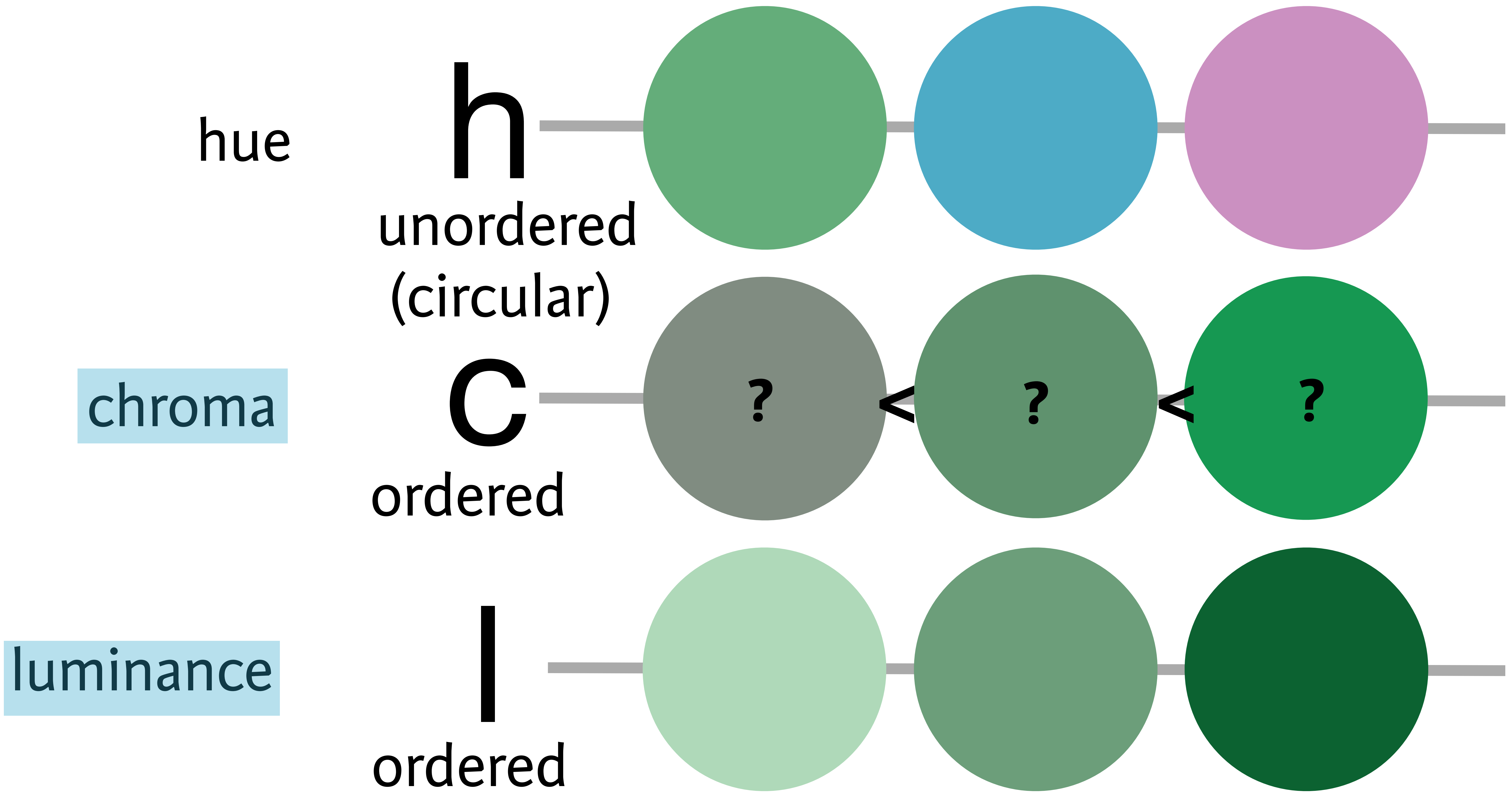


Image credit: Hadley Wickham

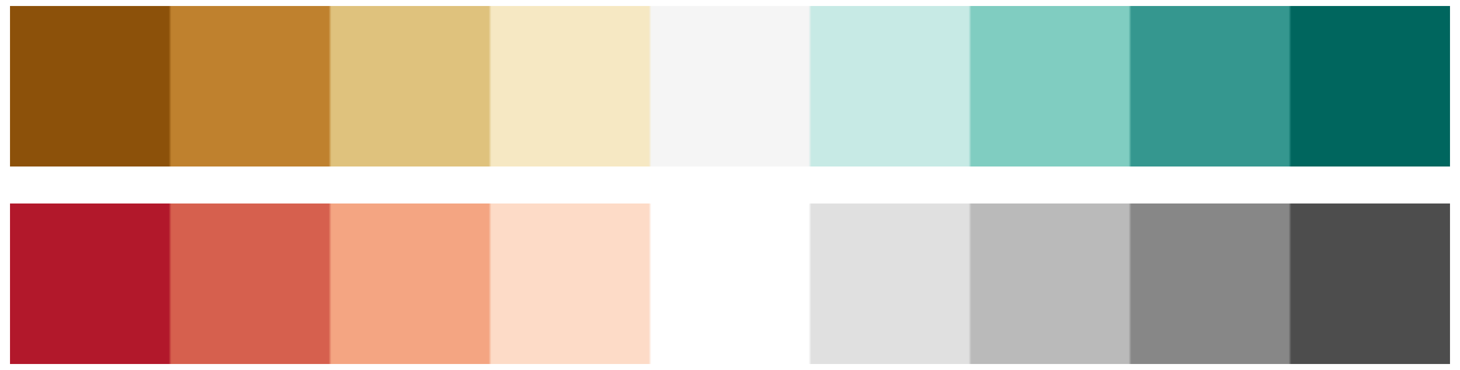
Types of scale

- Sequential - ordered

  steps in chroma and/or luminance
  hue maybe redundant coding

- Diverging - ordered but in two directions

  steps in chroma and/or luminance
  with hue distinguishing direction



- Qualitative - unordered

  steps in hue with
  equal chroma and luminance

