TRUMAN STATE UNIVERSITY

Department of STATISTICS

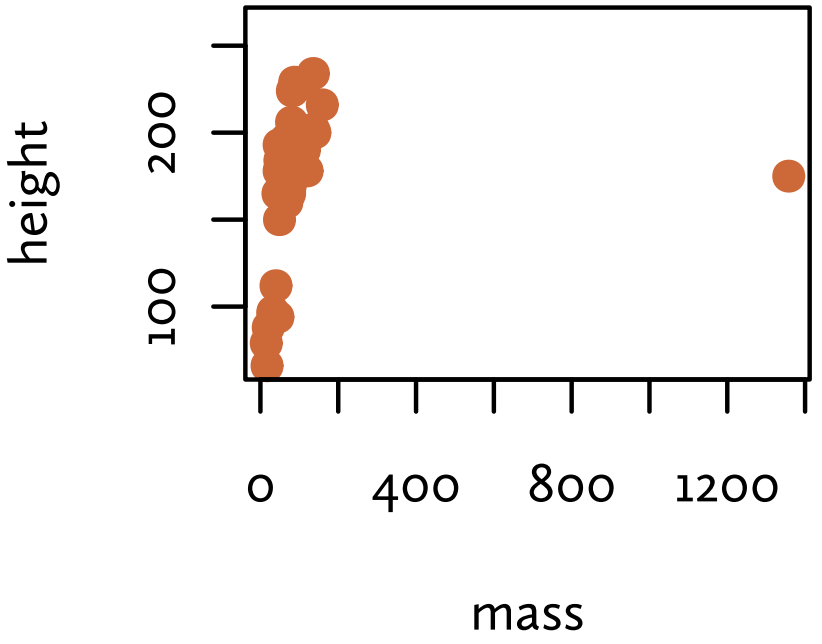
Colors in R

DISTINCT BY DESIGN

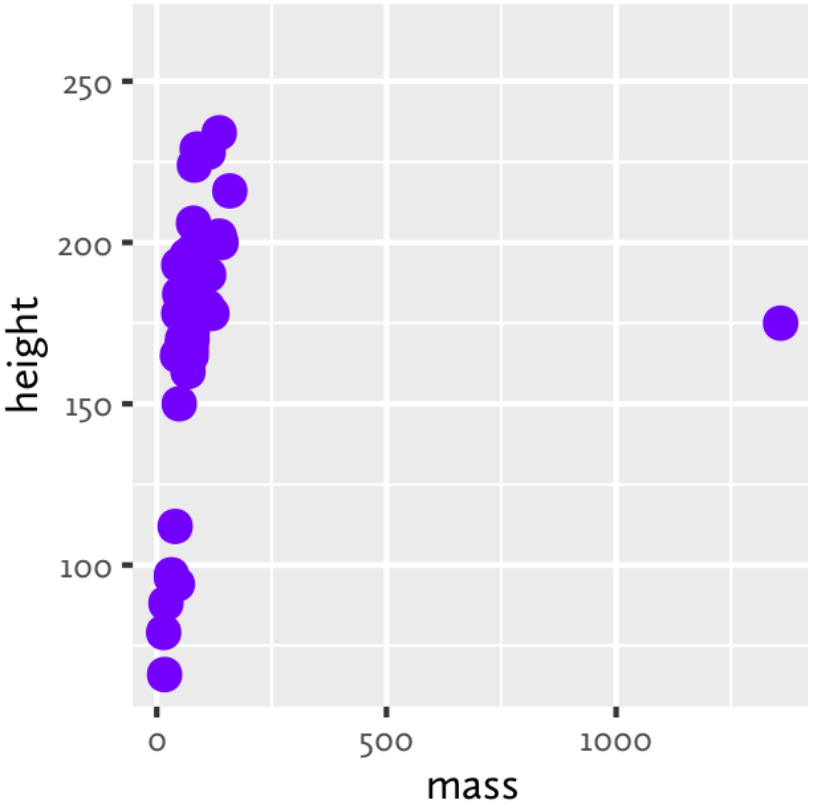
Specifying a Single Color

Single colors can be specified by one-word color names or RGB values in hexadecimal.

```
with(starwars,
  plot(mass, height,
  col="sienna3", pch=19))
```

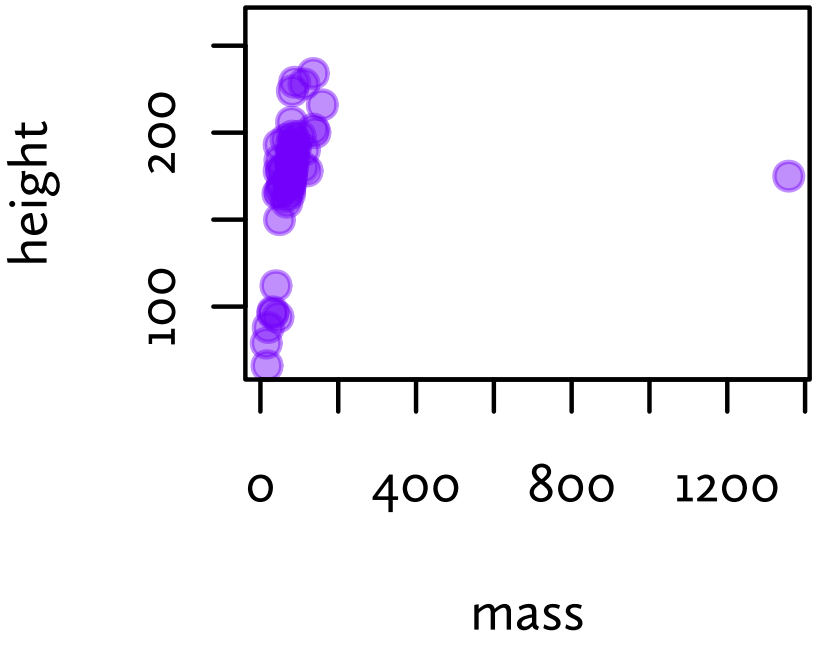



```
ggplot(starwars) +
geom_point(aes(x=mass, y=height),
  color="#7200FA", size=3)
```

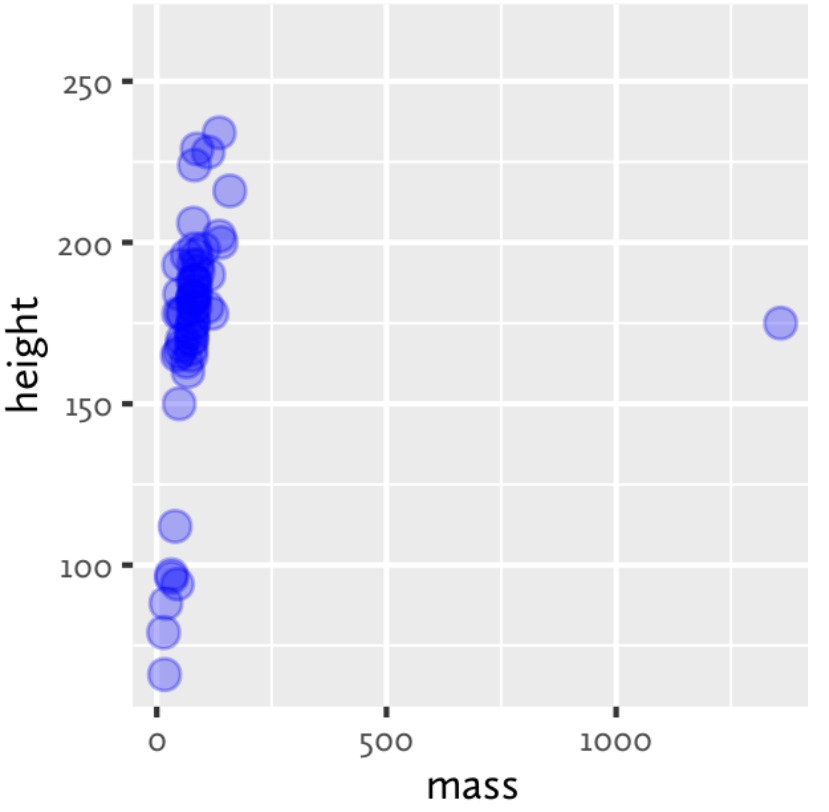


A fourth hex number specifies transparency. `ggplot` has an `alpha` aesthetic.

```
with(starwars,
  plot(mass, height,
  col="#7200FA70", pch=19))
```




```
ggplot(starwars) +
geom_point(aes(x=mass, y=height),
  color="blue", size=3, alpha=0.3)
```



Mid-Level: Other commands create color codes.

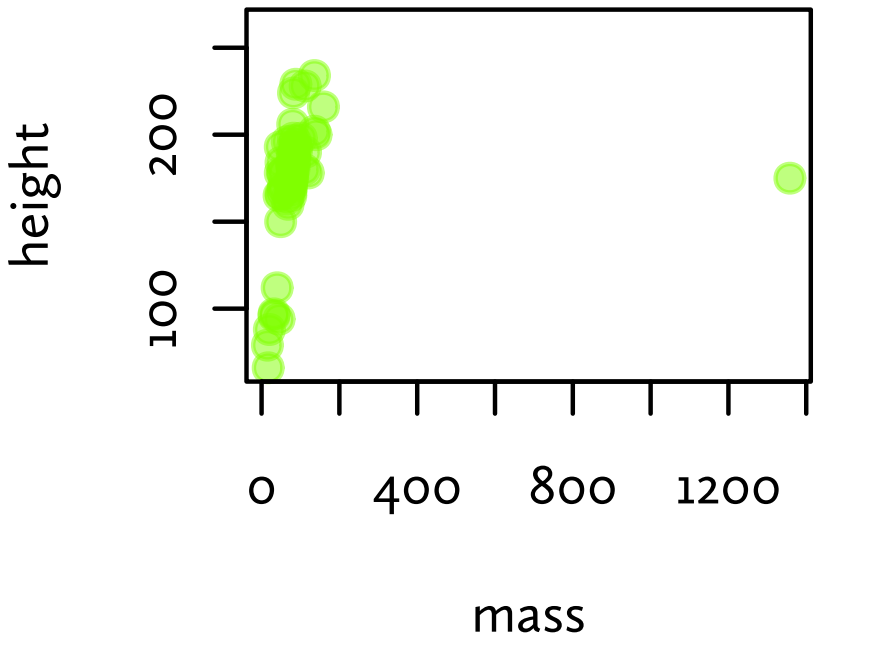
```
# See also `hsv` and `hcl`.
rgb(1, 0, 1)
```

```
[1] "#FF00FF"
```

```
grey(0.3)
```

```
[1] "#4D4D4D"
```

```
with(starwars,
  plot(mass, height,
  col=rgb(0.5, 1, 0, 0.5), pch=19))
```

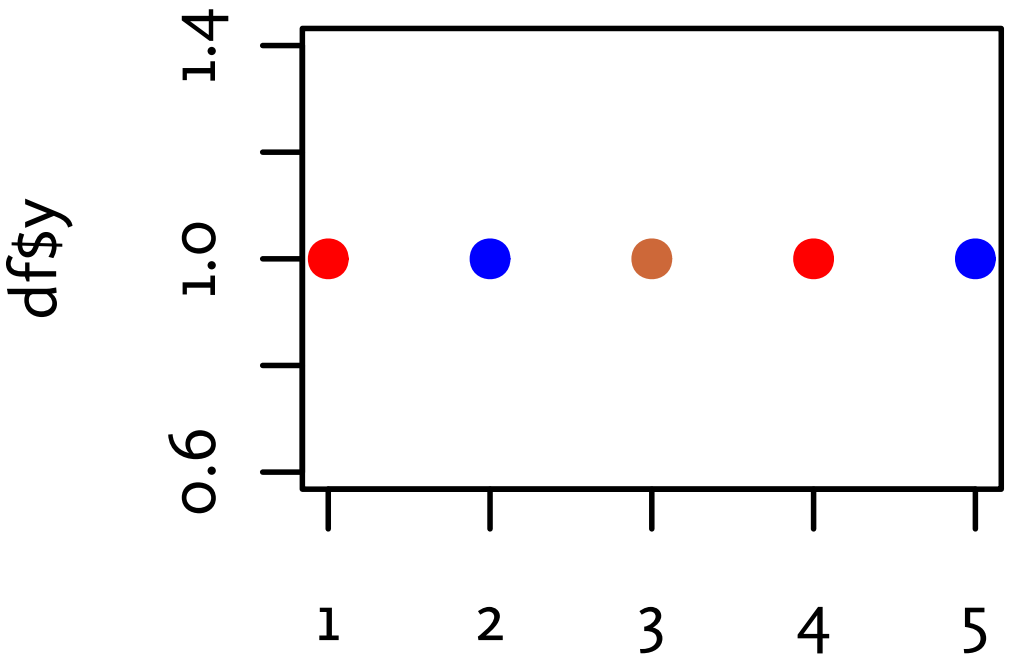


Vectors of Colors and Palettes

Color vectors can be directly specified.

