Introduction to Visualizing Data in R

```
library(car)
```

```
## Loading required package: carData
```

```
library(ggplot2)
```

```
load("helpdata.Rdata")
```

Learning Objectives

- Produce scatterplots, boxplots, and density plots using `ggplot2`.
- Set universal plot settings.
- Describe what faceting is and apply faceting in `ggplot2`.
- Modify the aesthetics of an existing `ggplot2` plot (e.g. axis labels and color).
- Build complex and customized plots from data in a data frame.

Inspection and exploration of data is necessary prior to statistical modeling. We start by installing and loading the required packages. **`ggplot2`** is included in the **`tidyverse`** package so you can just install the **`tidyverse`** package and that will automatically install **`ggplot2`**

```
install.packages("tidyverse")
install.packages("gridExtra")
install.packages("RColorBrewer")
install.packages("colorspace")
```

```
library(tidyverse)
```

There are three different plotting systems in R: base, lattice, and `ggplot2`. We will primarily focus on **`ggplot2`** with a couple of brief diversions into **base graphics**.

RStudio **`ggplot2`** cheat sheet

Overview of `ggplot2`

`ggplot2` is a plotting package that makes it simple to create complex plots from data in a data frame. It provides a more programmatic interface for specifying what variables to plot, how they are displayed, and general visual properties. Therefore, we only need minimal changes if the underlying data change or if we decide to change from a bar plot to a scatterplot. This helps in creating publication quality plots with minimal amounts of adjustments and tweaking.

`ggplot2` requires **tidy data**: i.e., a column for every dimension, and a row for every observation. Well structured data will save you lots of time when making figures with `ggplot2`.

`ggplot2` graphics are built step by step by adding new elements. Adding layers in this fashion allows for extensive flexibility and customization of plots.

To build a plot, we need to:

- use the `ggplot()` function and bind the plot to a specific data frame using the `data` argument

```
ggplot(data = helpdata)
```

- define aesthetics (`aes`), by selecting the variables to be plotted and the variables to define the presentation such as plotting size, shape color, etc.

```
ggplot(data = helpdata, aes(x = cesd, y = mcs))
```

- add `geoms` – graphical representation of the data in the plot (points, lines, bars). **`ggplot2`** offers many different `geoms`; we will use some common ones, including: * `geom_point()` for scatter plots, dot plots, etc. * `geom_boxplot()` for, well, boxplots! * `geom_line()` for trend lines, time-series, etc.

To add a `geom` to the plot use `+` operator.

Factors

A `factor` is a special `character` vector where the elements have pre-defined groups or 'levels'. You can think of these as qualitative or categorical variables. Factors are used to represent categorical data. Factors can be ordered or unordered, and understanding them is necessary for statistical analysis and for plotting. Factors are very useful and make R particularly well suited to working with data. Once created, factors can only contain a pre-defined set of values, known as *levels*. By default, R always sorts *levels* in alphabetical order.

Factors are stored as integers, and have labels (text) associated with these unique integers. While factors look (and often behave) like character vectors, they are actually integers under the hood, and you need to be careful when treating them like strings. Factors can be converted to `numeric` or `character` very easily.

For instance, if you have a factor with 2 levels:

```
sex <- factor(c("male", "female", "female", "male"))
str(sex)
```

```
##  Factor w/ 2 levels "female","male": 2 1 1 2
```

```
sex
```

```
## [1] male   female female male
## Levels: female male
```

R will assign `1` to the level `"female"` and `2` to the level `"male"` (because `f` comes before `m` in the alphabet, even though the first element in this vector is `"male"`). You can check this by using the function `levels()`, and check the number of levels using `nlevels()`:

```
levels(sex)
```

```
## [1] "female" "male"
```

```
nlevels(sex)
```

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

Sometimes, the order of the levels does not matter, other times you might want to specify the order because it is meaningful (e.g., "low", "medium", "high"), it improves your visualization, or it is required by a particular type of analysis. Here, one way to reorder our levels in the `sex` vector would be:

```
sex # current order
```

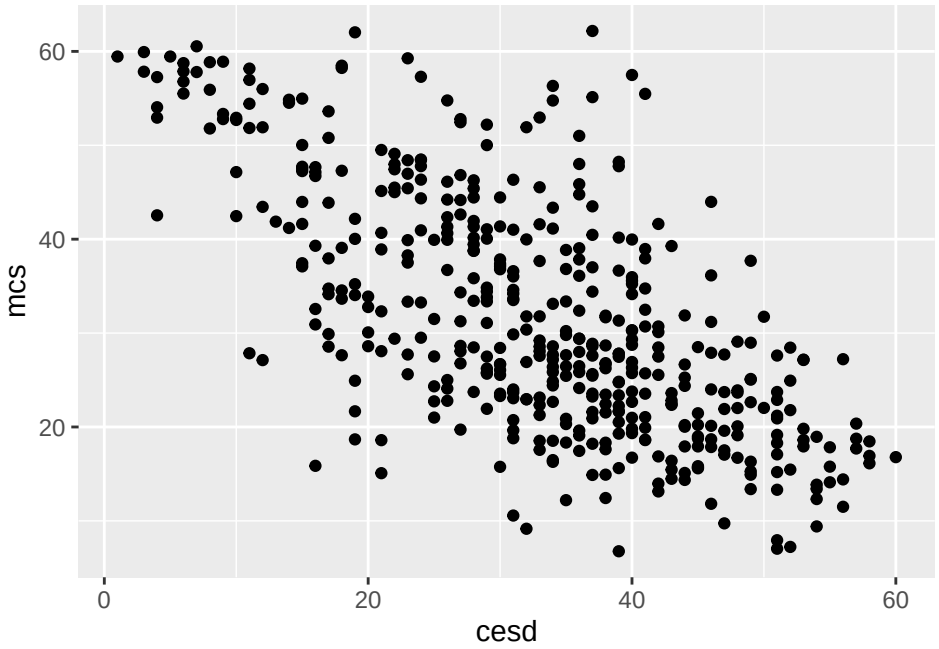
```
## [1] male   female female male
## Levels: female male
```

```
sex <- factor(sex, levels = c("male", "female"))
sex # after re-ordering
```

```
## [1] male   female female male
## Levels: male female
```

Scatterplots

```
ggplot(data = helpdata, aes(x = cesd, y = mcs)) +
  geom_point()
```



The + in the **ggplot2** package is particularly useful because it allows you to modify existing ggplot objects. This means you can easily set up plot "templates" and conveniently explore different types of plots, so the above plot can also be generated with code like this:

```
# Assign plot to an object
help_plot <- ggplot(data = helpdata, aes(x = cesd, y = mcs))

# Draw the plot
help_plot +
    geom_point()
```

Notes:

- Anything you put in the ggplot() function can be seen by any geom layers that you add (i.e., these are universal plot settings). This includes the x and y axis you set up in aes().
- You can also specify aesthetics for a given geom independently of the aesthetics defined globally in the ggplot() function.
- The + sign used to add layers must be placed at the end of each line containing a layer. If, instead, the + sign is added in the line before the other layer, **ggplot2** will not add the new layer and will return an error message.

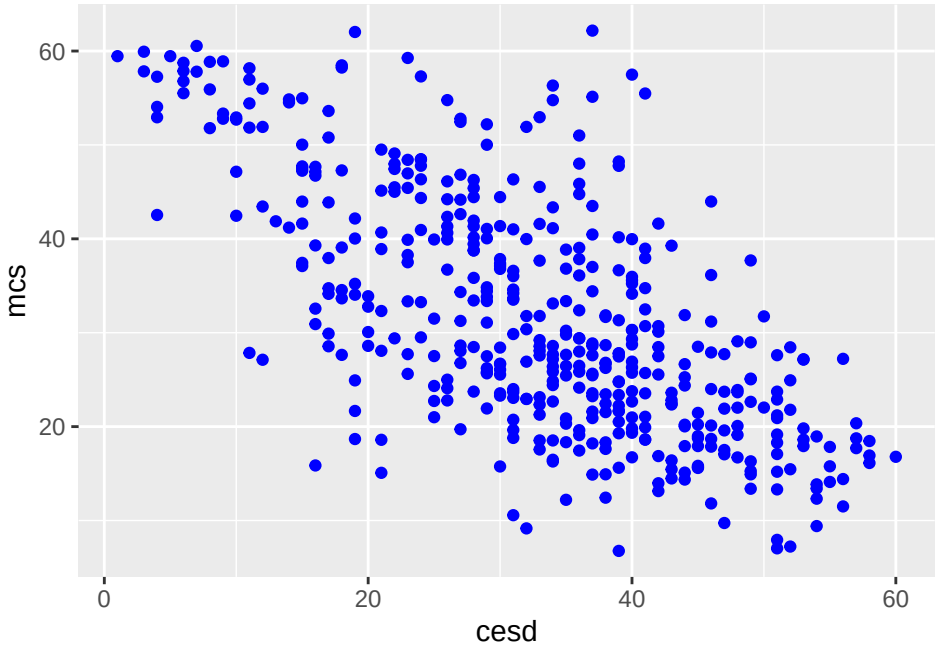
```
# this is the correct syntax for adding layers
help_plot +
  geom_point()

# this will not add the new layer and will return an error message
help_plot
```

```
  + geom_point()
```

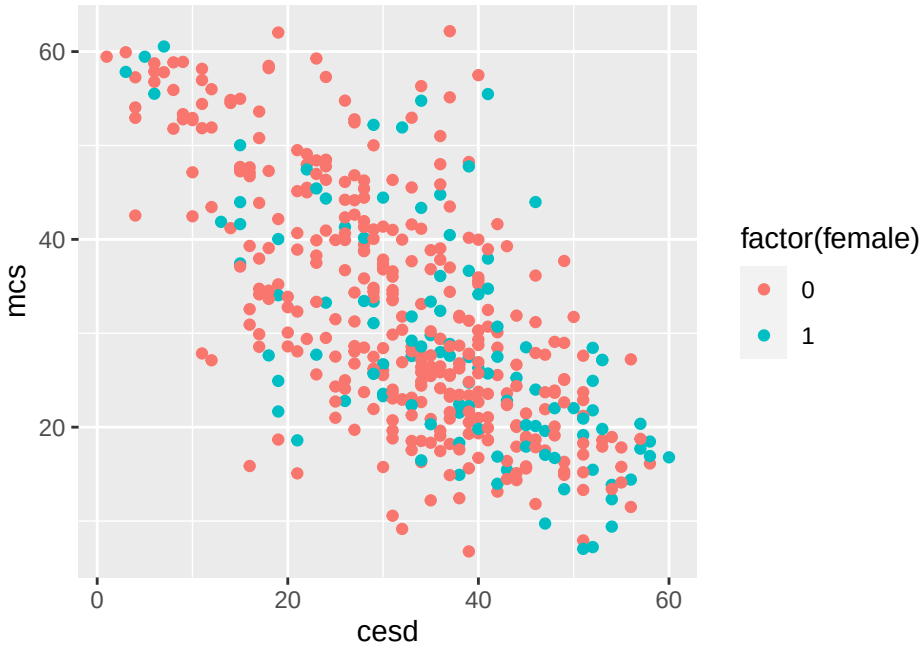
Building plots with ggplot is typically an iterative process. We start by defining the dataset we'll use, lay the axes, and choose a `geom`. Then, we start modifying this plot to, for example, add colors for all the points:

```
ggplot(data = helpdata, aes(x = cesd, y = mcs)) +
    geom_point(color = "blue")
```