Generating color scales in R

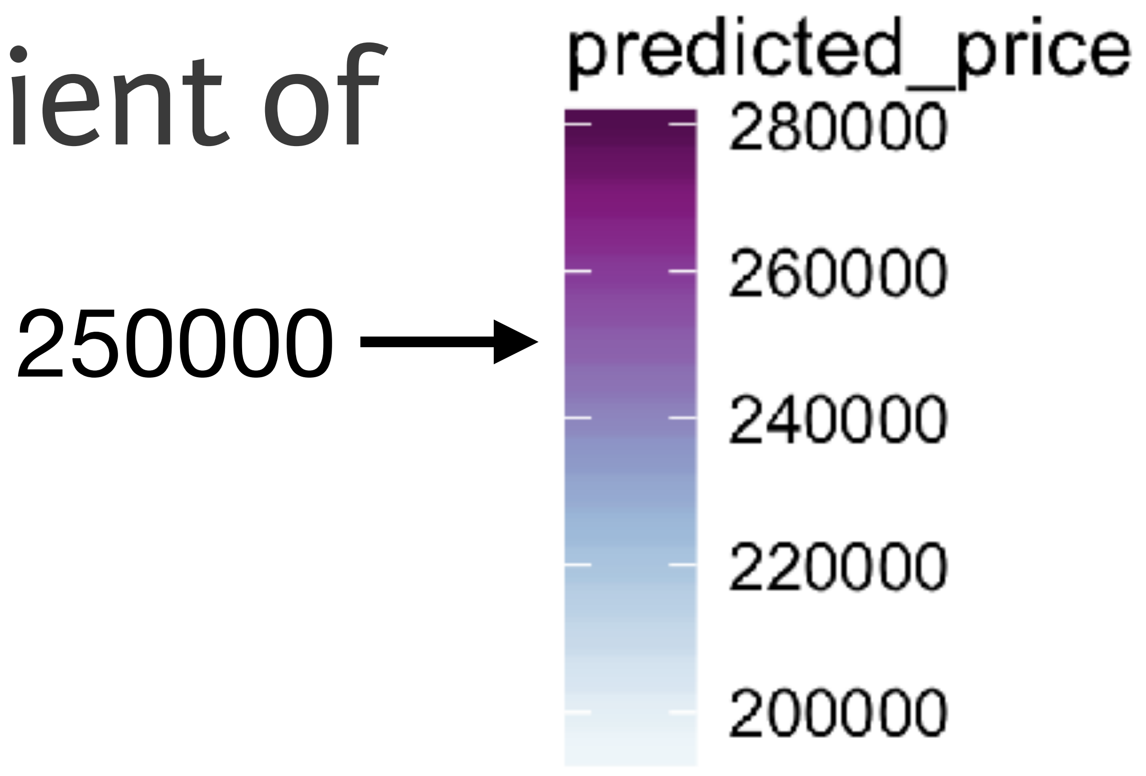
```
> library(RColorBrewer)
> display.brewer.all()

> brewer.pal(n = 9, "Blues")
[1] "#F7FBFF" "#DEEBF7" "#C6DBEF" "#9ECAE1"
[5] "#6BAED6" "#4292C6" "#2171B5" "#08519C"
[9] "#08306B"

> library(viridisLite)
> viridis(n = 9)
[1] "#440154FF" "#472D7BFF" "#3B528BFF" "#2C728EFF"    transparency
[5] "#21908CFF" "#27AD81FF" "#5DC863FF" "#AADC32FF"
[9] "#FDE725FF"
```