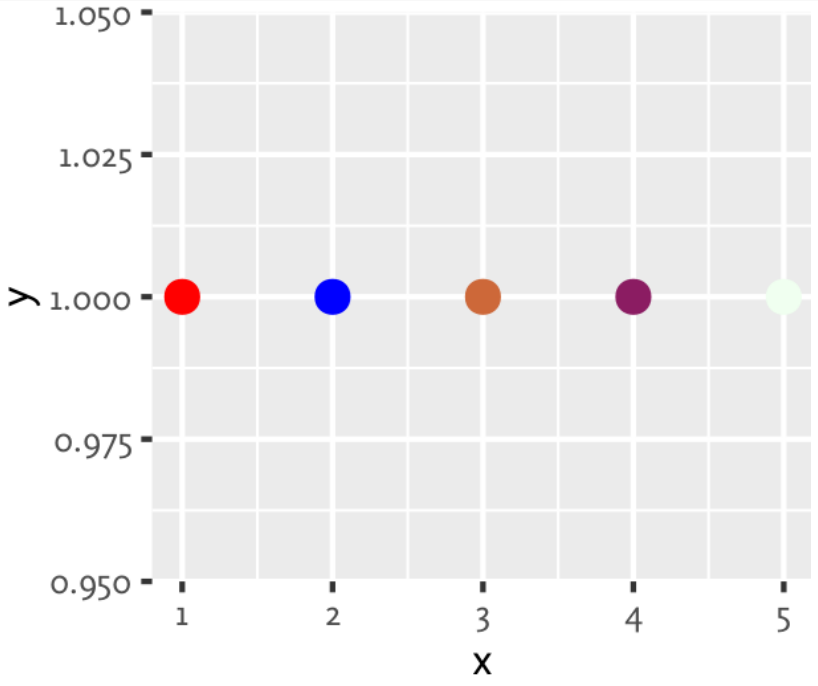
Used with base R graphics. The next graphical element gets the next color. The list starts over if you get to the end.

```
df <- data.frame(x=1:5, y=1)
col.vec = c("red", "blue", "sienna3")
plot(df$x, df$y, col=col.vec, pch=19)
```

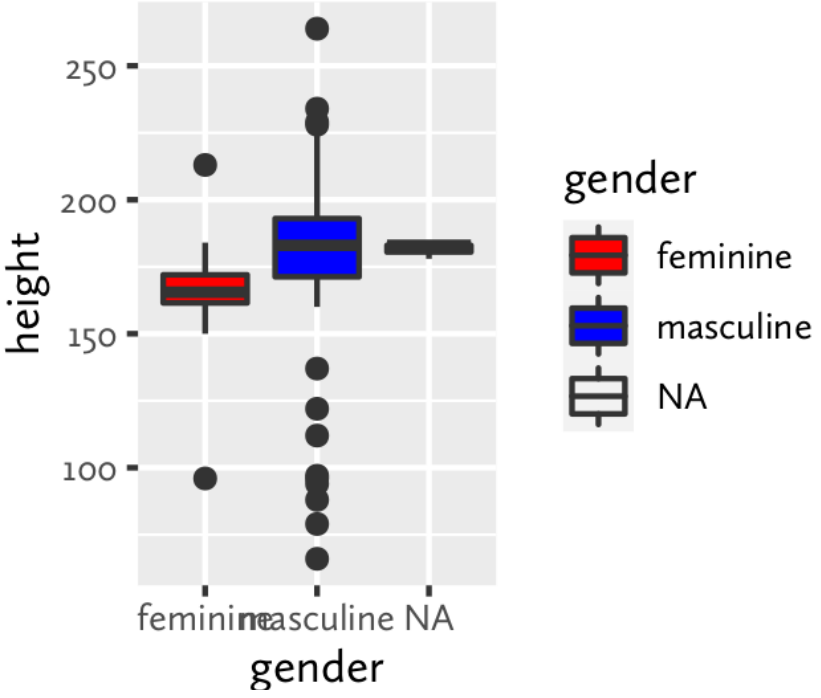


ggplot requires the list to be a single color or at least as long as the data set.

```
col.vec2 = c("red", "blue",
  "sienna3", "maroon4", "honeydew")
ggplot(df) +
  geom_point(aes(x=x, y=y),
    color=col.vec2, size=3)
```




```
ggplot(starwars) +
  geom_boxplot(aes(fill=gender,
    x=gender, y=height)) +
  scale_fill_manual(values=col.vec2)
```



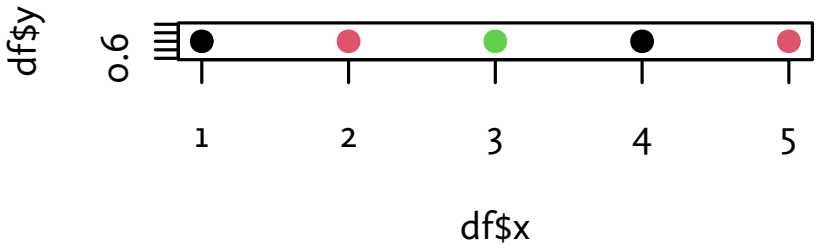
Colors specified by integers look up colors in *palettes*.

- If you specify an integer value for color, R will look up a corresponding color from its default *palette*.
- The `palette` command will also set a palette for base R graphics if given a vector of colors.
- Factors are secretly integers—that's how factors are mapped to colors.
- `ggplot` works a little differently (see below).

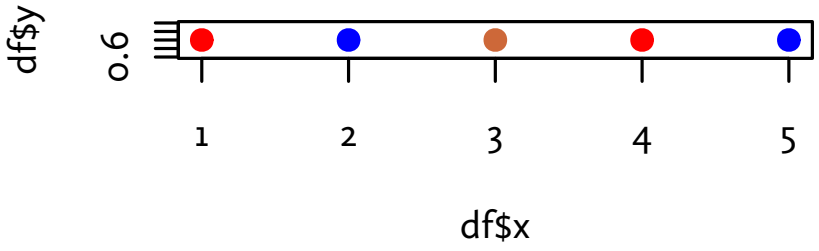
```
palette()
```

```
[1] "black"   "#DF536B" "#61D04F" "#22
[8] "gray62"
```

```
plot(df$x, df$y, col=1:3, pch=19)
```




```
palette(col.vec)
plot(df$x, df$y, col=1:3, pch=19)
```




```
palette("default")
```

Palette Commands automate the process of making vectors of colors.

- Generally, they have the form `palette_command(n, ...)` where `n` is the number of colors you want, and `...` may specify some other options.
- Syntax may vary by package.

```
# Built-in Palettes
rainbow(3)
```

```
[1] "#FF0000" "#00FF00" "#0000FF"
```

```
rainbow(5)
```

```
[1] "#FF0000" "#CCFF00" "#00FF66" "#0066FF" "#CC00FF"
```

```
heat.colors(4)
```

```
[1] "#FF0000" "#FF8000" "#FFFF00" "#FFFF80"
```

```
terrain.colors(5)
```

```
[1] "#00A600" "#E6E600" "#EAB64E" "#EEB99F" "#F2F2F2"
```

```
topo.colors(5)
```

```
[1] "#4C00FF" "#004CFF" "#00E5FF" "#00FF4D" "#FFFF00"
```

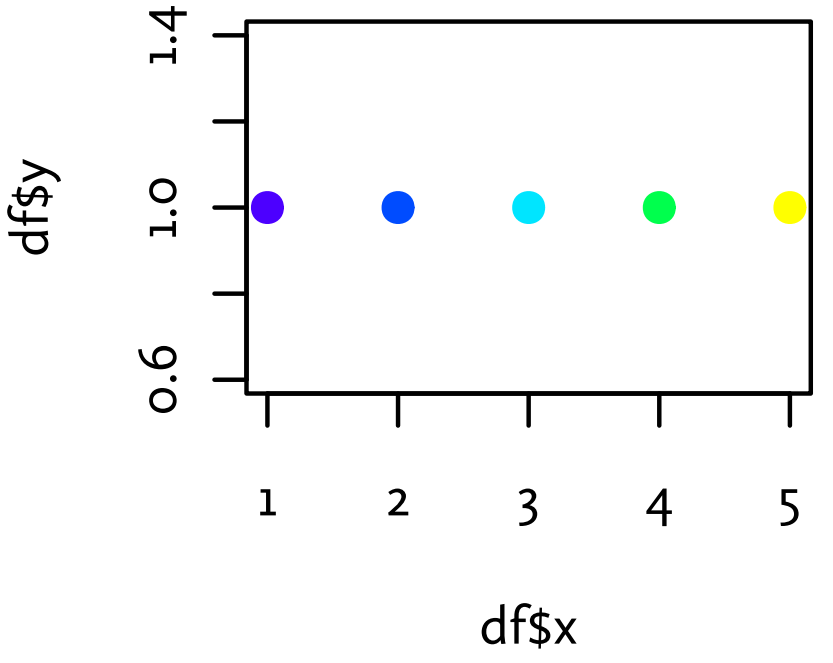
```
cm.colors(5)
```

```
[1] "#80FFFF" "#BFFFFF" "#FFFFFF" "#FFBFFF" "#FF80FF"
```

Palette Commands automate the process of making vectors of colors.

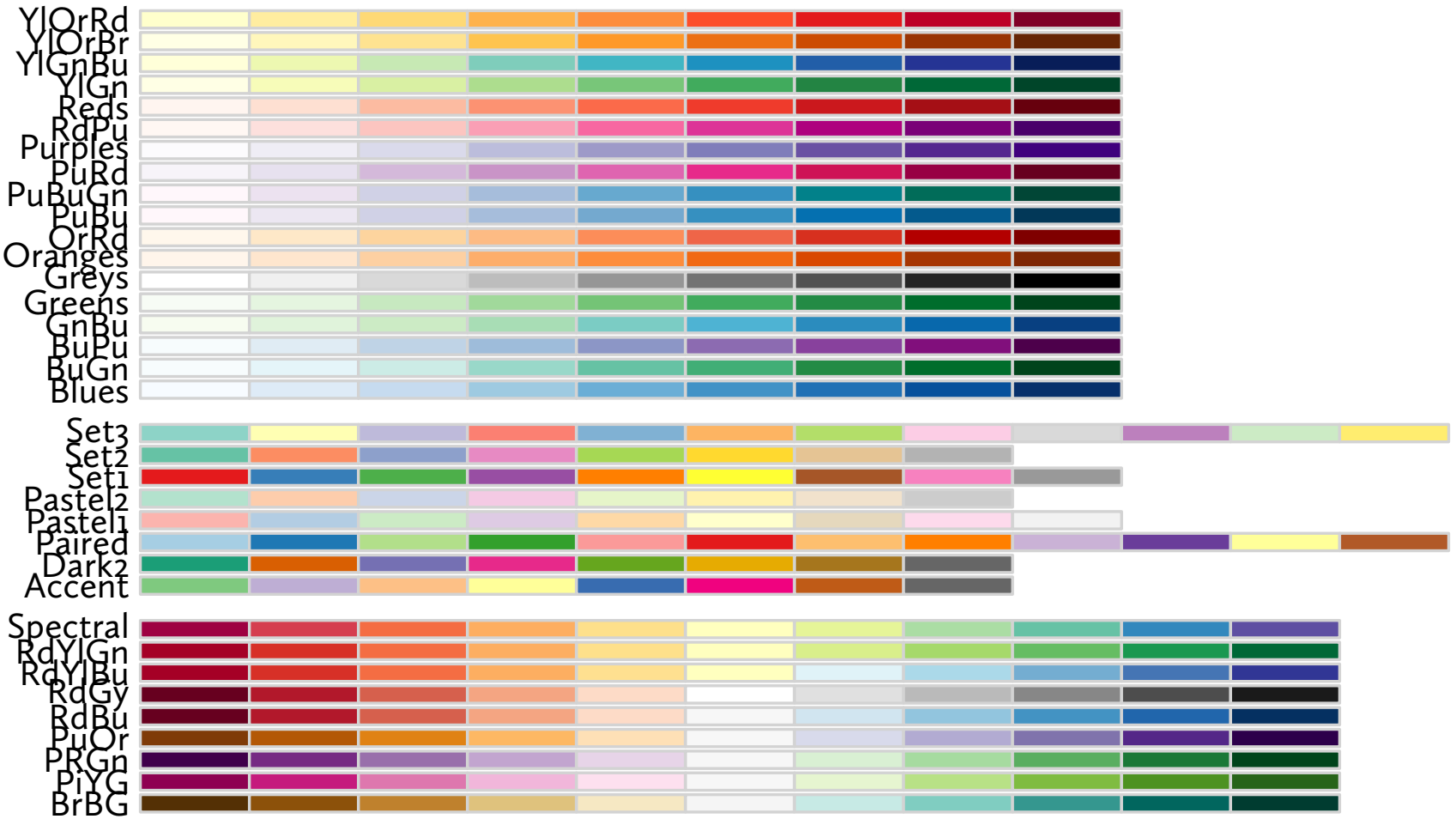
- Generally, they have the form `palette_command(n, ...)` where n is the number of colors you want, and `...` may specify some other options.
- Syntax may vary by package.

```
# Built-in Palettes
plot(df$x, df$y,
  col=topo.colors(5), pch=19)
```



Color Brewer is a commonly-used color palette package, featuring gradient, categorical and diverging palettes.

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



Others

See https://github.com/EmilHvitfeldt/r-color-palettes for many options.