We can also color each gender in the plot differently:

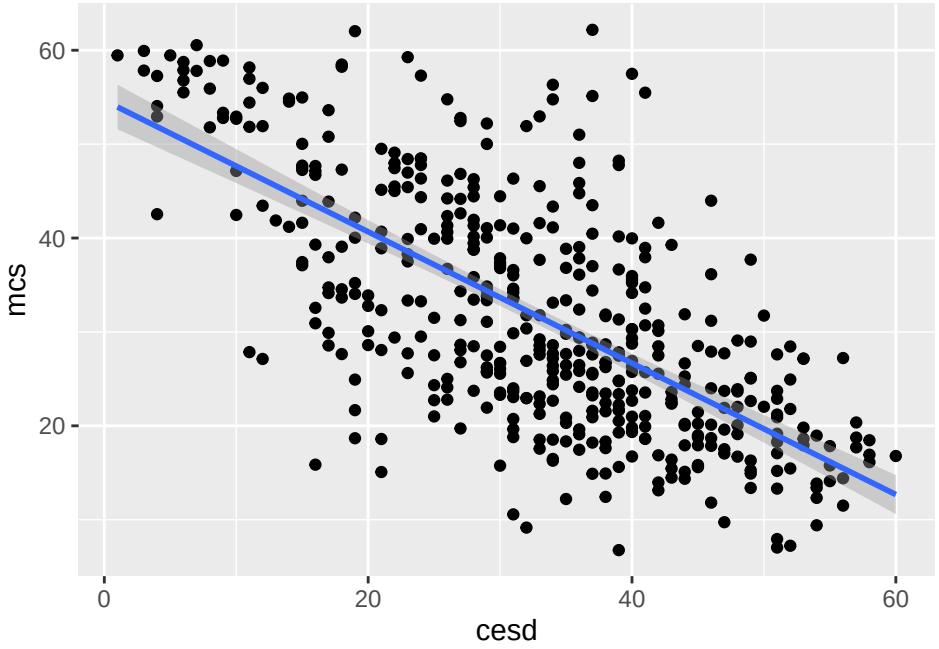
```
ggplot(data = helpdata, aes(x = cesd, y = mcs)) +
    geom_point(aes(color = factor(female)))
```



We can also add a fitted regression model line using `stat_smooth(method=lm)`:

```
ggplot(data = helpdata, aes(x = cesd, y = mcs)) +
    geom_point() +
    stat_smooth(method=lm)
```

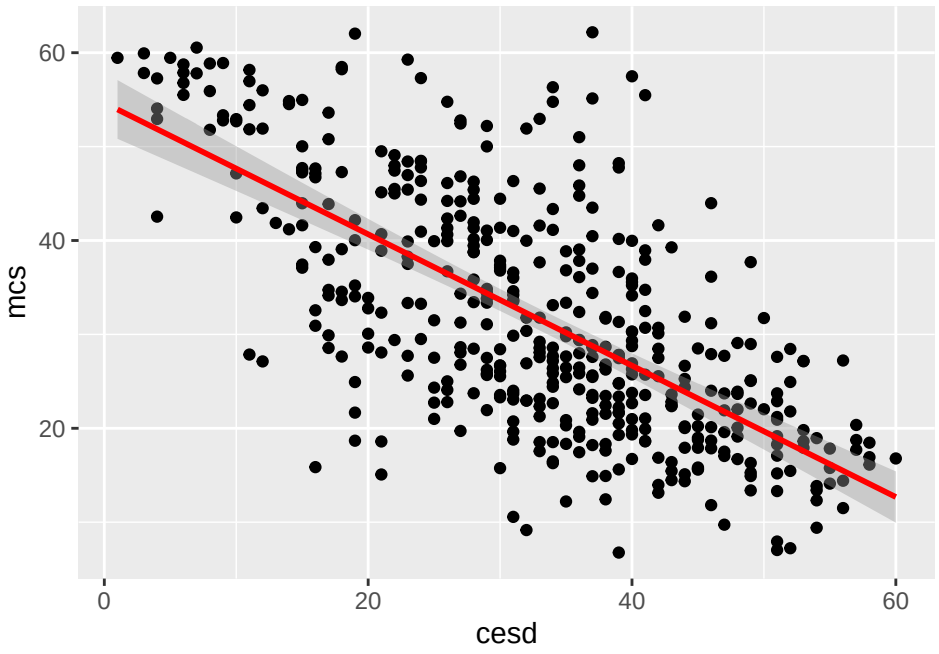
```
## `geom_smooth()` using formula = 'y ~ x'
```



By default a 95% confidence band was added. It can be changed to 90 or 99 using `level=` or it can be disabled using `se=FALSE`. The default color of the line (i.e., blue) can be changed using `color=`.

```
ggplot(data = helpdata, aes(x = cesd, y = mcs)) +
    geom_point() +
    stat_smooth(method = lm, level = 0.99, color = "red")
```

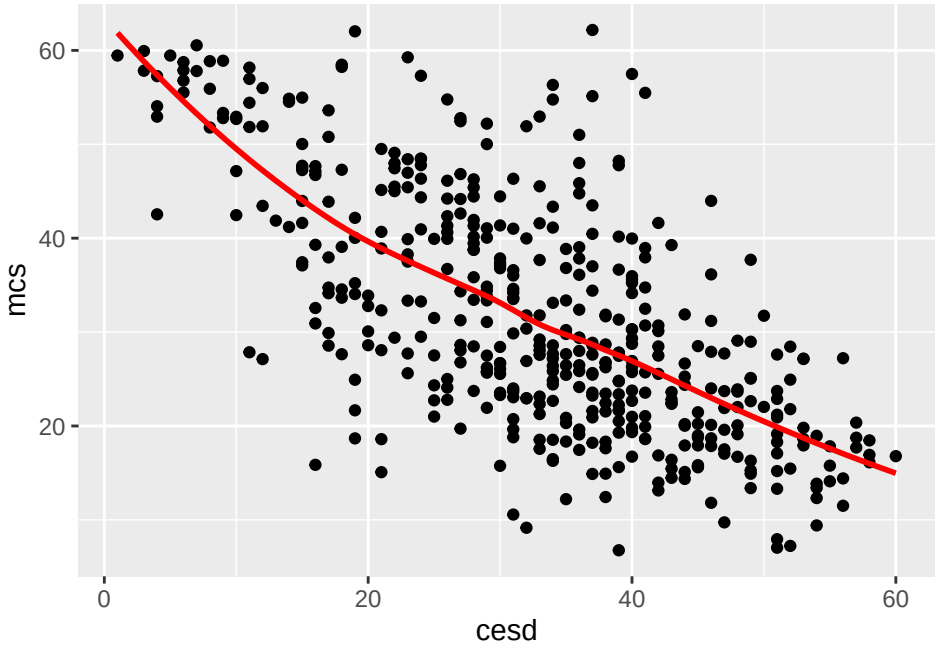
```
## `geom_smooth()` using formula = 'y ~ x'
```



If you do not specify `method=`, then by default `stat_smooth` adds a loess, locally weighted polynomial, curve.

```
ggplot(data = helpdata, aes(x = cesd, y = mcs)) +
    geom_point() +
    stat_smooth(se = FALSE, color = "red")
```

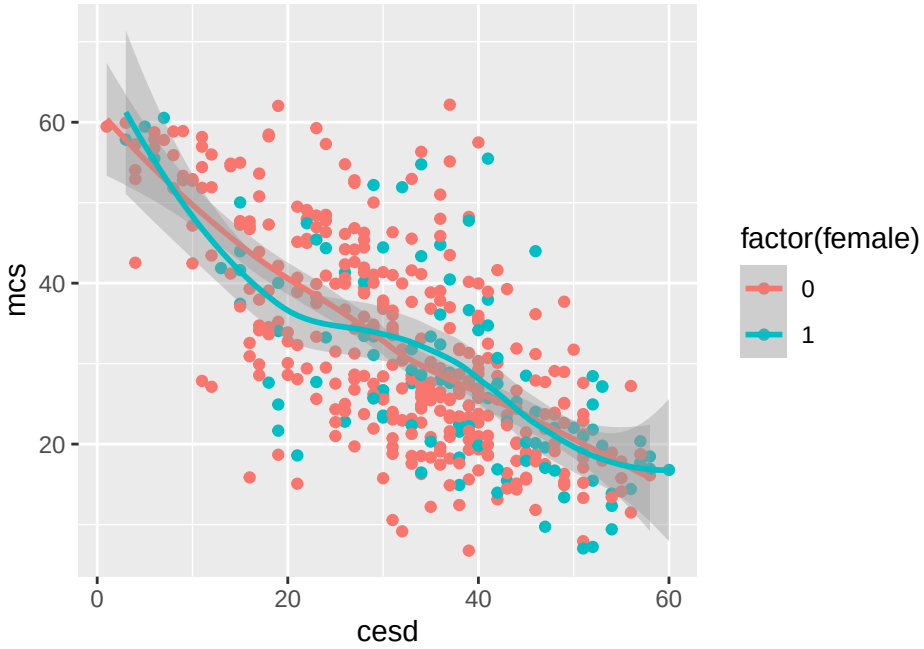
```
## `geom_smooth()` using method = 'loess' and formula = 'y ~ x'
```



For our data, this doesn't make much of a difference. We can also fit lines for each gender.

```
ggplot(data = helpdata, aes(x = cesd, y = mcs, color = factor(female))) +
    geom_point() +
    geom_smooth()
```

```
## `geom_smooth()` using method = 'loess' and formula = 'y ~ x'
```

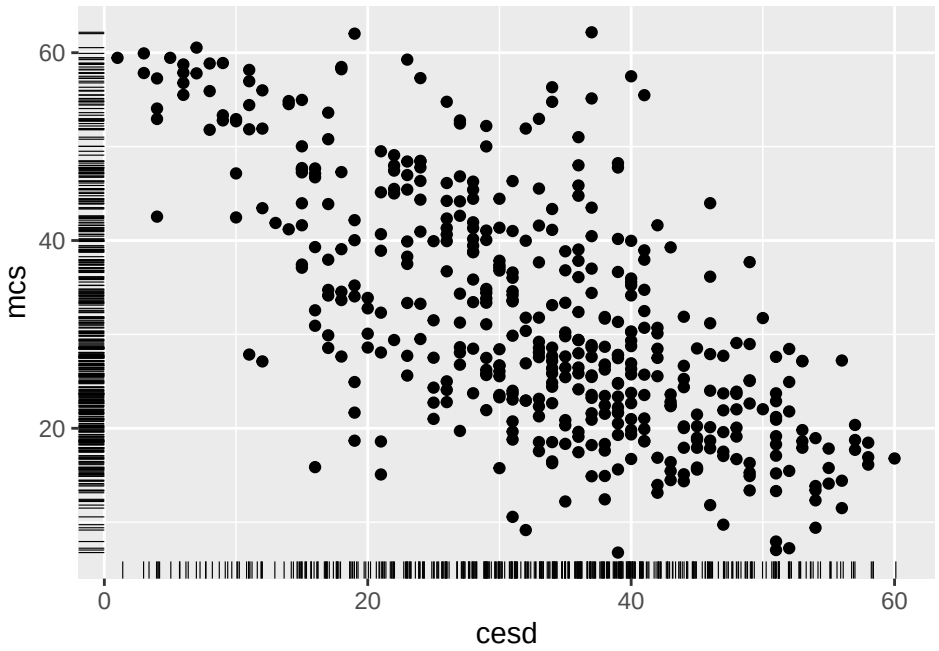


Note that the lines are limited to the range of the x-axis data for each group. In other words, it does not extrapolate outside of the observed data.