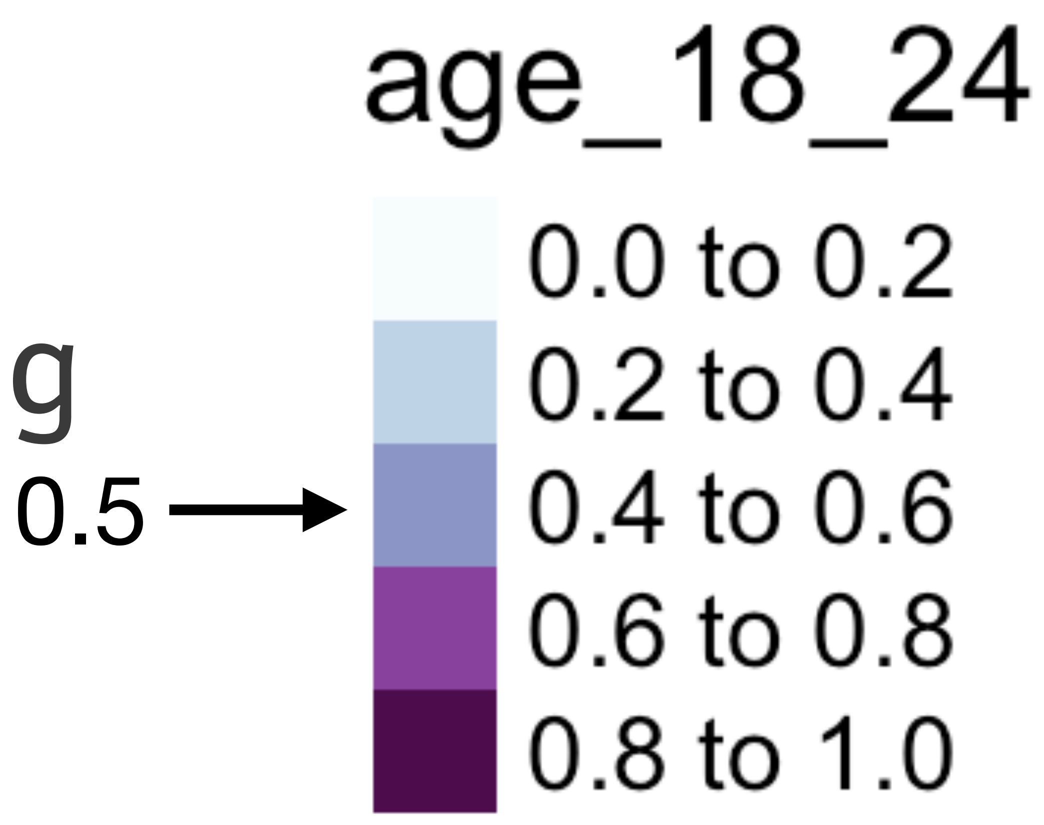
Let's practice!