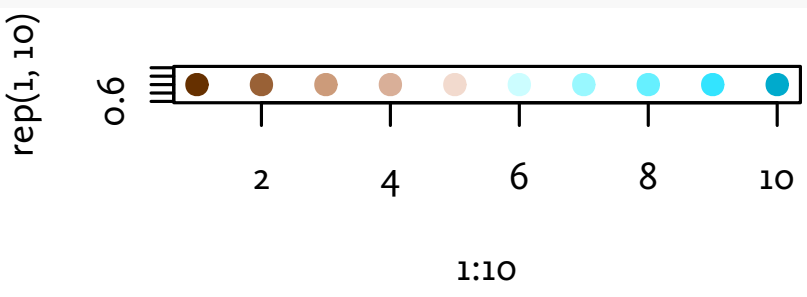
Warning: Various packages have various calling syntax. Some give lists of colors, some give functions that take n as shown above.

- viridis
- dichromat
- ghibli
- colorspace
- ggsci
- ggthemes

```
library(dichromat)
colorschemes$BrowntoBlue.10
```

```
 [1] "#663000" "#996136" "#CC9B7A" "#D
 [8] "#66F0FF" "#33E4FF" "#00AACC"
```

```
plot(x=1:10, y=rep(1, 10), pch=19,
  col=colorschemes$BrowntoBlue.10)
```



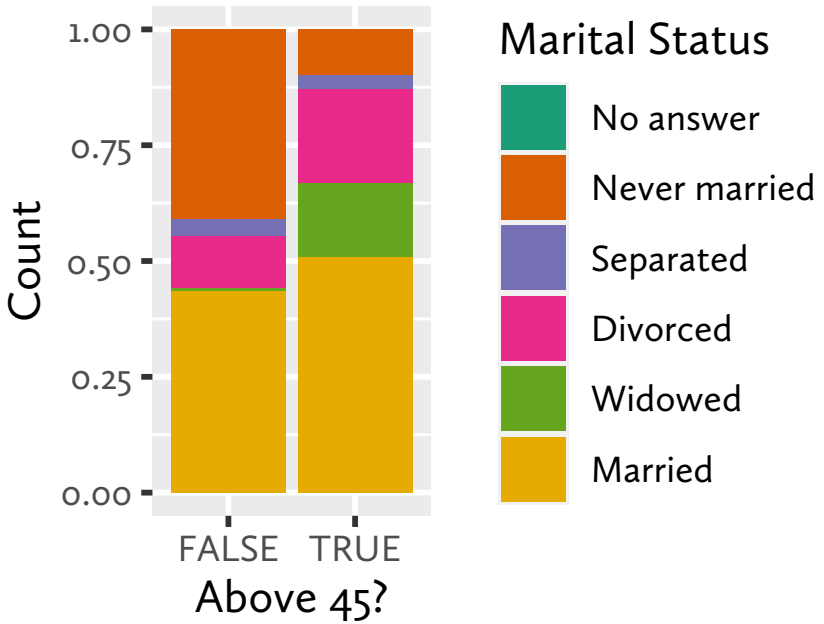
Difference between Base R and `ggplot`

- At the end of the day, Base R graphics tend to want a vector of colors.
  - How you get that vector of colors is up to you (thus the many formats from various packages).
  - Colors will be repeated, if necessary.
- On the other hand, `ggplot` has several methods:
  - it wants a single color (as `color=...`) or
  - a color for every data point if you're manually specifying colors (either `color=...` or one of the `scale_..._manual` commands).
  - `ggplot` also has many pre-built scale commands, as detailed next.

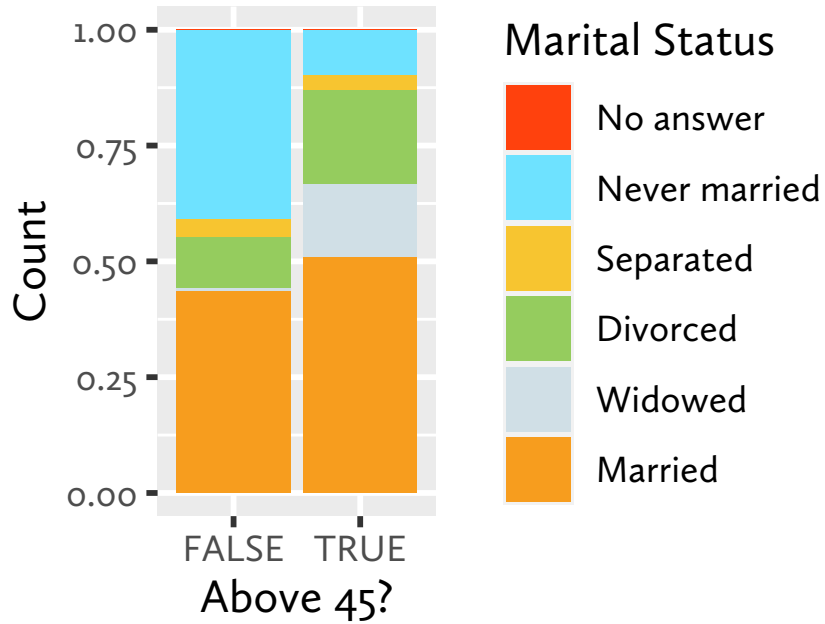
Built-in ggplot Color Scales

In ggplot, look for scale_fill... or scale_color... commands. They tend to count the n for you.

```r
# ggplot builds in color brewer palettes
gss_cat %>% select(age, marital) %>%
  na.omit() %>%
  mutate(above45 = age > 45) %>%
  ggplot(aes(x=above45, fill=marital)) +
  geom_bar(position="fill") +
  labs(y="Count", x="Above 45?",
    fill="Marital Status") +
  scale_fill_brewer(palette="Dark2")
```

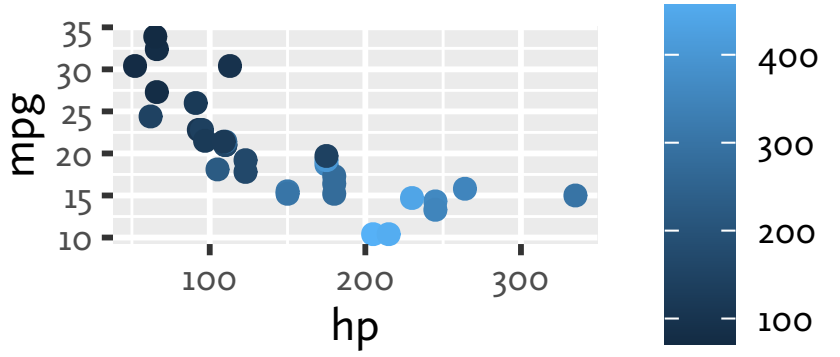



```r
library(ggsci)
gss_cat %>% select(age, marital) %>%
  na.omit() %>%
  mutate(above45 = age > 45) %>%
  ggplot(aes(x=above45, fill=marital)) +
  geom_bar(position="fill") +
  labs(y="Count", x="Above 45?",
    fill="Marital Status") +
  scale_fill_tron()
```

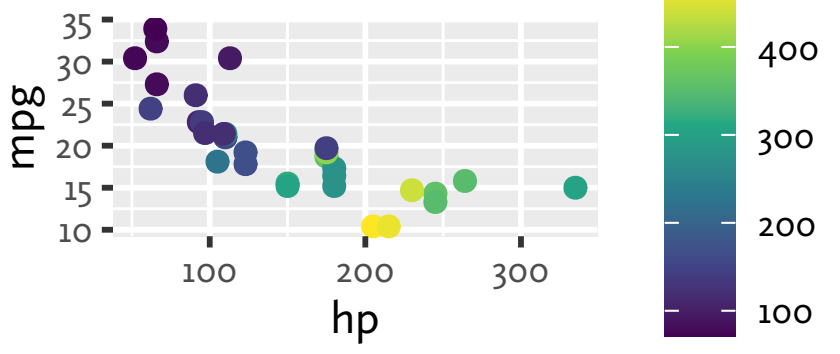


ggplot will allow you to set a **continuous** color map, but base R doesn't do that by default.

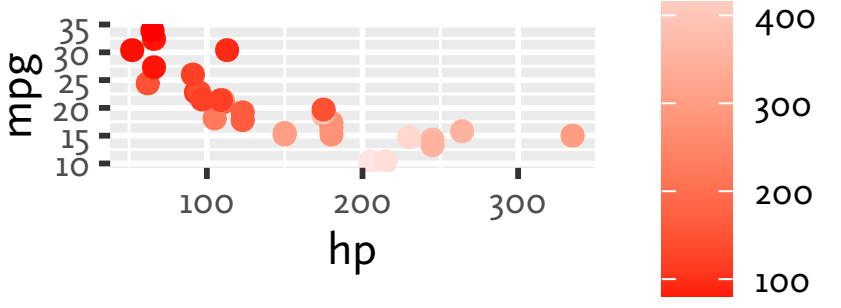
```
ggplot(mtcars, aes(x=hp, y=mpg)) +
  geom_point(aes(color=disp))
```



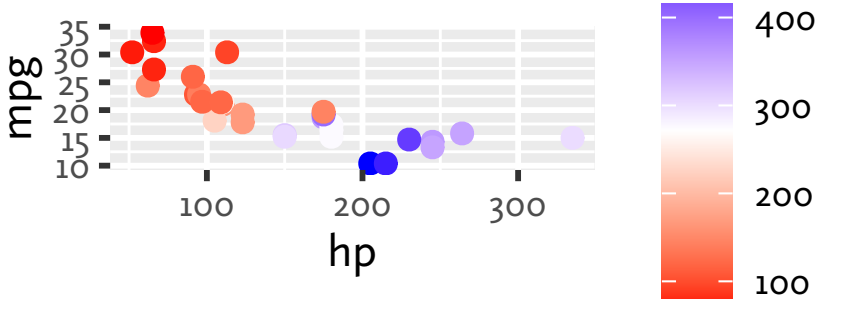

```
ggplot(mtcars, aes(x=hp, y=mpg)) +
  geom_point(aes(color=disp)) +
  scale_color_viridis_c()
```




```
ggplot(mtcars, aes(x=hp, y=mpg)) +
  geom_point(aes(color=disp)) +
  scale_color_gradient(
    low="red", high="mistyrose")
```




```
ggplot(mtcars, aes(x=hp, y=mpg)) +
  geom_point(aes(color=disp)) +
  scale_color_gradientn(
    colors=c("red", "white", "blue"))
```

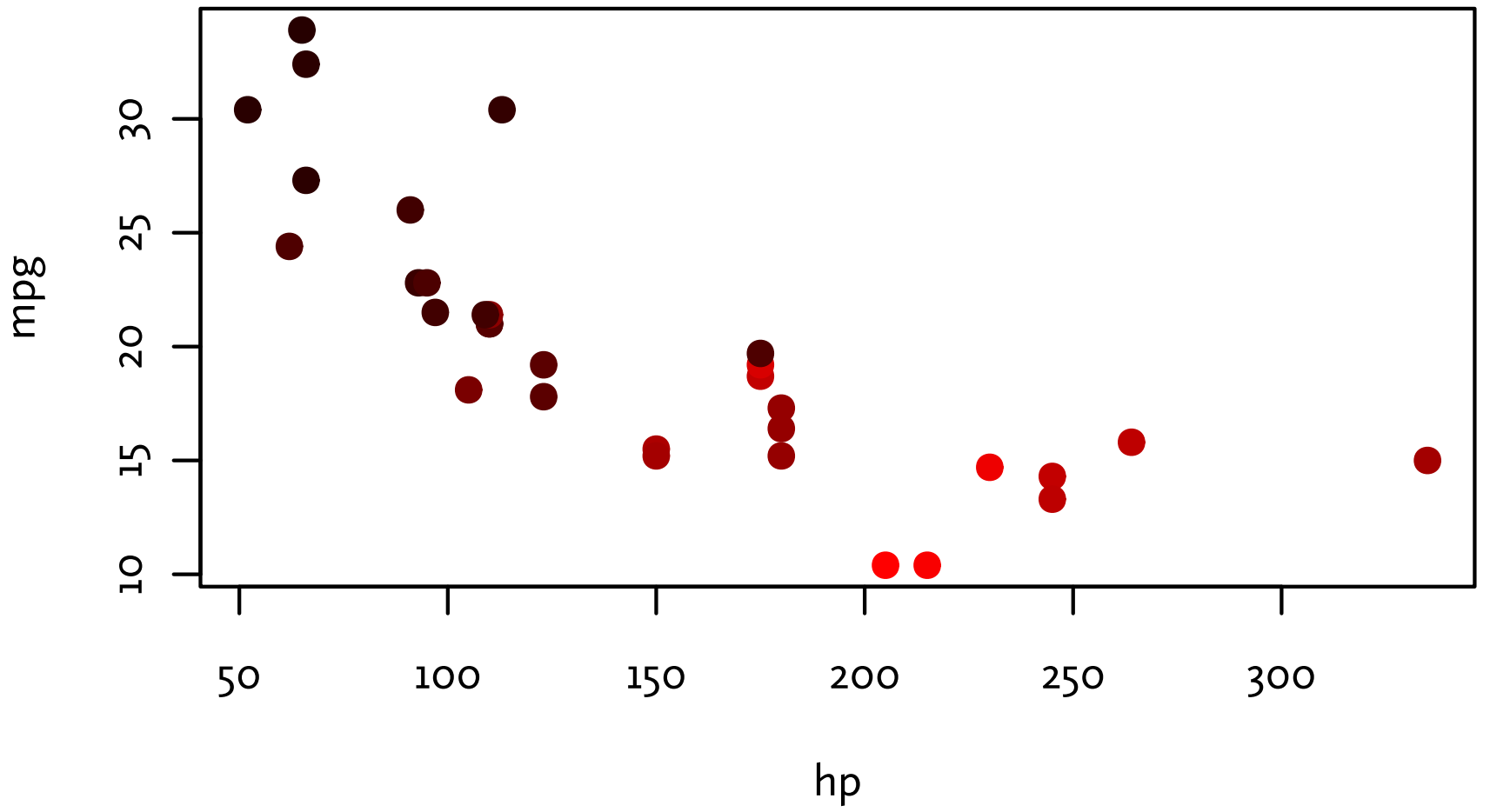


Other palette options exist.