You can also add what are called marginal rugs to the plot. A marginal rug plot is essentially a one-dimensional scatterplot that can be used to visualize the distribution of data on each axis.

```
ggplot(data = helpdata, aes(x = cesd, y = mcs)) +
  geom_point() +
  geom_rug(position="jitter", size=.2)
```

```
## Warning: Using `size` aesthetic for lines was deprecated in ggplot2 3.4.0.
## i Please use `linewidth` instead.
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_lifecycle_warnings()` to see where this warning was
## generated.
```



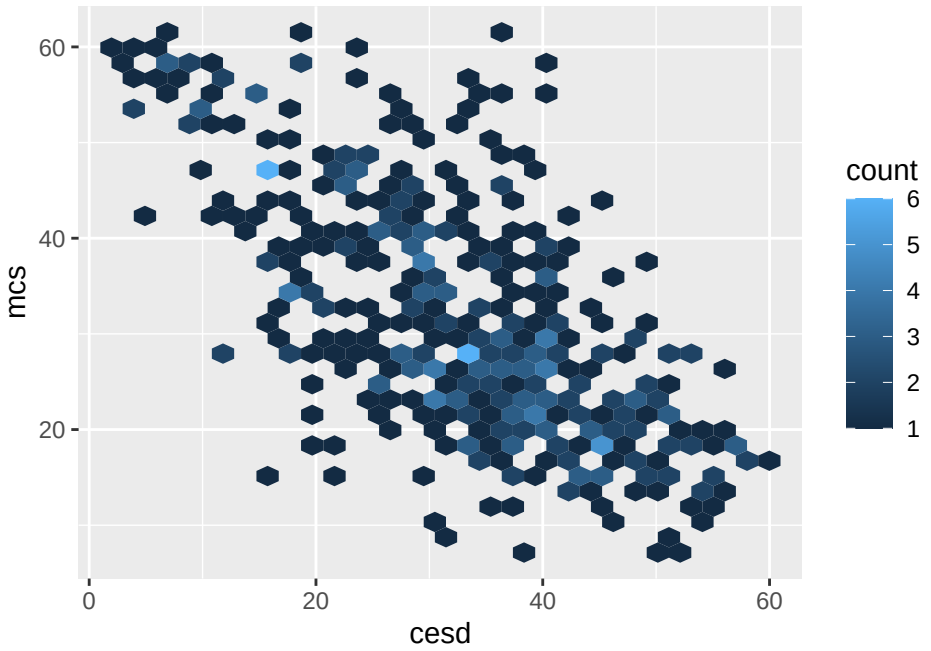
I added the options `position="jitter"` and `size=.2` to reduce overplotting of the rug lines.

Handling overplotting

Scatter plots can be useful exploratory tools for small datasets. For data sets with large numbers of observations, overplotting of points (meaning that the points obscure each other) can be a limitation of scatter plots. One strategy for handling such settings is to use hexagonal binning of observations. The plot space is tessellated into hexagons. Each hexagon is assigned a color based on the number of observations that fall within its boundaries. To use hexagonal binning with **ggplot2**, load the R package `hexbin`:

Then use the `geom_hex()` function:

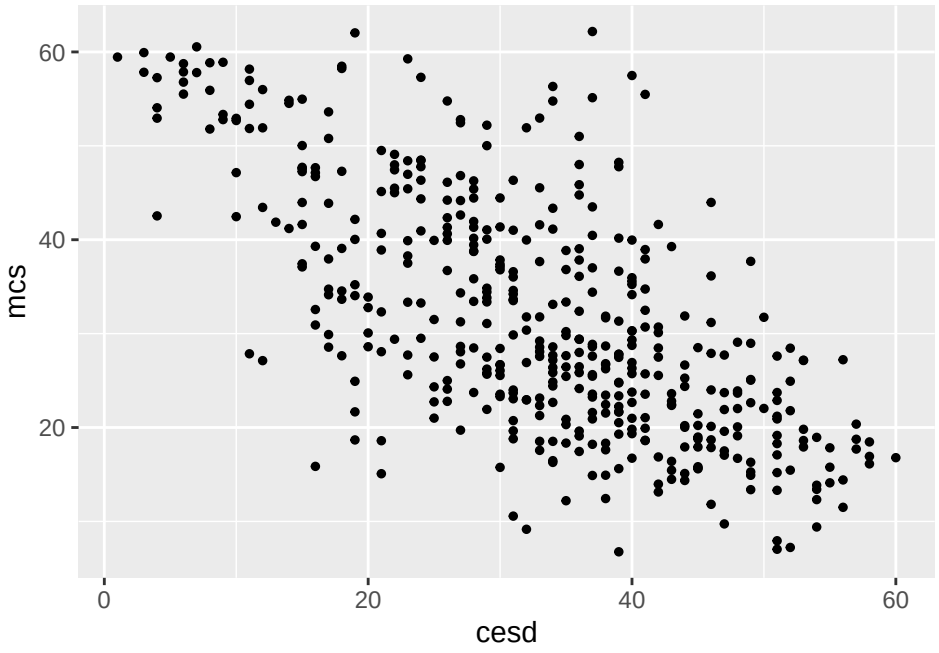
```
library(hexbin)
ggplot(data = helpdata, aes(x = cesd, y = mcs)) +
  geom_hex()
```



What are the relative strengths and weaknesses of a hexagonal bin plot compared to a scatter plot? Examine the above scatter plot and compare it with the hexagonal bin plot.

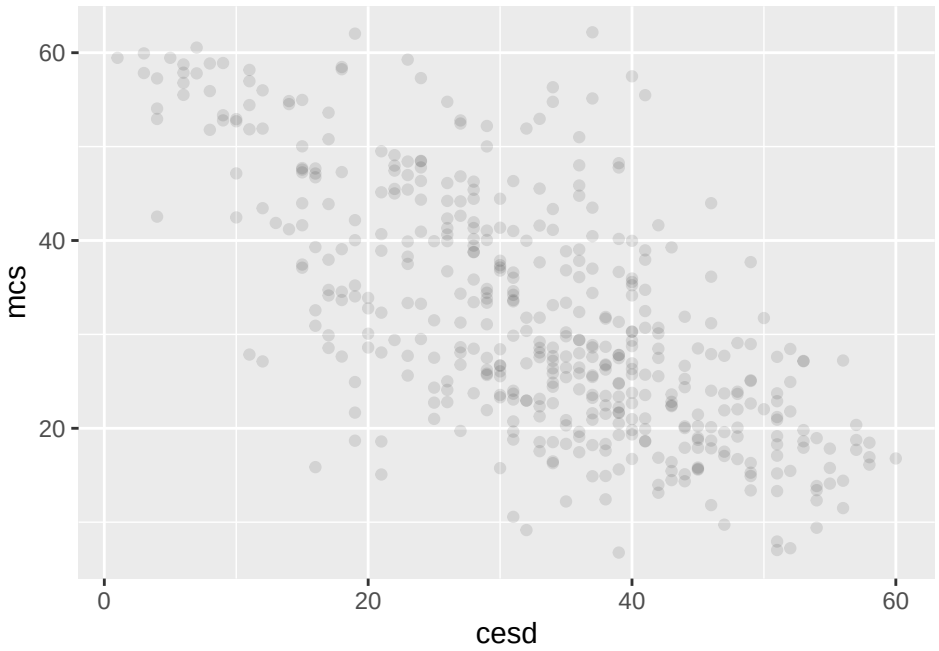
Another option to handle overplotting is to decrease the size of the points. The default point size is 2.

```
ggplot(data = helpdata, aes(x = cesd, y = mcs)) +
    geom_point(size=1)
```



Yet another option is to add transparency by using `alpha`.

```
ggplot(data = helpdata, aes(x = cesd, y = mcs)) +
    geom_point(alpha = 0.1)
```



A few points about scatterplots:

Scatterplots in which one or both variables are highly skewed are difficult to examine because the bulk of the data congregate in a small part of the display. It often helps to 'correct' substantial skews prior to examining the relationship between X and Y.

Scatterplots in which the variables are discrete are difficult to examine. Scatterplots of the relationship between discrete variables can be enhanced by randomly jittering the data.

Boxplots

Boxplots (due to John Tukey) present summary information on center, spread, skewness, and outliers.