Color scales 2

Mapping of numbers to color

- `ggplot2`: map to a continuous gradient of color
- `tmap`: map to a discrete set of colors
- Continuous map: control mapping by transforming the scale, e.g log
- Discrete map: control mapping by binning the variable



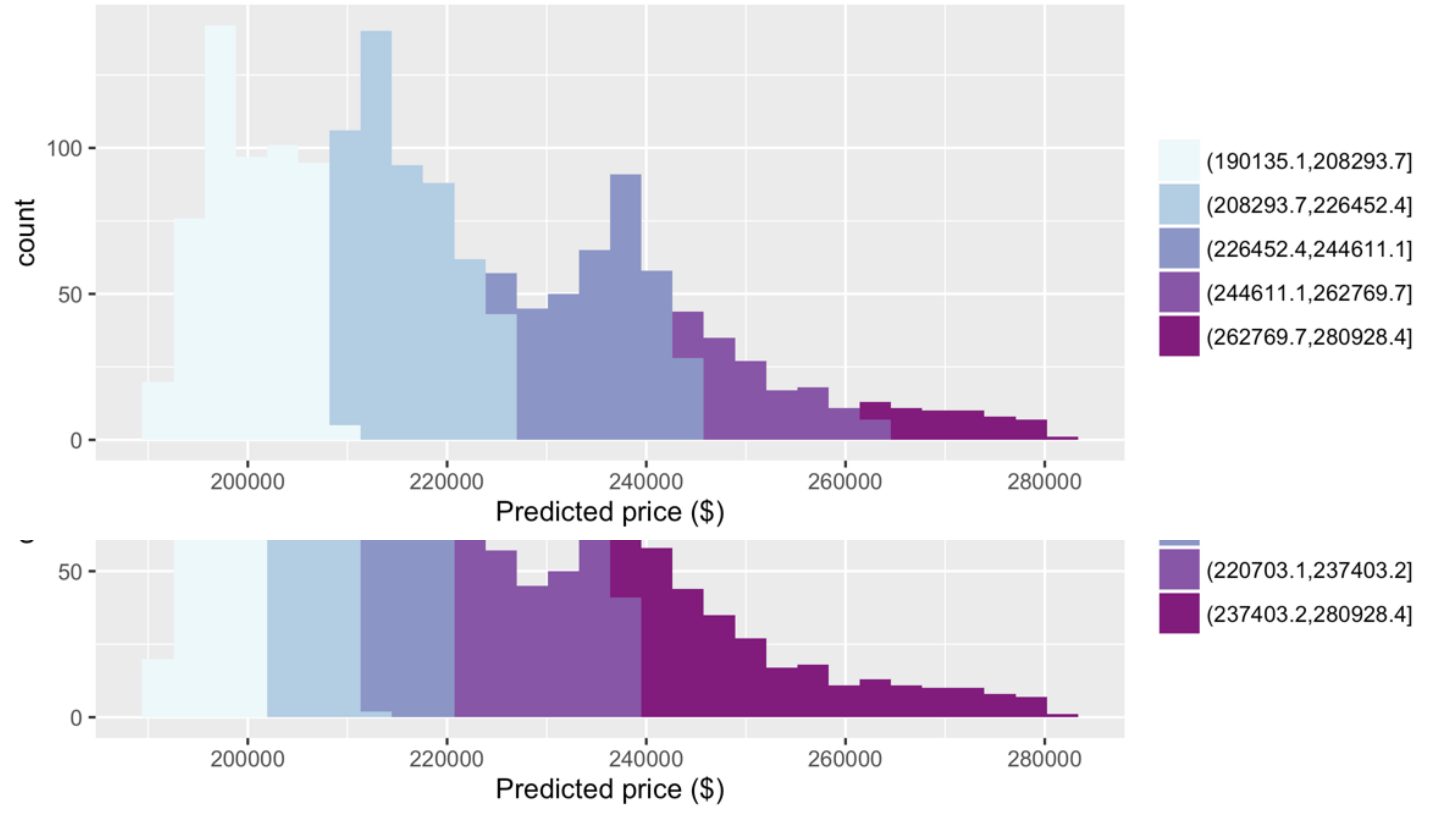


Discrete vs. continuous mapping

- Continuous:
  - Perceptually uniform: perceiving equivalent color difference to numerical difference
- Discrete:
  - Complete control over scale
  - Easier lookup