- `scale_color/fill_gradient2()` creates a gradient between two extremes, with a midpoint (by default at 0). This creates a diverging scale for positives and negatives.
- `scale_color/fill_distiller` gives Color Brewer palettes for continuous variables.
- `scale_color/fill_fermenter` gives Color Brewer palettes for binned data.

With base R graphics, you can plug a variable into a function like rgb to get a gradient.

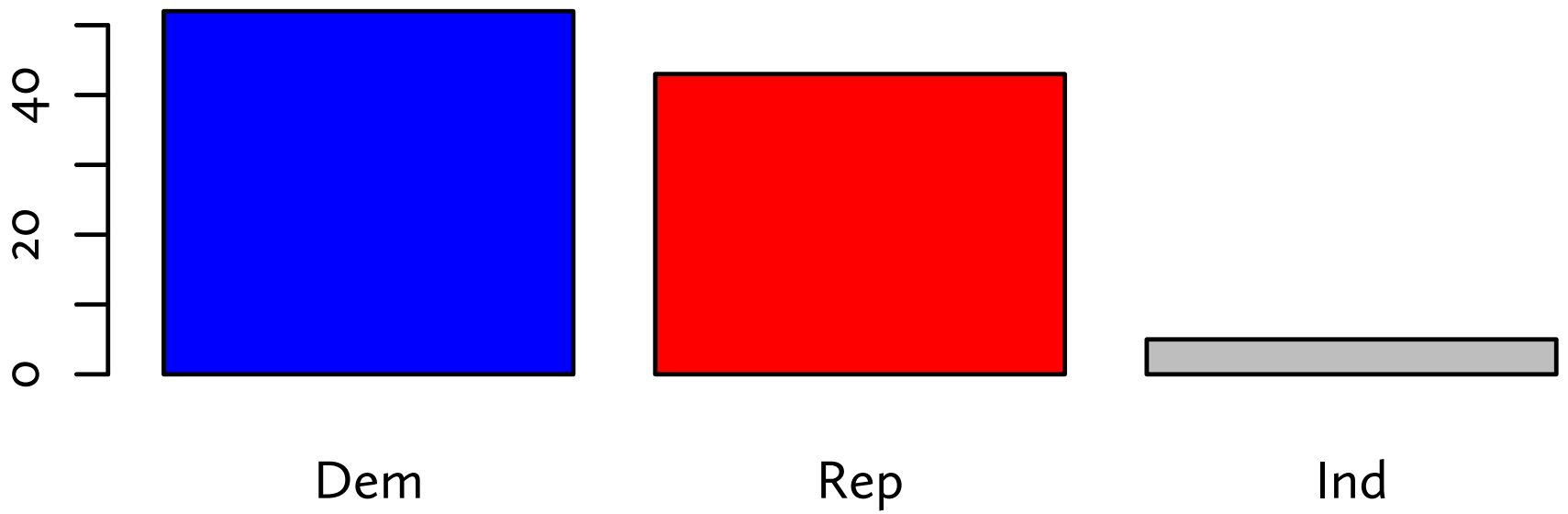
```
with(mtcars, plot(x=hp, y=mpg, col=rgb(disp/max(disp), 0, 0), pch=19))
```



To assign specific colors to specific factor levels in base R, make sure you know the order of the factors.

Then define a color pallet to match that order.

```
df = data.frame(
  Percent = c(52, 43, 5),
  Party = factor(c("Dem", "Rep", "Ind"), levels=c("Dem", "Rep", "Ind"))
)
party.color = c("blue", "red", "grey")
palette(party.color)
with(df, barplot(Percent, names.arg=Party, col=Party))
```



TRUMAN STATE UNIVERSITY

Department of STATISTICS

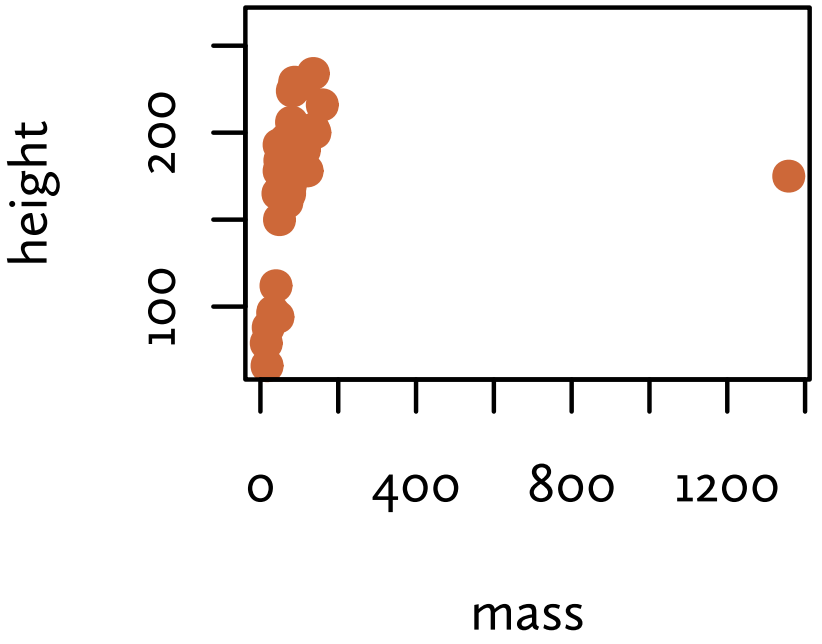
Colors in R

DISTINCT BY DESIGN

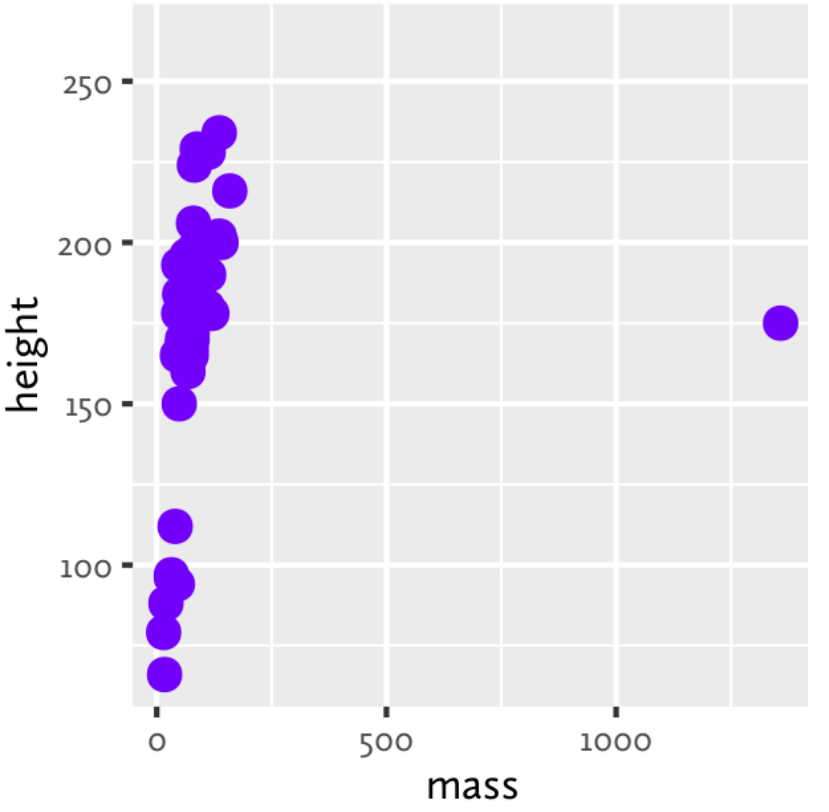
Specifying a Single Color

Single colors can be specified by one-word color names or RGB values in hexadecimal.

```
with(starwars,
  plot(mass, height,
  col="sienna3", pch=19))
```

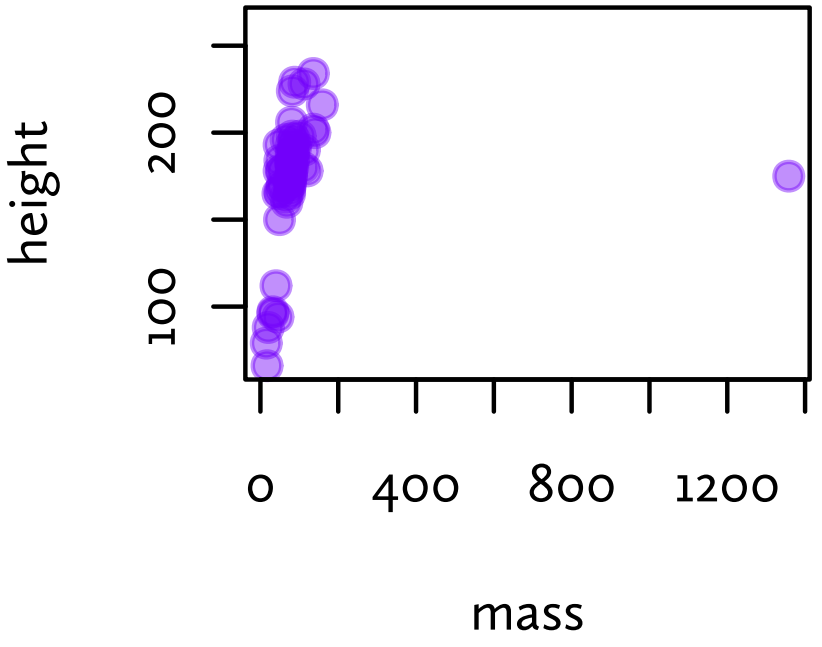



```
ggplot(starwars) +
geom_point(aes(x=mass, y=height),
  color="#7200FA", size=3)
```

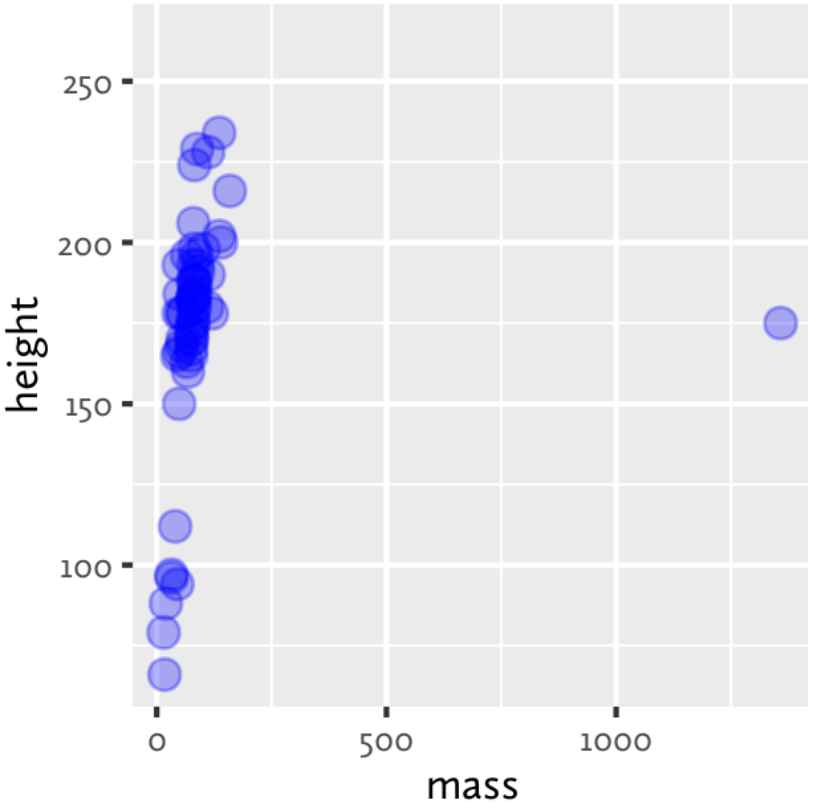


A fourth hex number specifies transparency. `ggplot` has an `alpha` aesthetic.

```
with(starwars,
  plot(mass, height,
  col="#7200FA70", pch=19))
```




```
ggplot(starwars) +
geom_point(aes(x=mass, y=height),
  color="blue", size=3, alpha=0.3)
```



Mid-Level: Other commands create color codes.