1. A scale is laid off to accommodate the extremes of the data
2. The central box is drawn between the hinges, which are simply defined quartiles, and therefore encompasses the middle half of the data. The line in the central box represents the median.
3. The following rule is used to identify outliers, which are shown individually in the boxplot:

a. The hinge-spread (or inter-quartile range) is the difference between the hinges: H-spread $= H_U - H_L$

b. The 'fences' are located 1.5 hinge-spreads beyond the hinges:

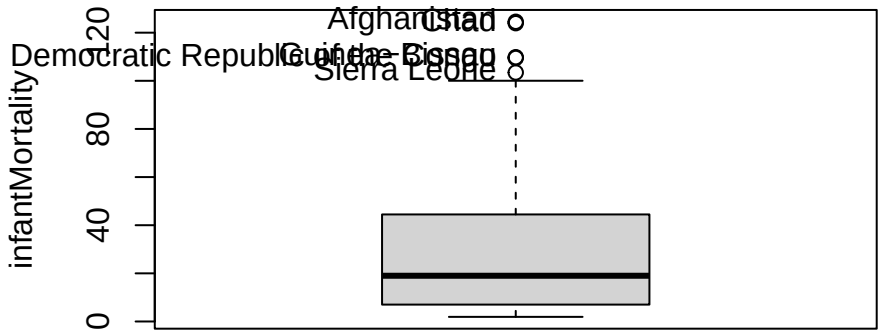
$$F_L = H_L - 1.5 \times H\text{-spread}$$
$$F_U = H_U + 1.5 \times H\text{-spread}$$

Observations beyond fences are identified as outliers. The fences themselves are not shown in the display. Points beyond $\pm 3 \times H$-spread are extreme outliers.

c. The 'whisker' growing from each end of the central box extends either to the extreme observation on its side of the distribution or to the most extreme non-outlying observation, called the 'adjacent value'

Boxplots are most useful as adjuncts to other displays (e.g., in the margins of a scatterplot) or for comparing several distributions.

```
Boxplot(~ infantMortality, data=UN)
```

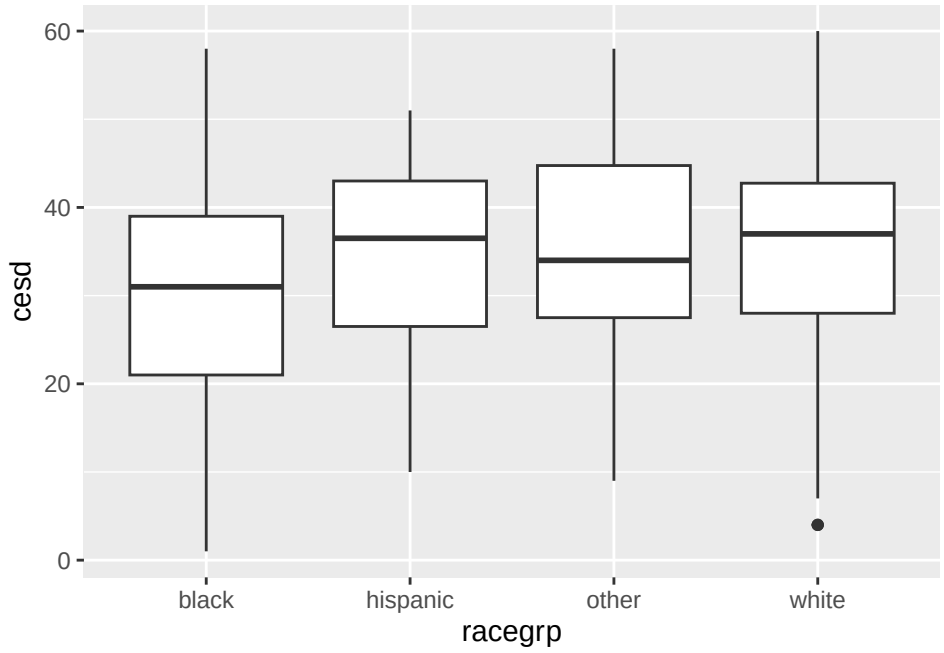



```
## [1] "Afghanistan"                         "Chad"
## [3] "Democratic Republic of the Congo" "Guinea-Bissau"
## [5] "Sierra Leone"
```

This plot was produced using the R base graphics. **ggplot** allows more flexibility and looks nicer.

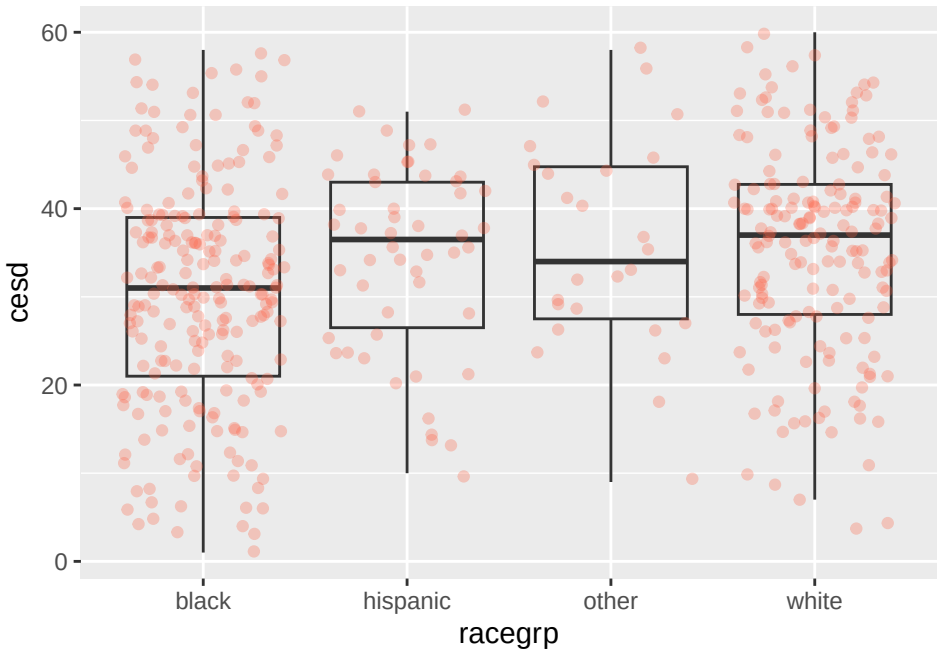
We can use boxplots to visualize the distribution of cesd by race group:

```
ggplot(data = helpdata, aes(x = racegrp, y = cesd)) +
    geom_boxplot()
```



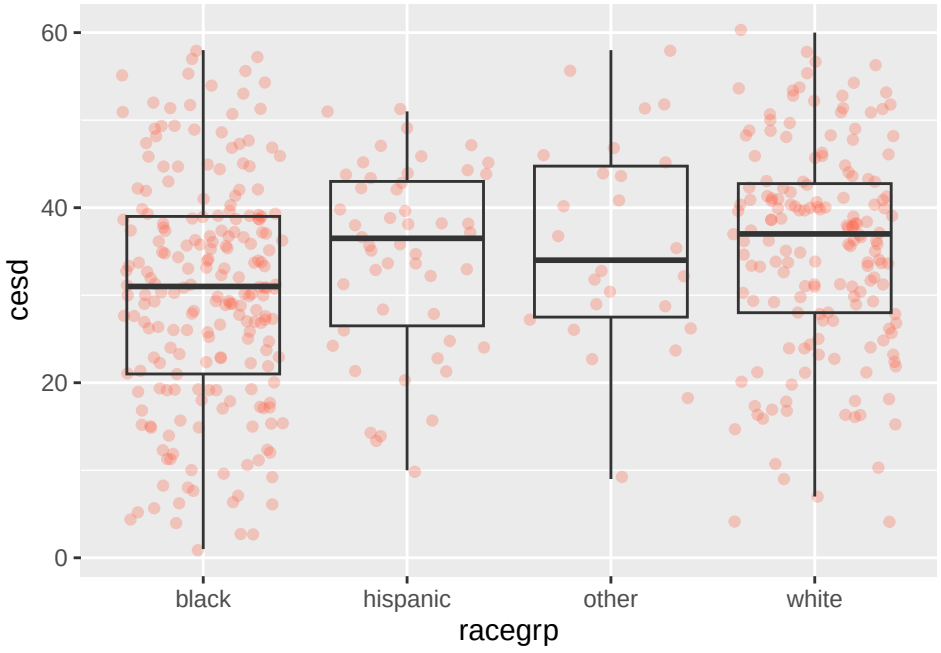
By adding points to boxplot, we can have a better idea of the number of measurements and of their distribution:

```
ggplot(data = helpdata, aes(x = racegrp, y = cesd)) +
    geom_boxplot(alpha = 0) +
    geom_jitter(alpha = 0.3, color = "tomato")
```



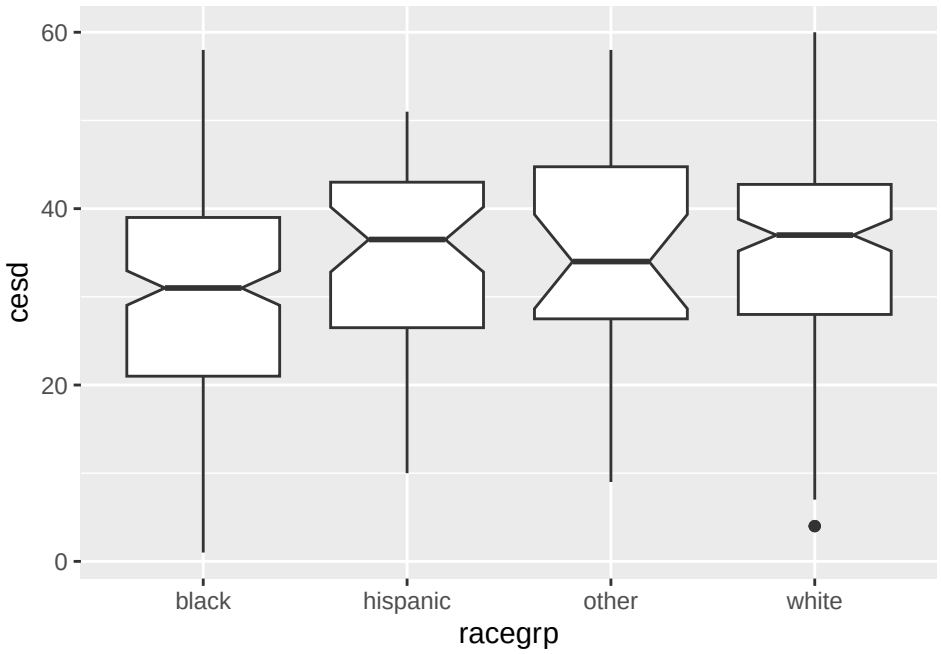
Notice how the boxplot layer is behind the jitter layer? What do you need to change in the code to put the boxplot in front of the points such that it's not hidden?

```
ggplot(data = helpdata, aes(x = racegrp, y = cesd)) +
    geom_jitter(alpha = 0.3, color = "tomato") +
    geom_boxplot(alpha = 0)
```



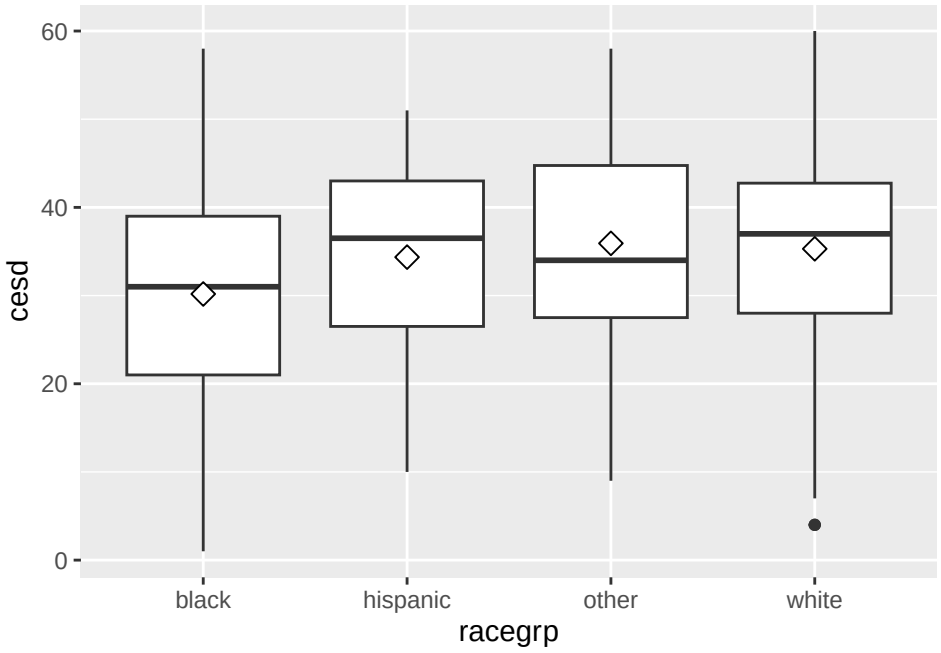
Another thing we could do is add notches to the boxplot to more easily visualize the median.

```
ggplot(data = helpdata, aes(x = racegrp, y = cesd)) +
    geom_boxplot(notch=TRUE)
```



You can also add the mean to the boxplot using `stat_summary`.