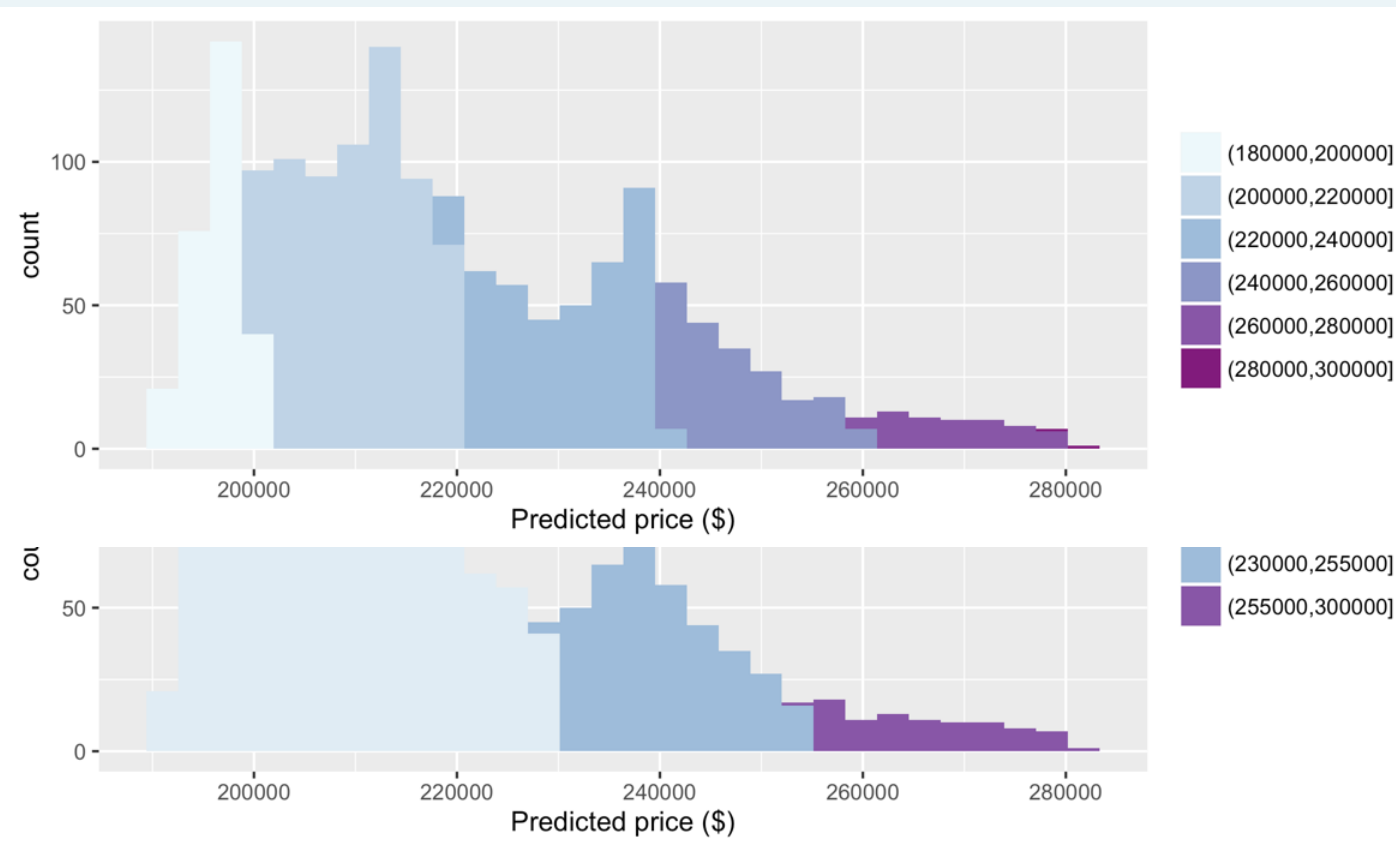
Cutting a variable into bins

```
> lib
> cla

style
[1901

[2446


> cla

style
[1901

[2207
```



Cutting a variable into bins

```
> class

style:
 [18000

[260000


> class
    fix

style:
 [1e+05
```



Let's practice!