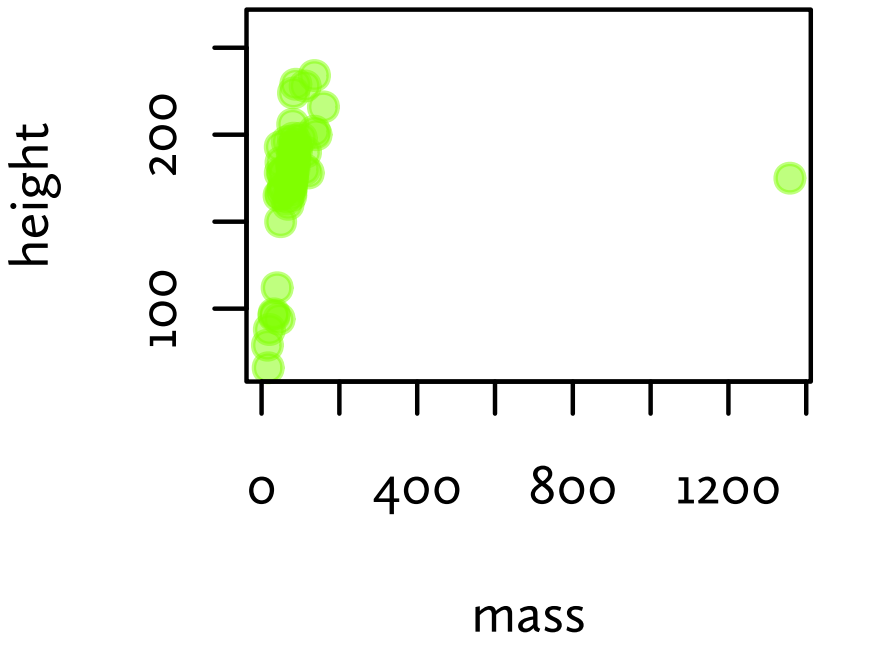
```
# See also `hsv` and `hcl`.
rgb(1, 0, 1)
```

```
[1] "#FF00FF"
```

```
grey(0.3)
```

```
[1] "#4D4D4D"
```

```
with(starwars,
  plot(mass, height,
  col=rgb(0.5, 1, 0, 0.5), pch=19))
```

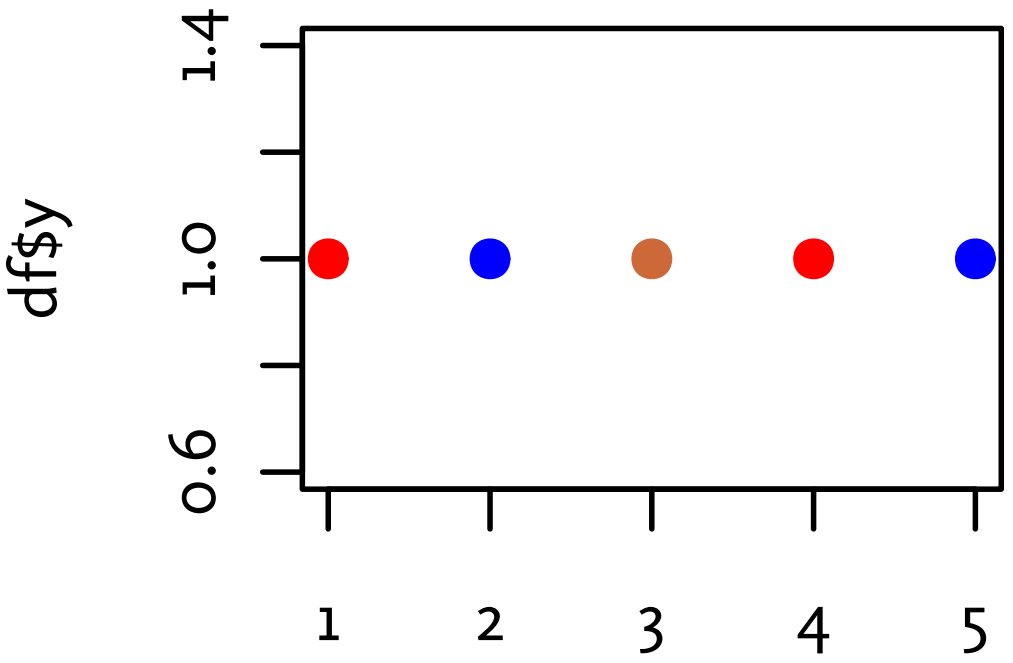


Vectors of Colors and Palettes

Color vectors can be directly specified.

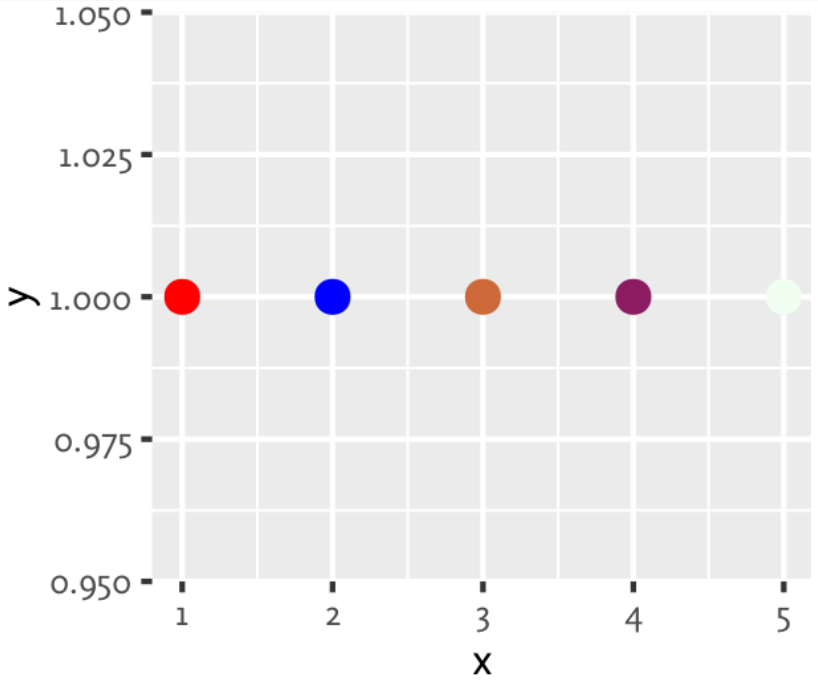
Used with base R graphics. The next graphical element gets the next color. The list starts over if you get to the end.

```
df <- data.frame(x=1:5, y=1)
col.vec = c("red", "blue", "sienna3")
plot(df$x, df$y, col=col.vec, pch=19)
```

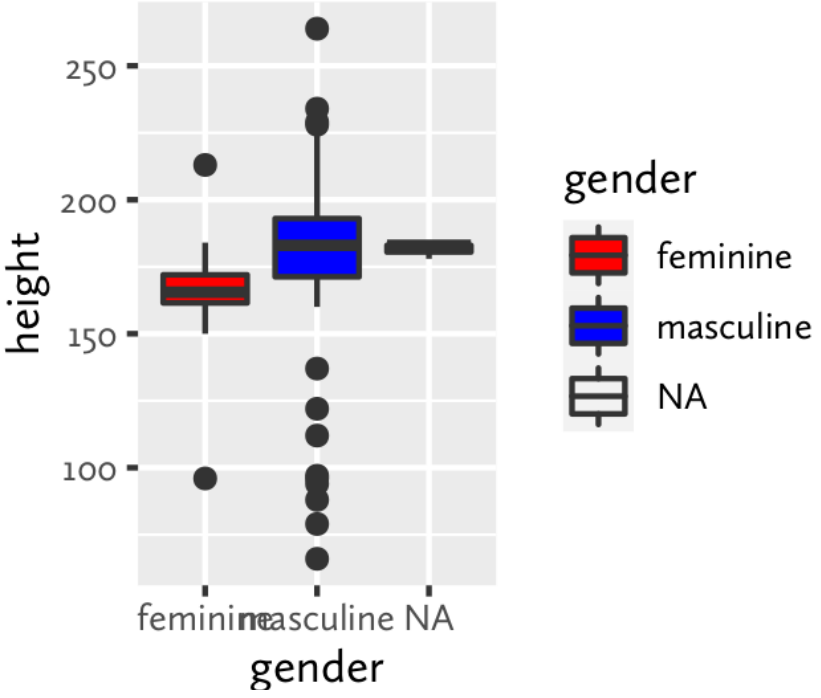


ggplot requires the list to be a single color or at least as long as the data set.

```
col.vec2 = c("red", "blue",
  "sienna3", "maroon4", "honeydew")
ggplot(df) +
  geom_point(aes(x=x, y=y),
    color=col.vec2, size=3)
```




```
ggplot(starwars) +
  geom_boxplot(aes(fill=gender,
    x=gender, y=height)) +
  scale_fill_manual(values=col.vec2)
```



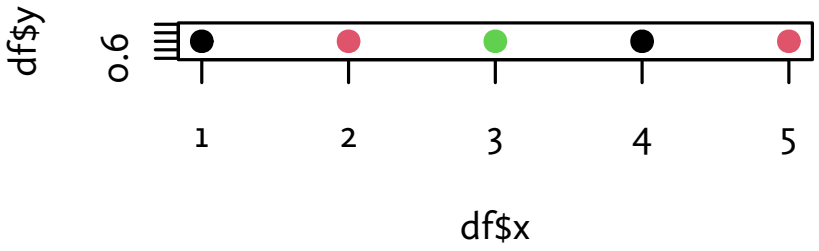
Colors specified by integers look up colors in *palettes*.

- If you specify an integer value for color, R will look up a corresponding color from its default *palette*.
- The `palette` command will also set a palette for base R graphics if given a vector of colors.
- Factors are secretly integers—that's how factors are mapped to colors.
- `ggplot` works a little differently (see below).

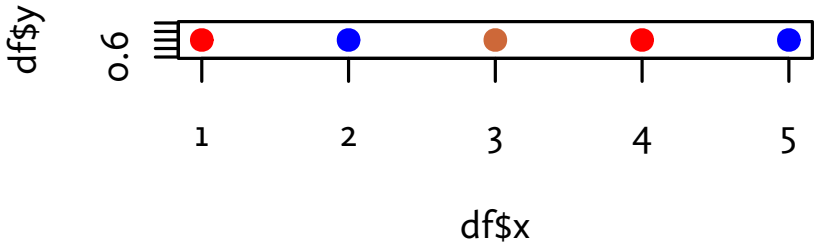
```
palette()
```

```
[1] "black"   "#DF536B" "#61D04F" "#22
[8] "gray62"
```

```
plot(df$x, df$y, col=1:3, pch=19)
```




```
palette(col.vec)
plot(df$x, df$y, col=1:3, pch=19)
```




```
palette("default")
```

Palette Commands automate the process of making vectors of colors.

- Generally, they have the form `palette_command(n, ...)` where `n` is the number of colors you want, and `...` may specify some other options.
- Syntax may vary by package.

```
# Built-in Palettes
rainbow(3)
```

```
[1] "#FF0000" "#00FF00" "#0000FF"
```

```
rainbow(5)
```

```
[1] "#FF0000" "#CCFF00" "#00FF66" "#0066FF" "#CC00FF"
```

```
heat.colors(4)
```

```
[1] "#FF0000" "#FF8000" "#FFFF00" "#FFFF80"
```

```
terrain.colors(5)
```

```
[1] "#00A600" "#E6E600" "#EAB64E" "#EEB99F" "#F2F2F2"
```

```
topo.colors(5)
```

```
[1] "#4C00FF" "#004CFF" "#00E5FF" "#00FF4D" "#FFFF00"
```

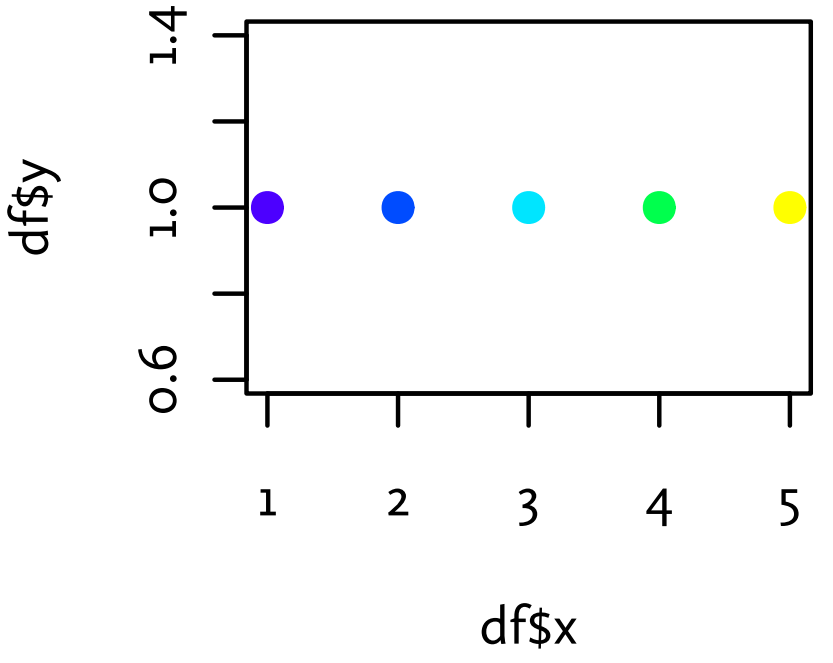
```
cm.colors(5)
```

```
[1] "#80FFFF" "#BFFFFF" "#FFFFFF" "#FFBFFF" "#FF80FF"
```

Palette Commands automate the process of making vectors of colors.