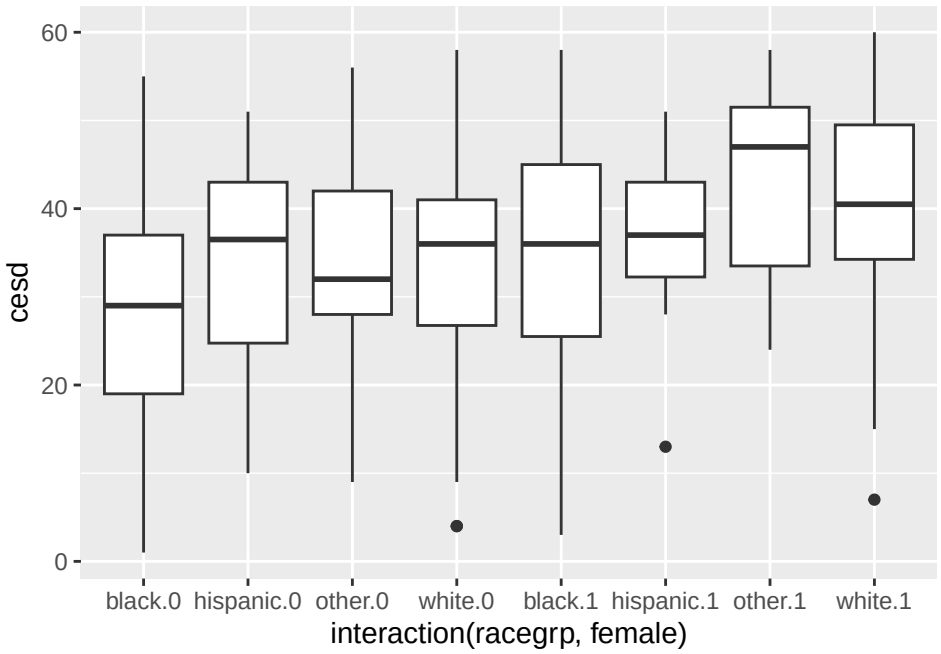
```
ggplot(data = helpdata, aes(x = racegrp, y = cesd)) +
    geom_boxplot() +
    stat_summary(fun="mean", geom="point", shape=23, size=3, fill="white")
```



Note the `shape=23`. There are all kinds of shapes available.

we can create box plots for multiple variables on the x-axis.

```
ggplot(data = helpdata, aes(x = interaction(racegrp, female), y = cesd)) +
    geom_boxplot()
```

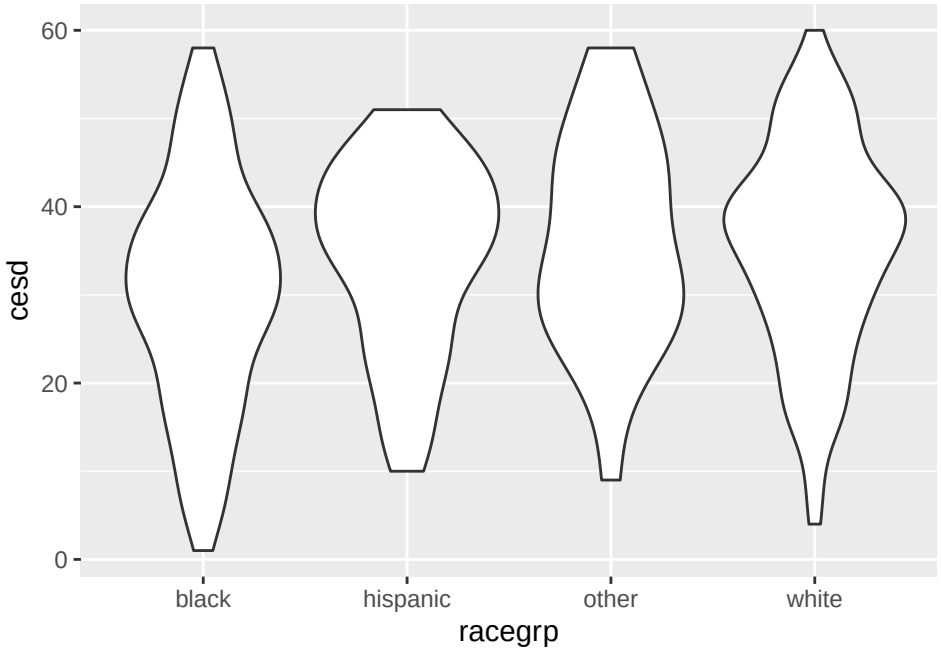


Violin plots

Boxplots are useful summaries, but hide the *shape* of the distribution. For example, if there is a bimodal distribution, it would not be observed with a boxplot. An alternative to the boxplot is the violin plot, where the shape (of the density of points) is drawn.

Replace the box plot with a violin plot; see `geom_violin()`.

```
ggplot(data = helpdata, aes(x = racegrp, y = cesd)) +
    geom_violin()
```

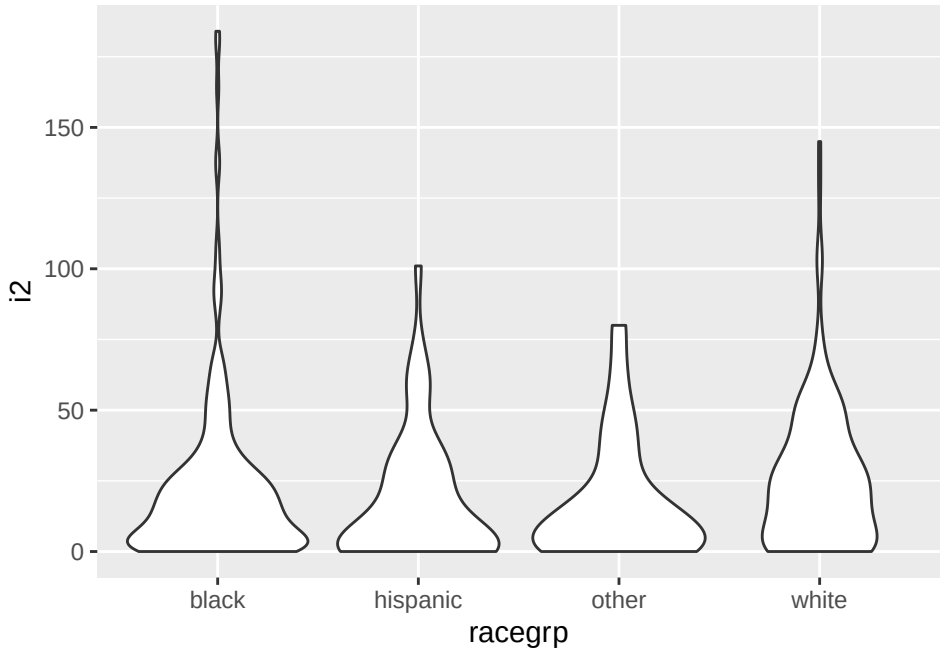


In many types of data, it is important to consider the *scale* of the observations. For example, it may be worth changing the scale of the axis to better distribute the observations in the space of the plot. Changing

the scale of the axes is done similarly to adding/modifying other components (i.e., by incrementally adding commands).

Let's look at another variable.

```
ggplot(data = helpdata, aes(x = racegrp, y = i2)) +
    geom_violin()
```

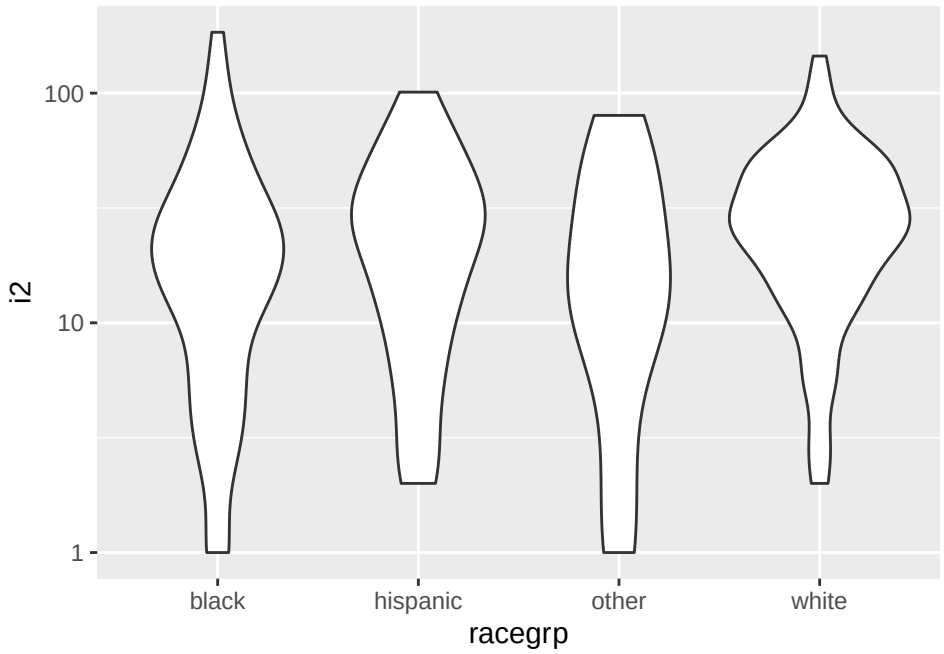


Wow! Ok. Maybe represent average drinks/day on the log10 scale.

```
ggplot(data = helpdata, aes(x = racegrp, y = i2)) +
    geom_violin() +
    scale_y_log10("i2")
```

```
## Warning: Transformation introduced infinite values in continuous y-axis
```

```
## Warning: Removed 87 rows containing non-finite values (`stat_ydensity()`).
```



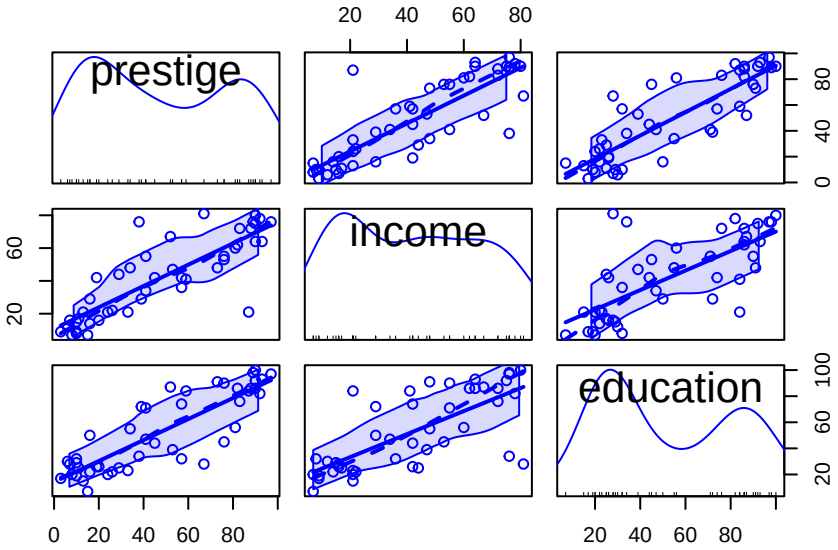
Plotting Multivariate Data

Because paper and computer screens are two-dimensional, graphical display of multivariate data is intrinsically difficult. Multivariate displays for quantitative data often project the higher dimensional 'point cloud' of the data onto a two-dimensional space. The essential trick of effective multidimensional display is to select projections that reveal important characteristics of the data. In certain circumstances, projections can be selected on the basis of a statistical model fit to the data or on the basis of explicitly stated criteria. A simple approach to multivariate data, which does not require a statistical model, is to examine bivariate scatterplots for all pairs of variables. Arraying these plots in a 'scatterplot matrix' produces a graphical analog to the correlation matrix.

However, by projecting the multidimensional point cloud onto pairs of axes, the plot focuses on the marginal relationships between the corresponding pairs of variables. The object of data analysis for several variables is typically to investigate partial relationships, not marginal associations. Y can be related marginally to X even when there is no partial relationship between the two variables controlling for other X's. It is also possible for there to be a partial association between Y and X but no marginal association. Furthermore, if the X's themselves are nonlinearly related, then the marginal relationship between Y and a specific X can be nonlinear even when their partial relationship is linear.

We can create a scatterplot matrix is using the **car** package.

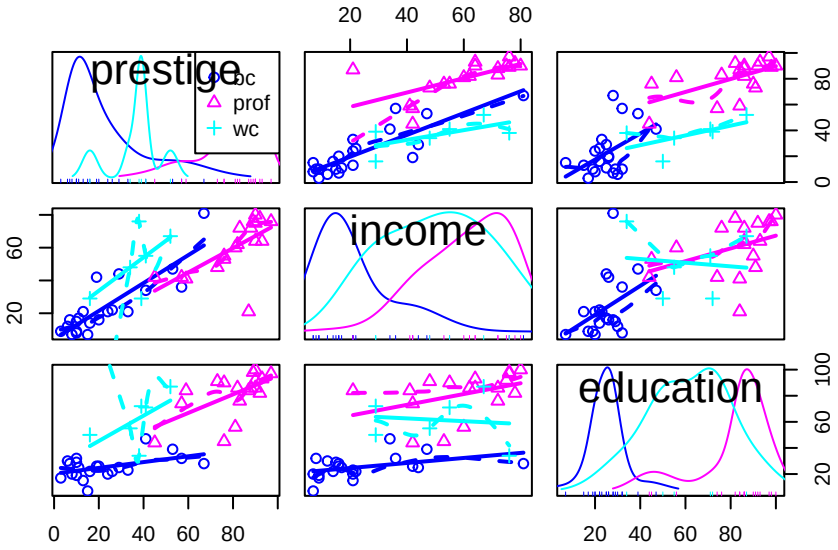
```
scatterplotMatrix(~ prestige + income + education, data=Duncan)
```



Information about a categorical third variable may be entered on a bivariate scatterplot by coding the plotting symbols. The most effective codes use different colors to represent categories, but degrees of fill, distinguishable shapes, and distinguishable letters can also be effective.

```
scatterplotMatrix(~ prestige + income + education | type,
    diag="oned", data=Duncan)
```

```
## Warning in applyDefaults(diagonal, defaults = list(method = "adaptiveDensity"),
## : unnamed diag arguments, will be ignored
```



The `scatterplotMatrix()` function in the **car** package can also do the following:
* Condition the scatter plot matrix on a factor.
* Include linear and loess fit lines.
* Place box plots, densities, or histograms in the principal diagonal.
* Add rug plots in the margins of the cells.

Correlation matrix

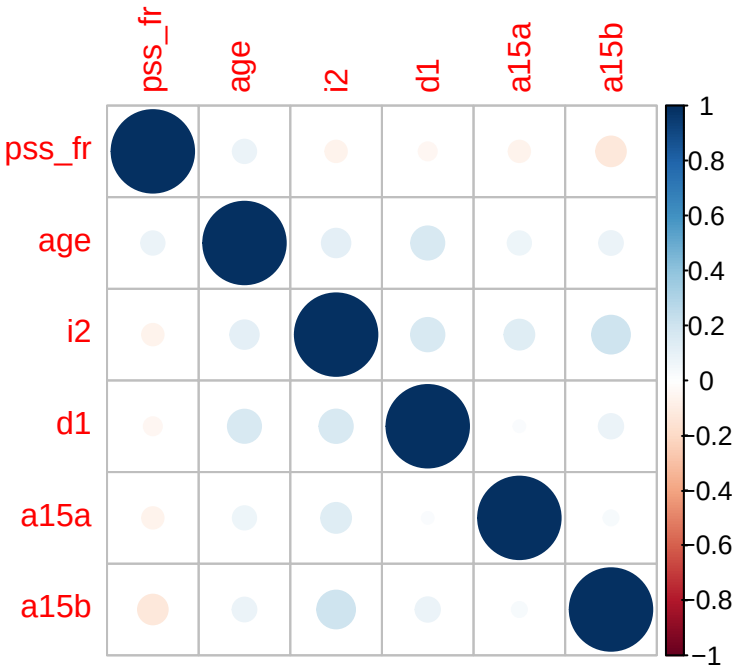
We can construct a correlation matrix of a subset of the HELP variables,

```
contvars <- dplyr::select(helpdata, pss_fr, age, i2, d1, a15a, a15b)
corrmat <- cor(contvars)
corrmat
```

```
##             pss_fr        age          i2          d1        a15a        a15b
## pss_fr  1.00000000 0.08014973 -0.06754741 -0.04779948 -0.06720626 -0.12919889
## age     0.08014973 1.00000000  0.11916482  0.16135030  0.07970404  0.08313623
## i2     -0.06754741 0.11916482  1.00000000  0.16338952  0.13108352  0.20979226
## d1     -0.04779948 0.16135030  0.16338952  1.00000000  0.02036584  0.08735889
## a15a   -0.06720626 0.07970404  0.13108352  0.02036584  1.00000000  0.03314848
## a15b   -0.12919889 0.08313623  0.20979226  0.08735889  0.03314848  1.00000000
```

and visualize this plot using the `corrplot` package:

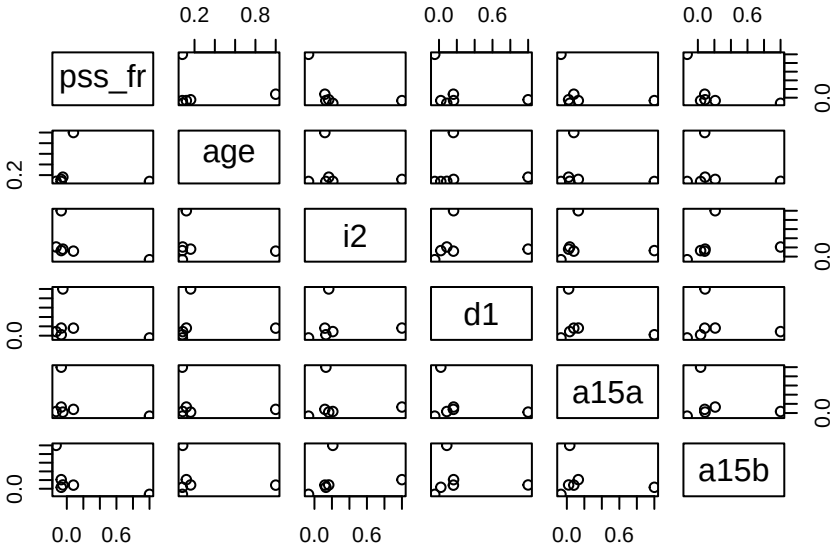
```
library(corrplot)
corrplot(corrmat)
```



This package has all kinds of additional options.

You can also create a scatter plot matrix using **base graphics** but unlike the function in the `car` package, you need to first create a correlation matrix of the variables as we did for the `corrplot` package.

```
pairs(corrmat)
```



Mosaic plots

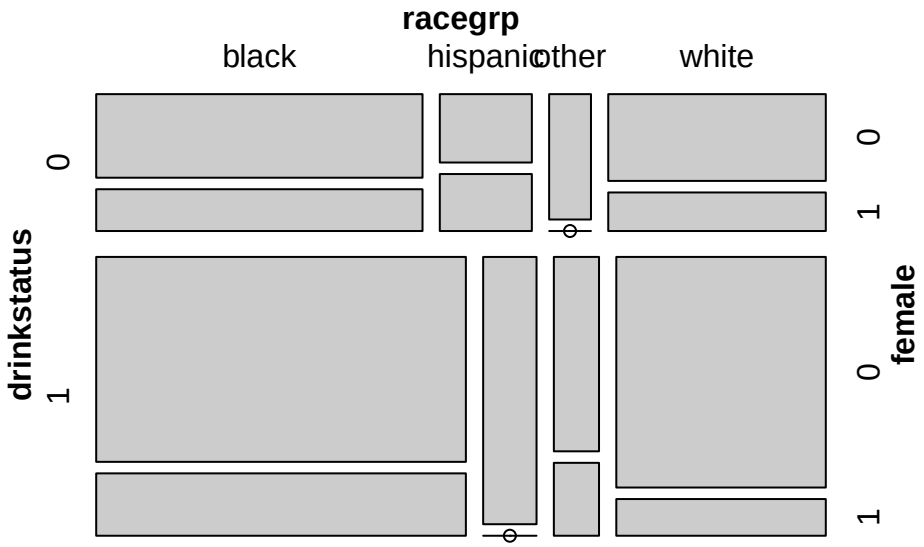
Mosaic plots are good way to visualize contingency tables. Construct a contingency table:

```
catvars <- dplyr::select(helpdata, drinkstatus, female, racegrp)
ftable(catvars)
```

```
##                     racegrp black hispanic other white
## drinkstatus female
## 0           0                  26        6     5    18
##             1                  13        5     0     8
## 1           0                  69       13     8    44
##             1                  21        0     3     7
```