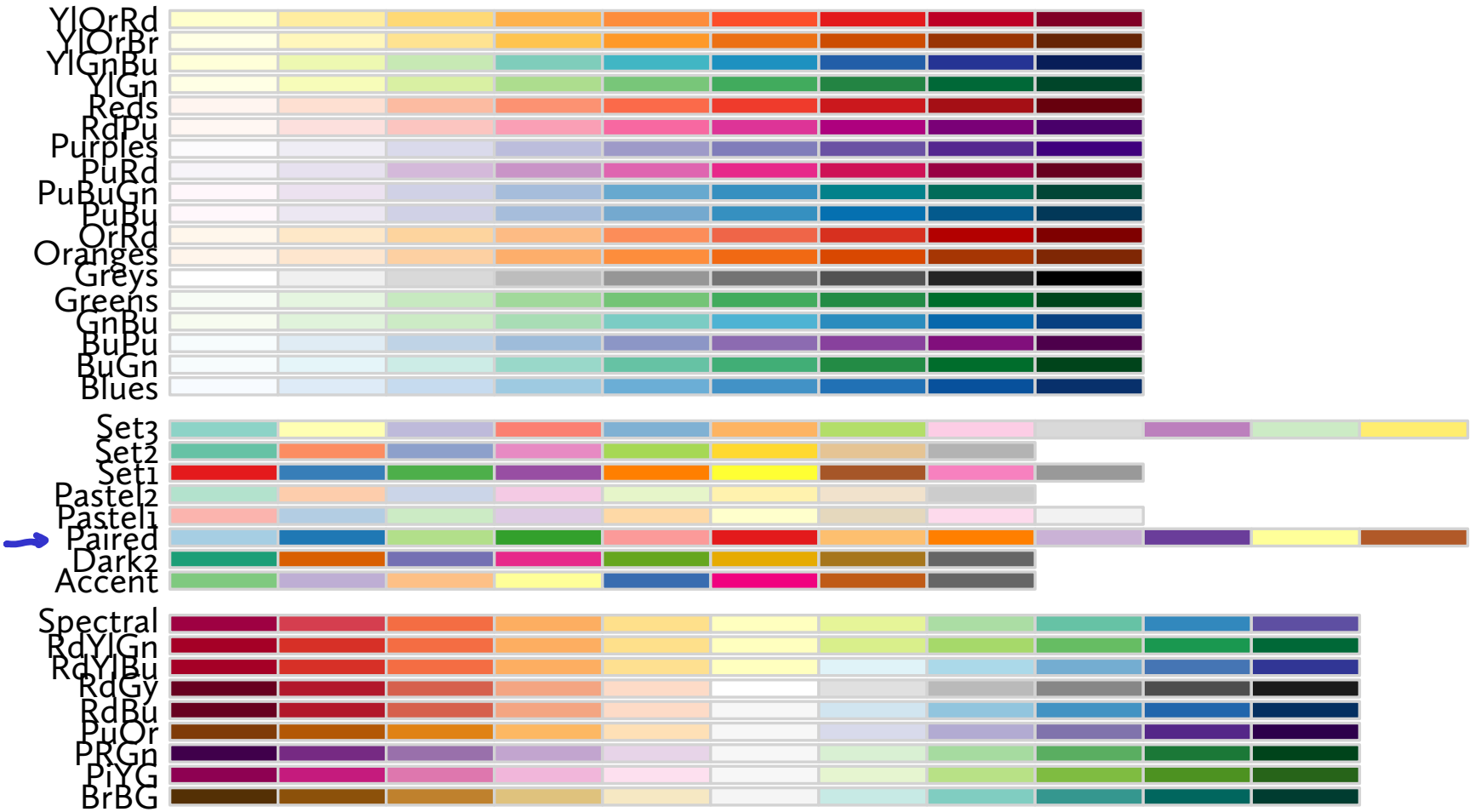
- Generally, they have the form `palette_command(n, ...)` where n is the number of colors you want, and `...` may specify some other options.
- Syntax may vary by package.

```
# Built-in Palettes
plot(df$x, df$y,
  col=topo.colors(5), pch=19)
```



Color Brewer is a commonly-used color palette package, featuring gradient, categorical and diverging palettes.

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



Others

See https://github.com/EmilHvitfeldt/r-color-palettes for many options.

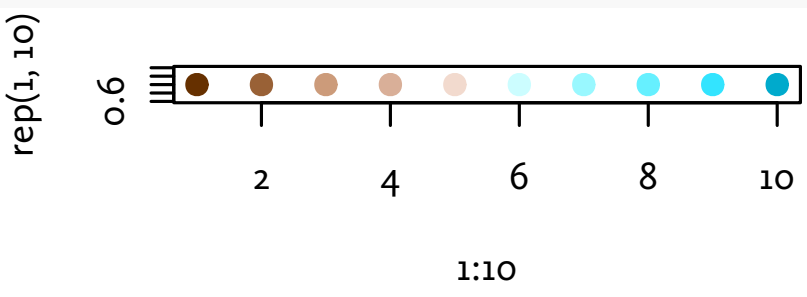
Warning: Various packages have various calling syntax. Some give lists of colors, some give functions that take n as shown above.

- viridis
- dichromat
- ghibli
- colorspace
- ggsci
- ggthemes

```
library(dichromat)
colorschemes$BrowntoBlue.10
```

```
 [1] "#663000" "#996136" "#CC9B7A" "#D
 [8] "#66F0FF" "#33E4FF" "#00AACC"
```

```
plot(x=1:10, y=rep(1, 10), pch=19,
  col=colorschemes$BrowntoBlue.10)
```



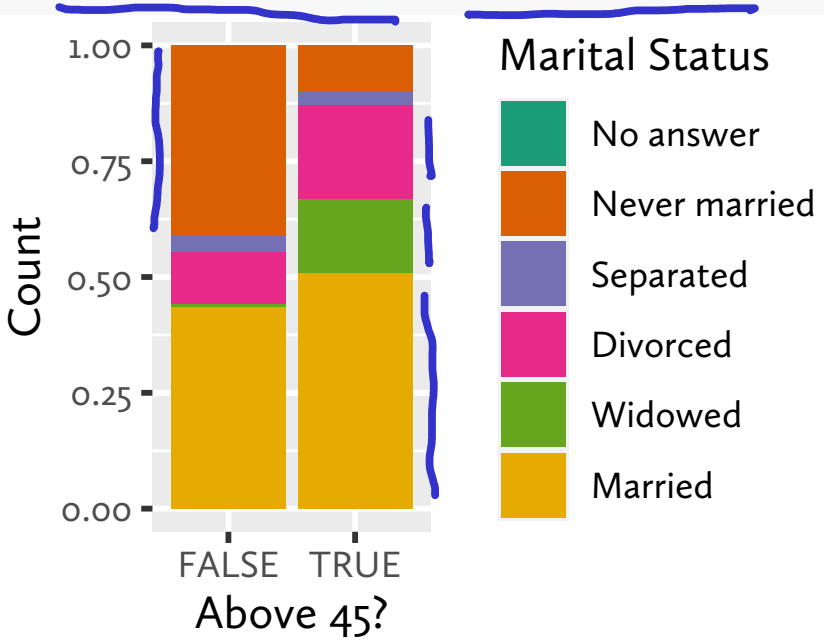
Difference between Base R and `ggplot`

- At the end of the day, Base R graphics tend to want a vector of colors.
  - How you get that vector of colors is up to you (thus the many formats from various packages).
  - Colors will be repeated, if necessary.
- On the other hand, `ggplot` has several methods:
  - it wants a single color (as `color=...`) or
  - a color for every data point if you're manually specifying colors (either `color=...` or one of the `scale_..._manual` commands).
  - `ggplot` also has many pre-built scale commands, as detailed next.

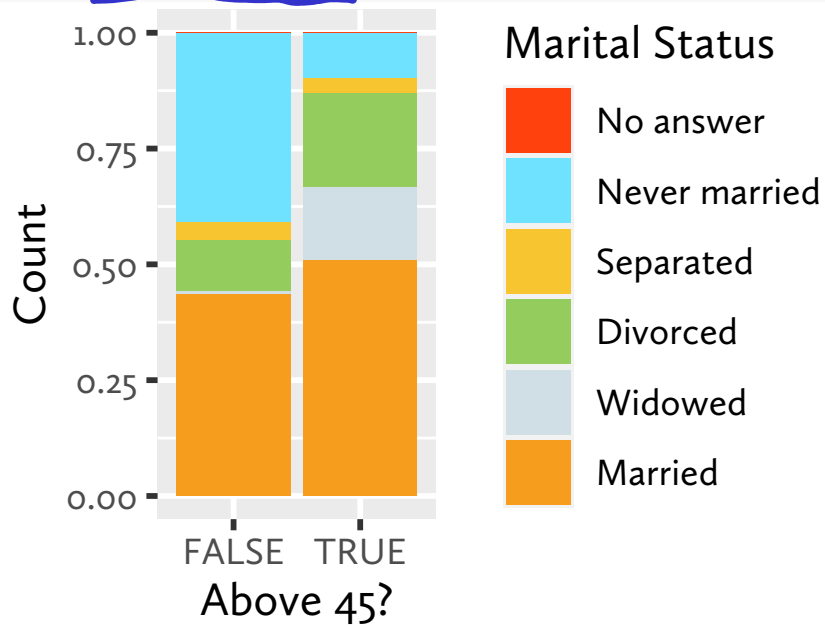
Built-in ggplot Color Scales

In ggplot, look for scale_fill... or scale_color... commands. They tend to count the n for you.

```r
# ggplot builds in color brewer palettes
gss_cat %>% select(age, marital) %>%
  na.omit() %>%
  mutate(above45 = age > 45) %>%
  ggplot(aes(x=above45, fill=marital)) +
  geom_bar(position="fill") +
  labs(y="Count", x="Above 45?",
    fill="Marital Status") +
  scale_fill_brewer(palette="Dark2")
```

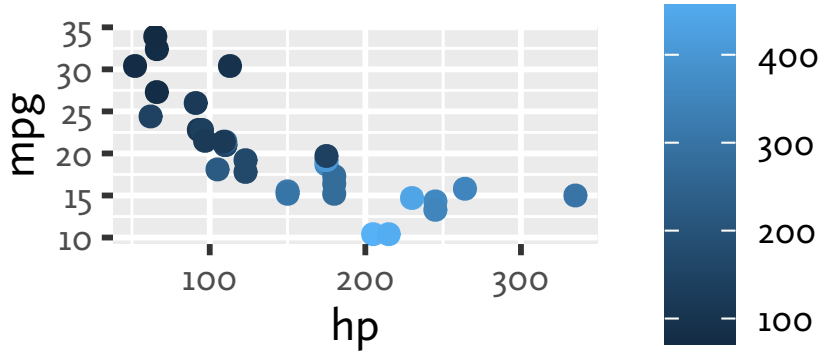



```r
library(ggsci)
gss_cat %>% select(age, marital) %>%
  na.omit() %>%
  mutate(above45 = age > 45) %>%
  ggplot(aes(x=above45, fill=marital)) +
  geom_bar(position="fill") +
  labs(y="Count", x="Above 45?",
    fill="Marital Status") +
  scale_fill_tron()
```

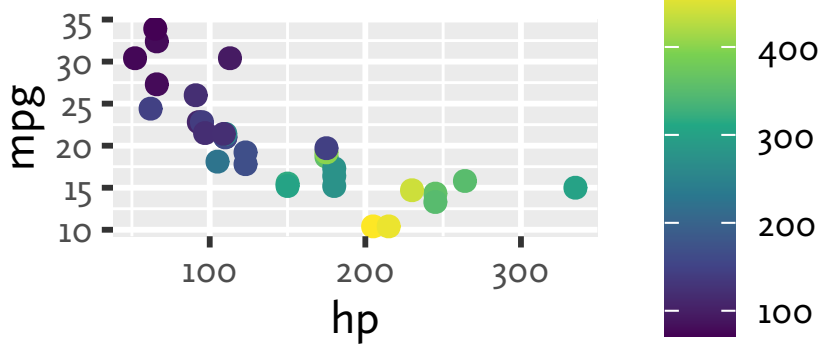


ggplot will allow you to set a **continuous** color map, but base R doesn't do that by default.

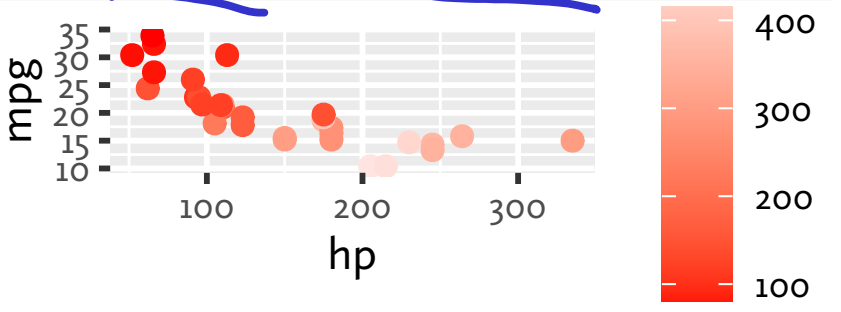
```
ggplot(mtcars, aes(x=hp, y=mpg)) +
  geom_point(aes(color=disp))
```



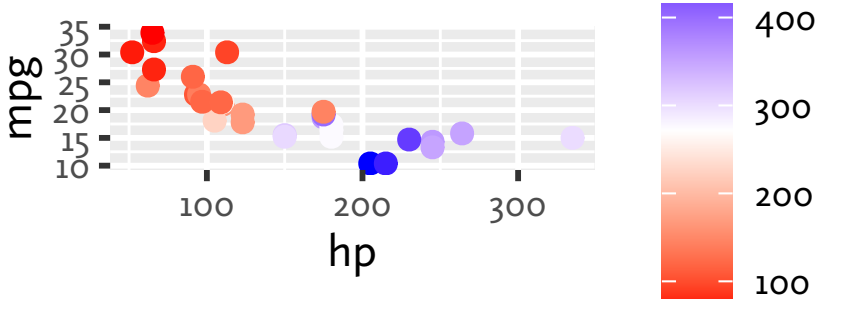

```
ggplot(mtcars, aes(x=hp, y=mpg)) +
  geom_point(aes(color=disp)) +
  scale_color_viridis_c()
```




```
ggplot(mtcars, aes(x=hp, y=mpg)) +
  geom_point(aes(color=disp)) +
  scale_color_gradient(
    low="red", high="mistyrose")
```




```
ggplot(mtcars, aes(x=hp, y=mpg)) +
  geom_point(aes(color=disp)) +
  scale_color_gradientn(
    colors=c("red", "white", "blue"))
```

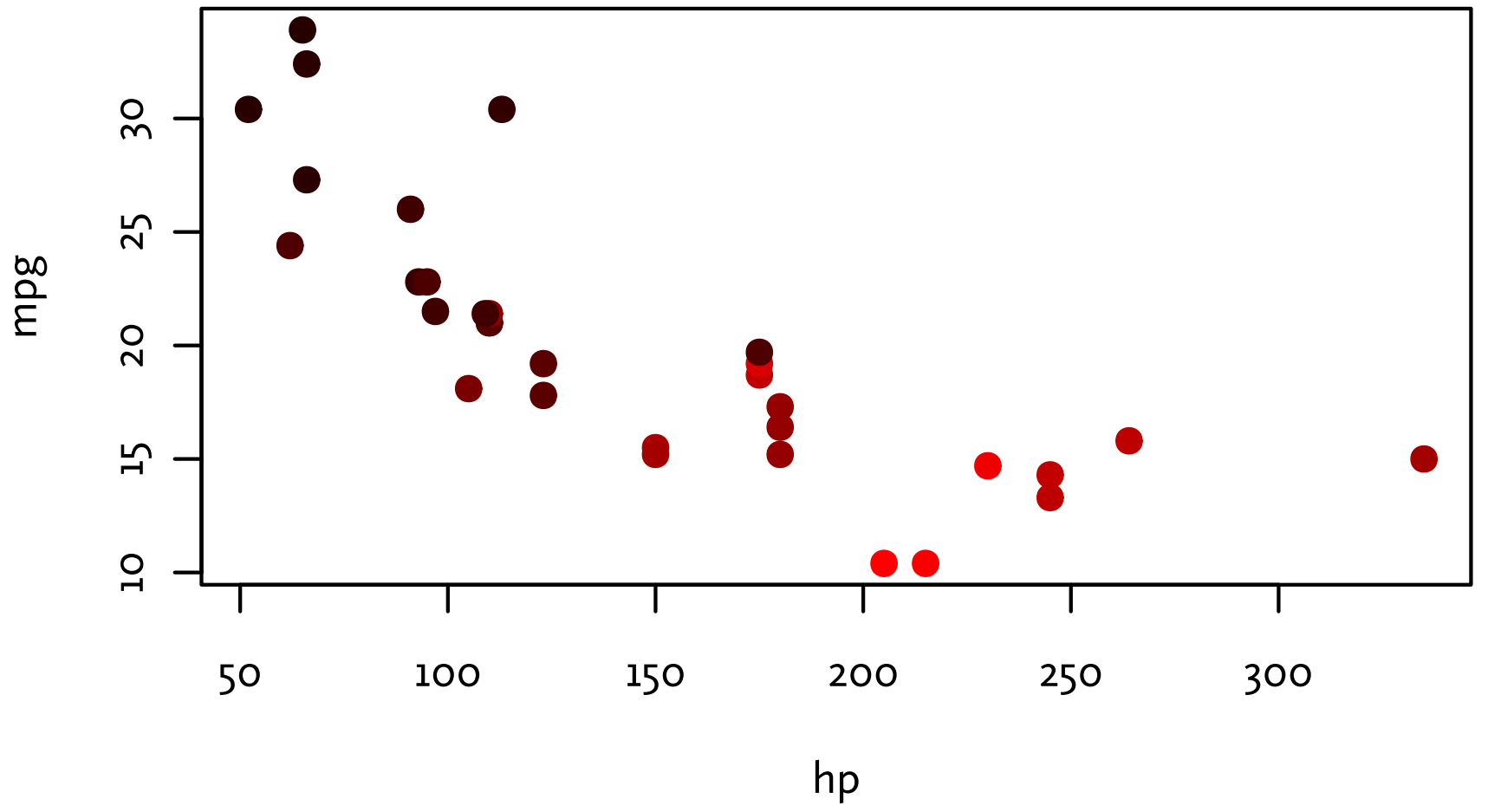


Other palette options exist.

- `scale_color/fill_gradient2()` creates a gradient between two extremes, with a midpoint (by default at 0). This creates a diverging scale for positives and negatives.
- `scale_color/fill_distiller` gives Color Brewer palettes for continuous variables.
- `scale_color/fill_fermenter` gives Color Brewer palettes for binned data.

With base R graphics, you can plug a variable into a function like rgb to get a gradient.

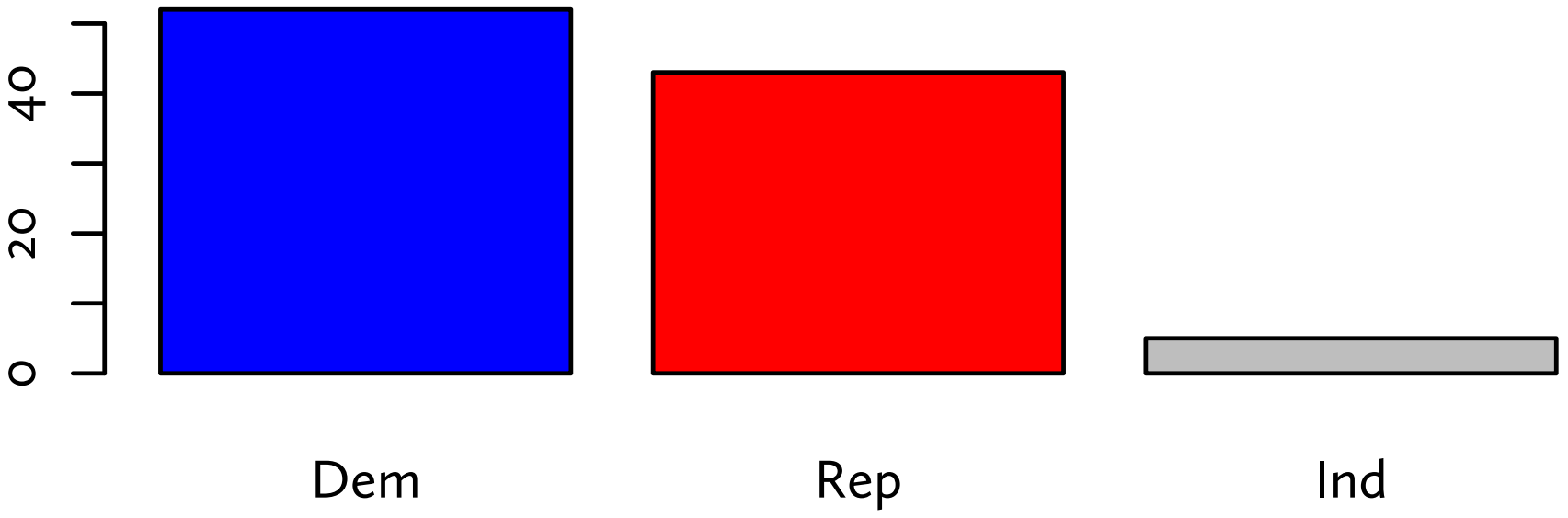
```
with(mtcars, plot(x=hp, y=mpg, col=rgb(disp/max(disp), 0, 0), pch=19))
```



To assign specific colors to specific factor levels in base R, make sure you know the order of the factors.

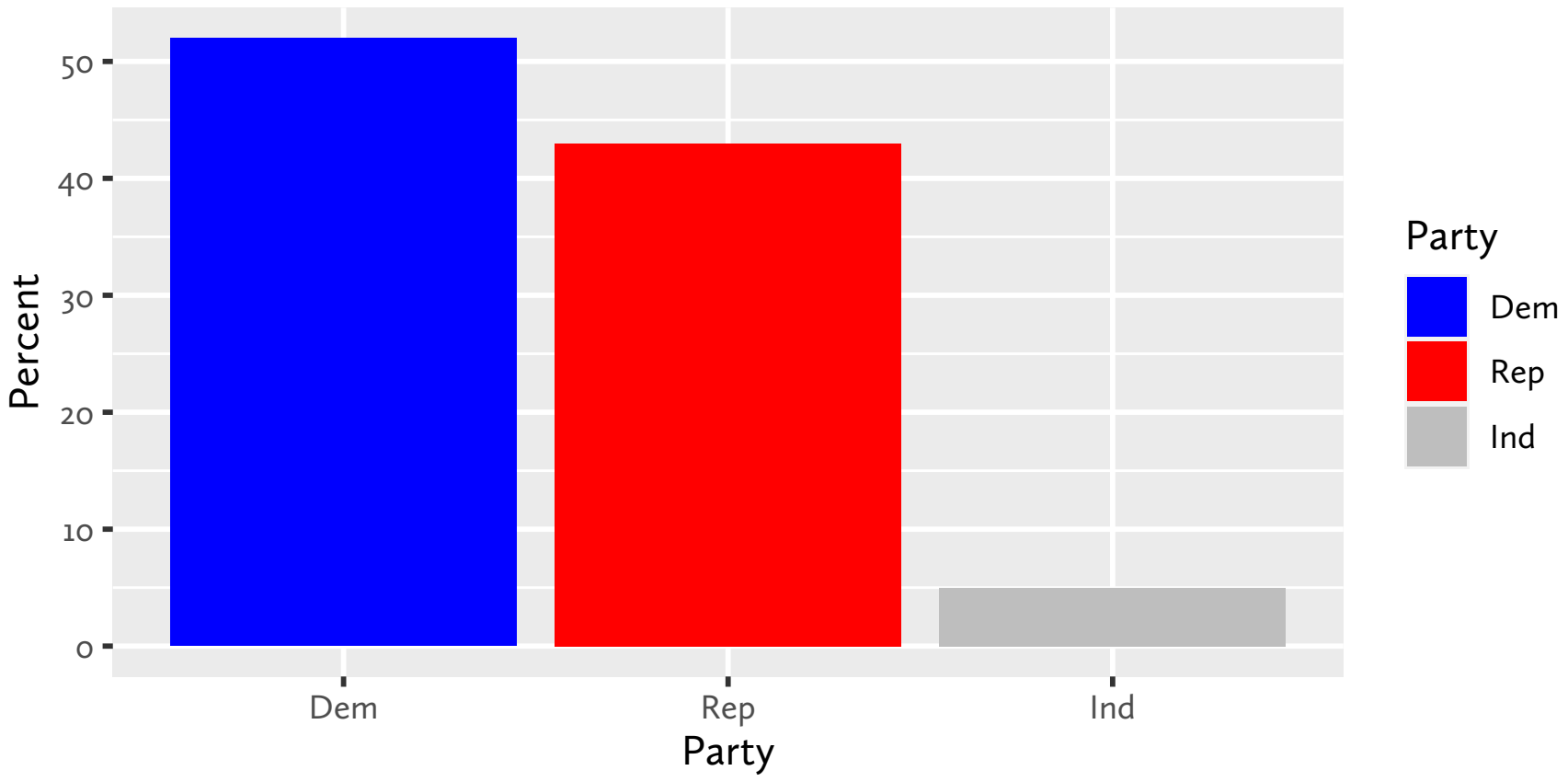
Then define a color pallet to match that order.

```
df = data.frame(
  Percent = c(52, 43, 5),
  Party = factor(c("Dem", "Rep", "Ind"), levels=c("Dem", "Rep", "Ind"))
)
party.color = c("blue", "red", "grey")
palette(party.color)
with(df, barplot(Percent, names.arg=Party, col=Party))
```



ggplot makes it easier with named color arguments.