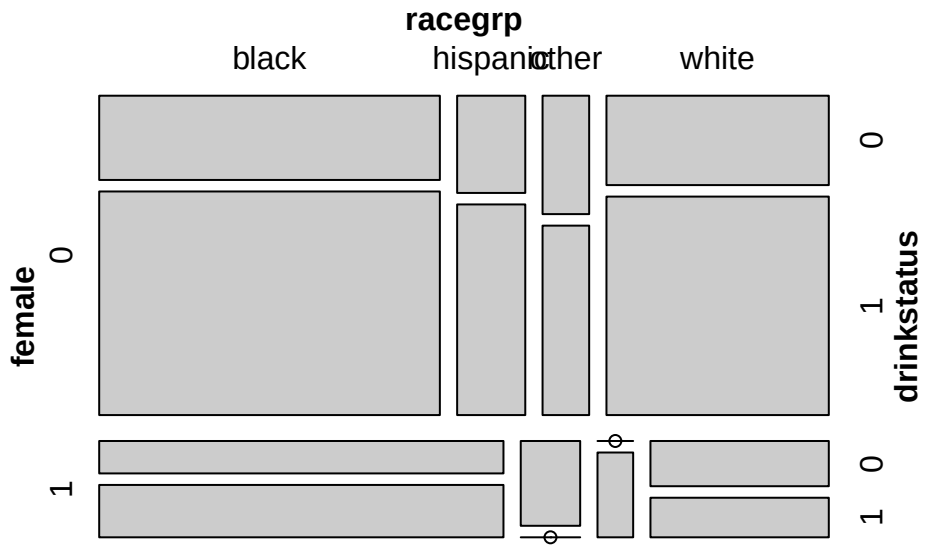
You can visualize this table using the `mosaic()` function from the `vcd` package.

```
library(vcd)
mosaic(~ drinkstatus + racegrp + female, data=helpdata)
```

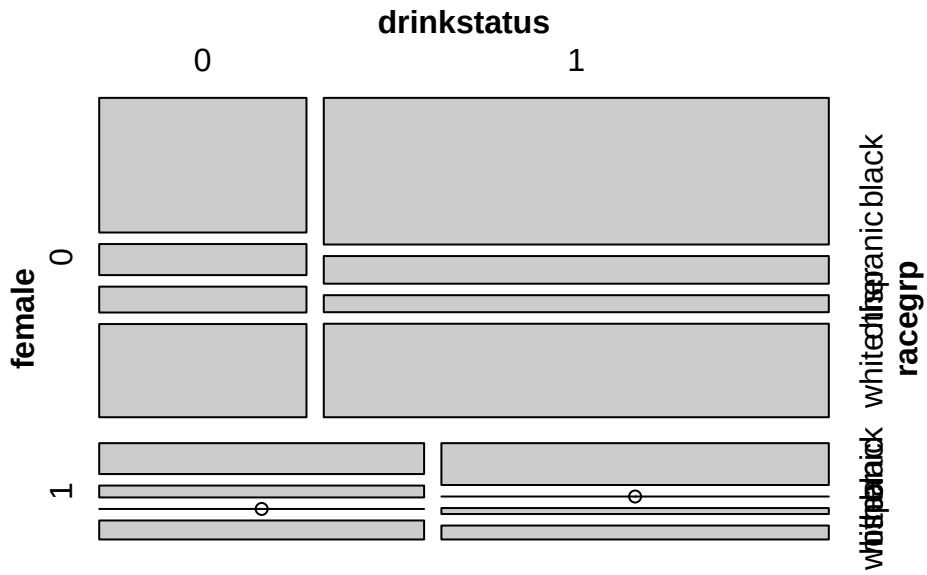


The order in which the variables are entered into the function determines the presentation.

```
mosaic(~ female + racegrp + drinkstatus, data=helpdata)
```

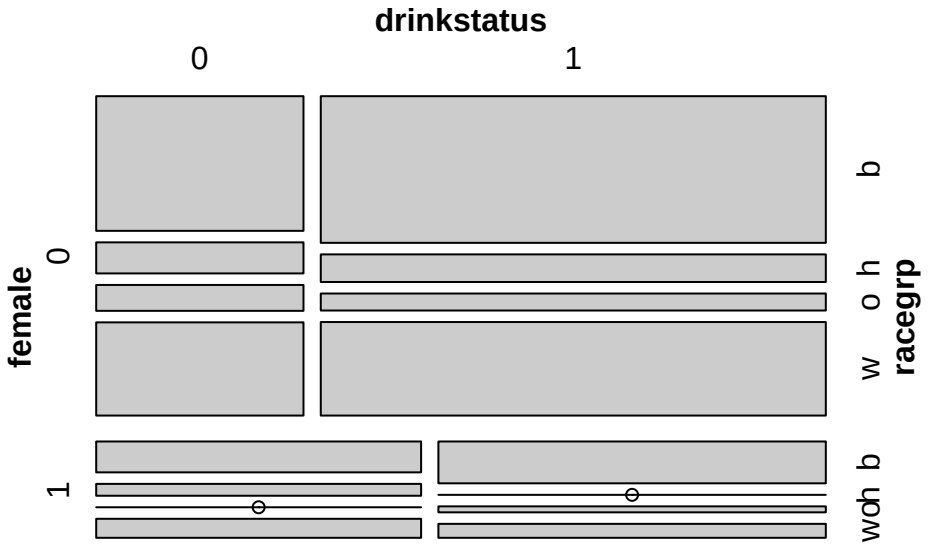



```
mosaic(~ female + drinkstatus + racegrp, data=helpdata)
```



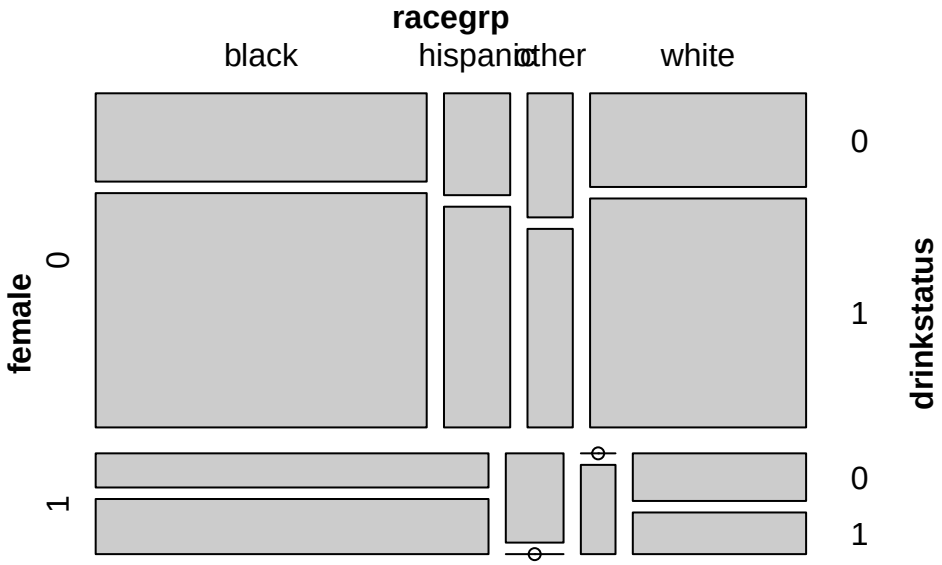
We have a problem with overlapping labels. Here is one way to deal with it.

```
mosaic(~ female + drinkstatus + racegrp, data=helpdata, labeling_args =
        list(abbreviate_labs = c(racegrp = TRUE, drinkstat=TRUE)))
```



Here is another.

```
mosaic(~ female + racegrp + drinkstatus, data=helpdata, labeling_args =
         list(rot_labels = c(bottom = 90, right = 0),
              offset_varnames = c(right = 1), offset_labels = c(right = 0.3)),
              margins = c(right = 4, bottom = 3))
```



The `vcd` package, which is an abbreviation for visualizing categorical data, is really useful if you are working a lot with categorical data.

Visualizing Densities

The most simplistic plot for visualizing distributions that you should have learned in introductory biostatistics is stem and leaf plots. `ggplot` does not create them (to my knowledge) so we will use **base graphics**.

Stem and leaf plot

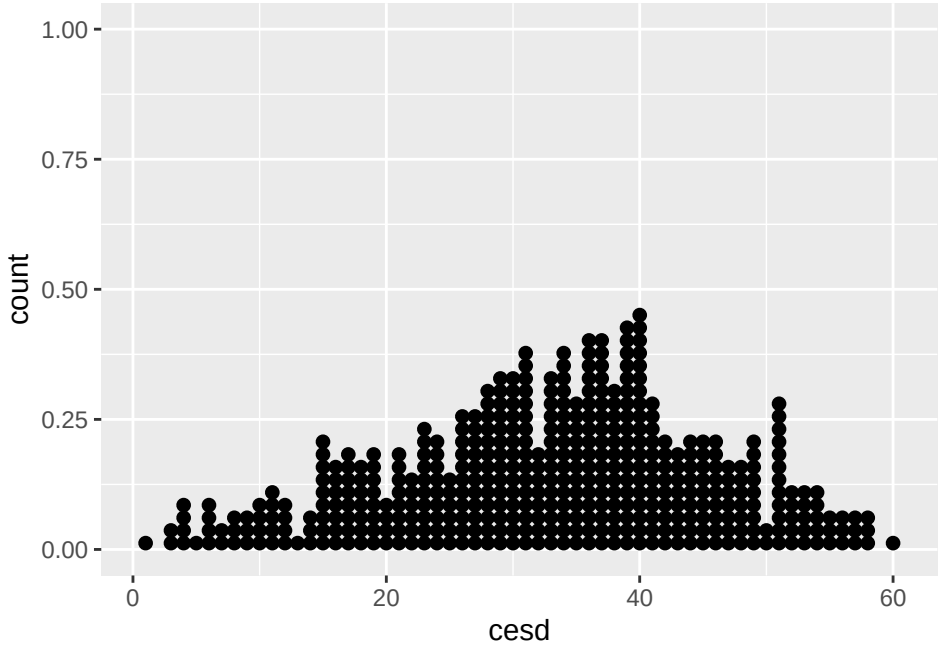
```
stem(helpdata$cesd)
```

```
##
##   The decimal point is 1 digit(s) to the right of the |
##
##   0 | 1334444
##   0 | 5666677888999
##   1 | 00001111122223444
##   1 | 55555555566666667777777788888899999999
##   2 | 000011111111222222333333333444444444
##   2 | 55555566666666666777777777778888888888888999999999999999
##   3 | 00000000000000111111111111112222222233333333333333344444444444444444
##   3 | 55555555555566666666666666666777777777777777777888888888888999999999
##   4 | 00000000000000000001111111111122222222333333333444444444
##   4 | 55555555566666666677777778888888999999999
##   5 | 00111111111111222223333344444
##   5 | 555666777888
##   6 | 0
```