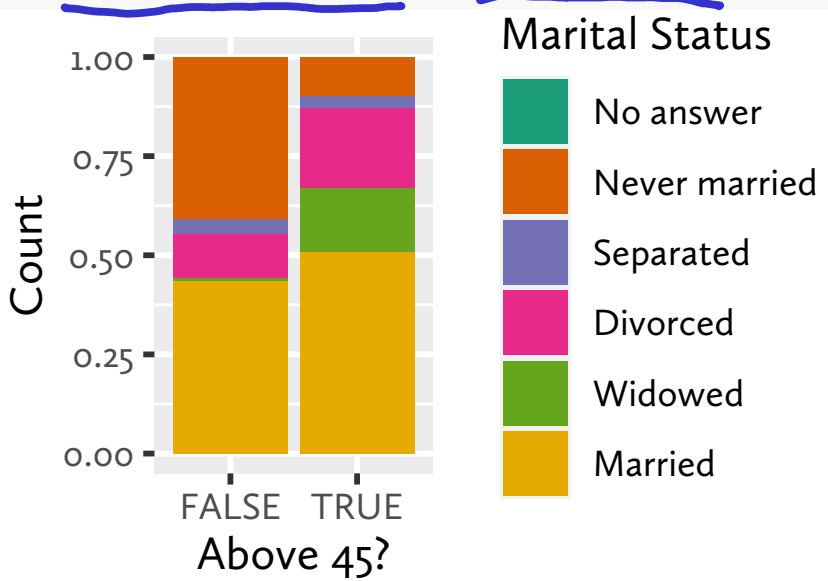
```
ggplot(df) +
  geom_col(aes(x=Party, y=Percent, fill=Party)) +
  scale_fill_manual(values=c("Rep"="red", "Dem"="blue", "Ind"="grey"))
```



Robust Color Choices

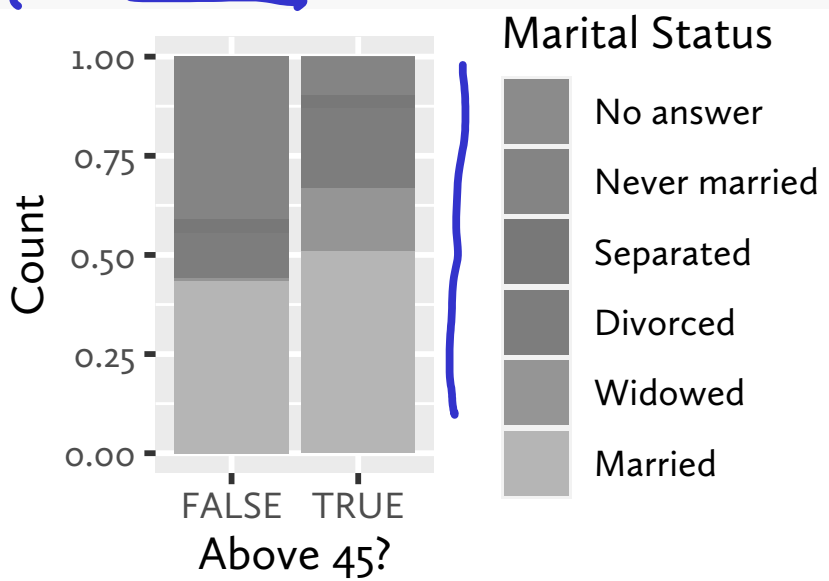
If someone will print your graphs in black and white, do the colors stand out. Use `desaturate` to test.

```
library(colorspace)
brew.pal <- brewer.pal(6, "Dark2")
gss_cat %>% select(age, marital) %>%
  na.omit() %>%
  mutate(above45 = age > 45) %>%
  ggplot(aes(x=above45, fill=marital)) +
  geom_bar(position="fill") +
  labs(y="Count", x="Above 45?",
    fill="Marital Status") +
  scale_fill_manual(values=brew.pal)
```



library(colorspace)

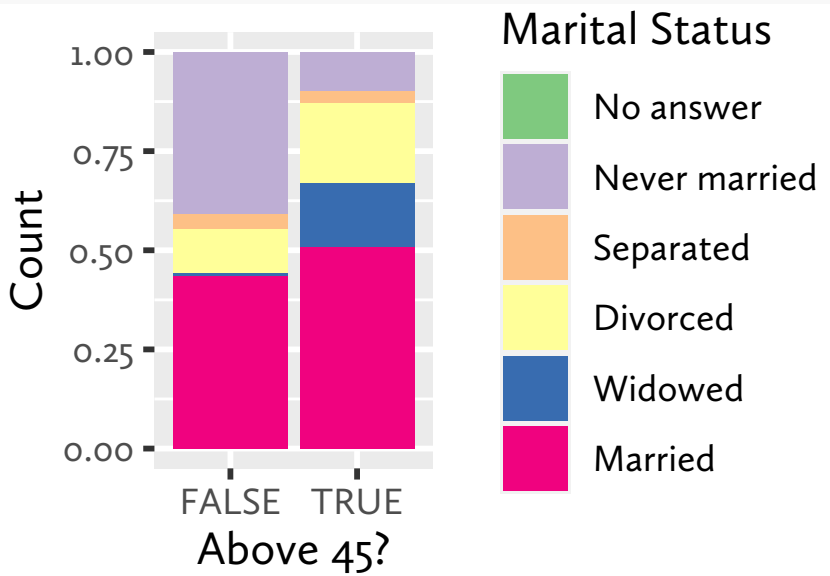
```
# Here we desaturate the palette.
gss_cat %>% select(age, marital) %>%
  na.omit() %>%
  mutate(above45 = age > 45) %>%
  ggplot(aes(x=above45, fill=marital)) +
  geom_bar(position="fill") +
  labs(y="Count", x="Above 45?",
    fill="Marital Status") +
  scale_fill_manual(values=
    desaturate(brew.pal))
```



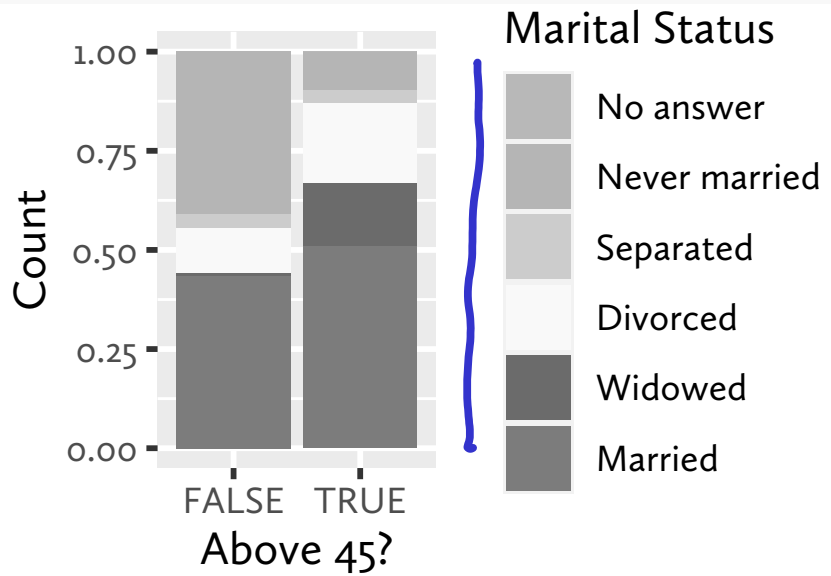
The "Accent" palette with more differences in value as well as hue might do better.

gradient scale → BW
Paired?

```
library(colorspace)
brew.pal <- brewer.pal(6, "Accent")
gss_cat %>% select(age, marital) %>%
  na.omit() %>%
  mutate(above45 = age > 45) %>%
  ggplot(aes(x=above45, fill=marital)) +
  geom_bar(position="fill") +
  labs(y="Count", x="Above 45?",
    fill="Marital Status") +
  scale_fill_manual(values=brew.pal)
```




```
# Here we desaturate the palette.
gss_cat %>% select(age, marital) %>%
  na.omit() %>%
  mutate(above45 = age > 45) %>%
  ggplot(aes(x=above45, fill=marital)) +
  geom_bar(position="fill") +
  labs(y="Count", x="Above 45?",
    fill="Marital Status") +
  scale_fill_manual(values=
    desaturate(brew.pal))
```



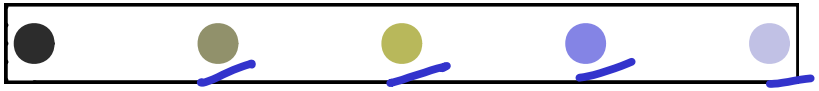
`RColorBrewer` claims to know which of its palettes are friendly to those with color blindness, but test first.

```
display.brewer.all(colorblindFriendly=TRUE)
```

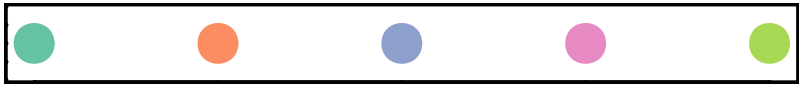
Viridis offers a few friendly palettes. The dichromat package lets you test. Which seem to work well?

```
library(dichromat); library(viridis)
par(mar=c(0,0,0,0))
pal <- palette("default")
plot(x=1:5, y=rep(1,5), col=pal, pch=19)
```

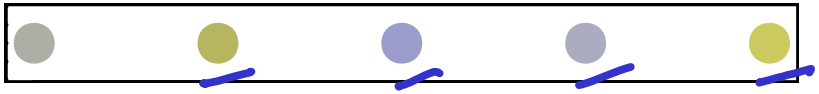
```
pal <- dichromat(pal, type="deutan")
plot(x=1:5, y=rep(1,5), col=pal, pch=19)
```



```
pal <- brewer.pal(5, "Set2")
plot(x=1:5, y=rep(1,5), col=pal, pch=19)
```



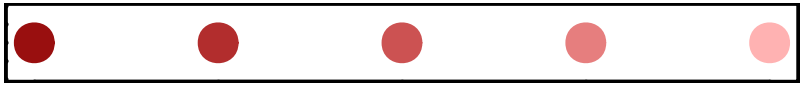
```
pal <- dichromat(pal, type="deutan")
plot(x=1:5, y=rep(1,5), col=pal, pch=19)
```



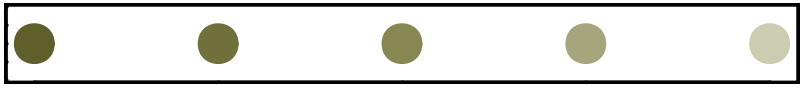
```
par(mar=c(0,0,0,0))
pal <- inferno(5)
plot(x=1:5, y=rep(1,5), col=pal, pch=19)
```

```
pal <- dichromat(pal, type="deutan")
plot(x=1:5, y=rep(1,5), col=pal, pch=19)
```

```
pal <- colorschemes$SteppedSequential.5
plot(x=1:5, y=rep(1,5), col=pal, pch=19)
```

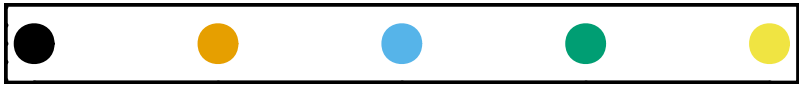


```
pal <- dichromat(pal, type="deutan")
plot(x=1:5, y=rep(1,5), col=pal, pch=19)
```

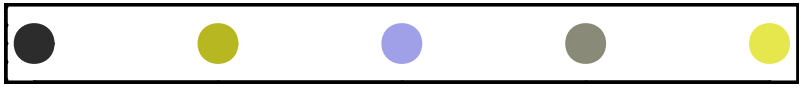


The khroma package provides other colorblind friendly palettes for base R and ggplot.

```
library(khroma)
par(mar=c(0,0,0,0))
pal <- color("okabe ito")(5)
plot(x=1:5, y=rep(1,5), col=pal, pch=19)
```

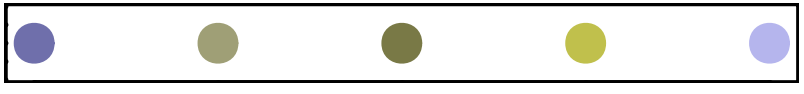


```
pal <- dichromat(pal, type="deutan")
plot(x=1:5, y=rep(1,5), col=pal, pch=19)
```



```
pal <- color("bright")(5)
plot(x=1:5, y=rep(1,5), col=pal, pch=19)
```

```
pal <- dichromat(pal, type="deutan")
plot(x=1:5, y=rep(1,5), col=pal, pch=19)
```



```
gr <- data.frame(x=1:5, y=1) %>%
  ggplot(aes(x=x, y=y)) +
  geom_point(aes(color=as.factor(x))) +
  theme_void()
gr + scale_color_okabeito(guide=NULL)
```

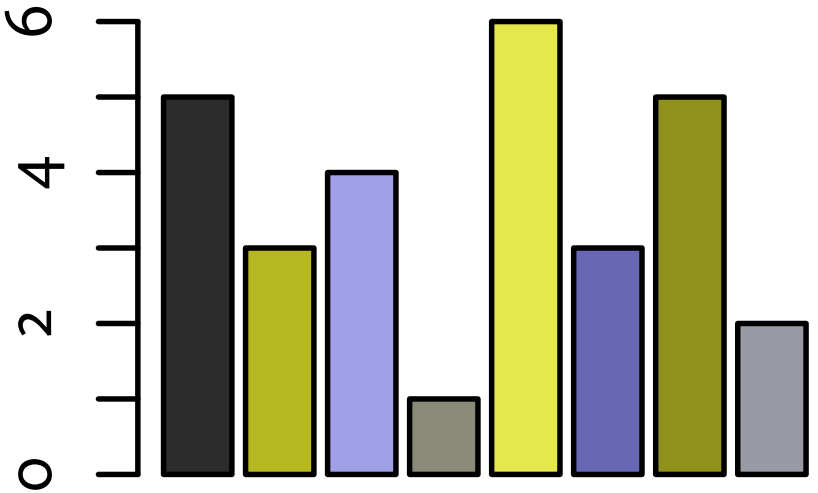
```
gr + scale_color_bright(guide=NULL)
```

```
gr + scale_color_vibrant(guide=NULL)
```

is relative

Readability depends on context: size, pattern, etc.

```
pal <- color("okabe ito")(8)
pal <- dichromat(pal, type="deutan")
barplot(c(5, 3, 4, 1, 6, 3, 5, 2), col
```



Drab?
Ugly Yellow?

```
plot(rnorm(100), rnorm(100), col=pal,
```