Dot plots

Also known as a Wilkinson dot plot. Use `geom_dotplot()`

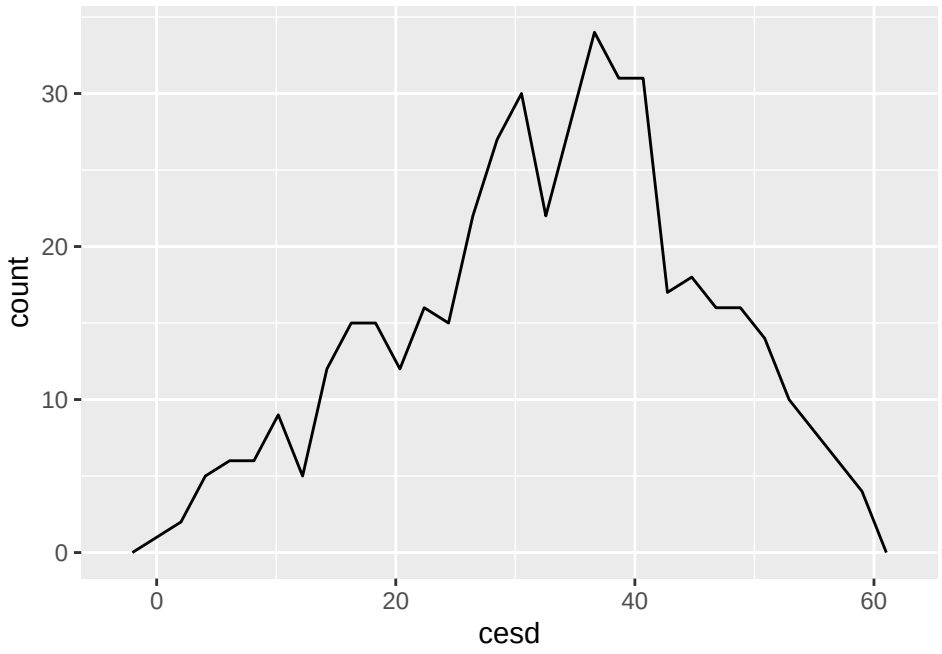
```
ggplot(data = helpdata, aes(x = cesd)) +
  geom_dotplot(binwidth = 1)
```



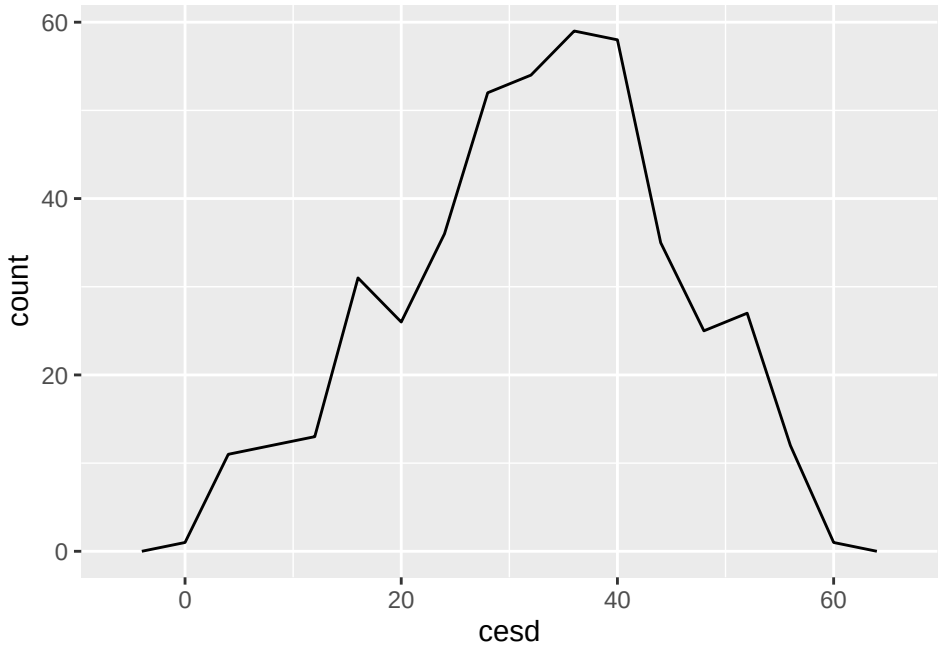
Frequency polygon

```
ggplot(data = helpdata, aes(x = cesd)) +
  geom_freqpoly()
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```

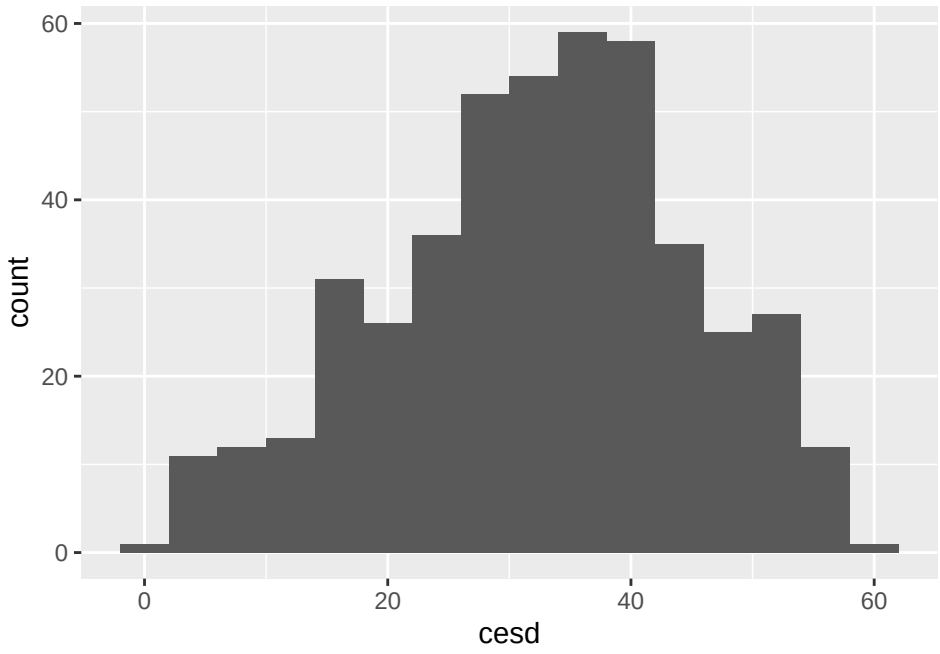



```
ggplot(data = helpdata, aes(x = cesd)) +
  geom_freqpoly(binwidth = 4)
```



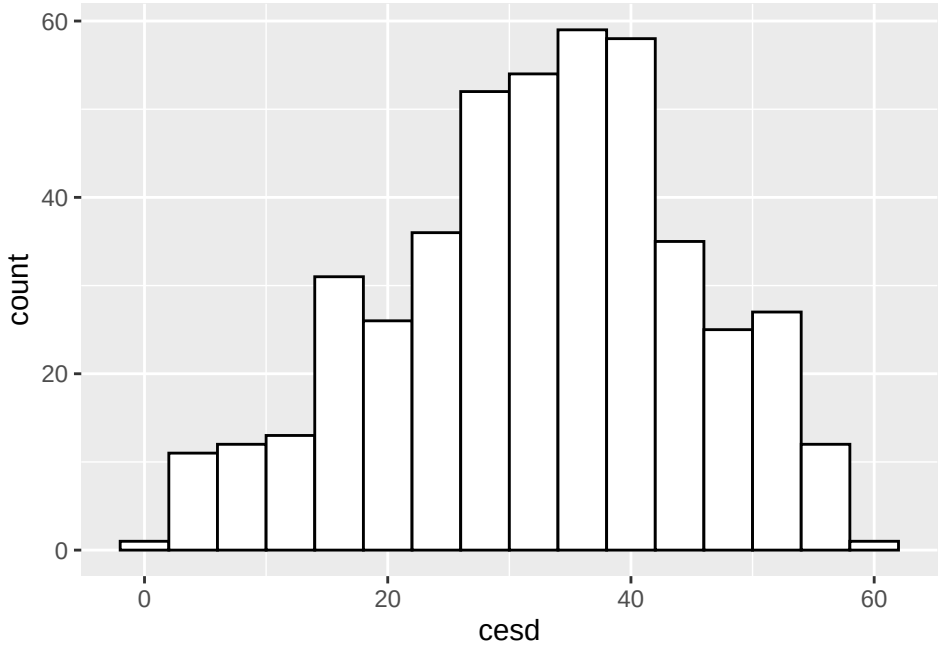
Histograms

```
ggplot(data = helpdata, aes(x = cesd)) +
  geom_histogram(binwidth = 4)
```



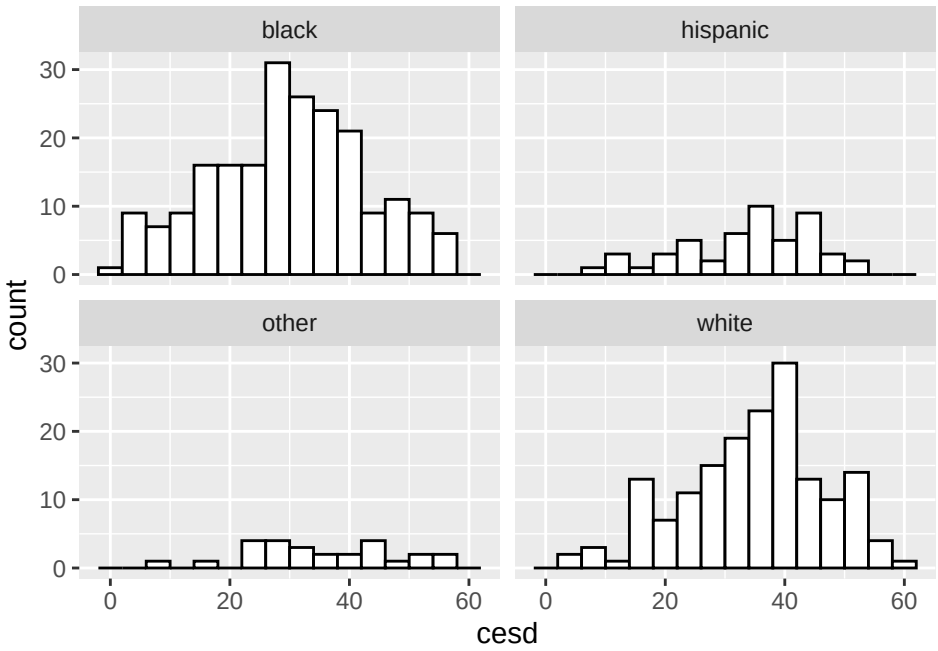
This dark fill without an outline is difficult to read. Let's try a white fill with a black outline.

```
ggplot(data = helpdata, aes(x = cesd)) +
  geom_histogram(binwidth = 4, fill="white", colour="black")
```



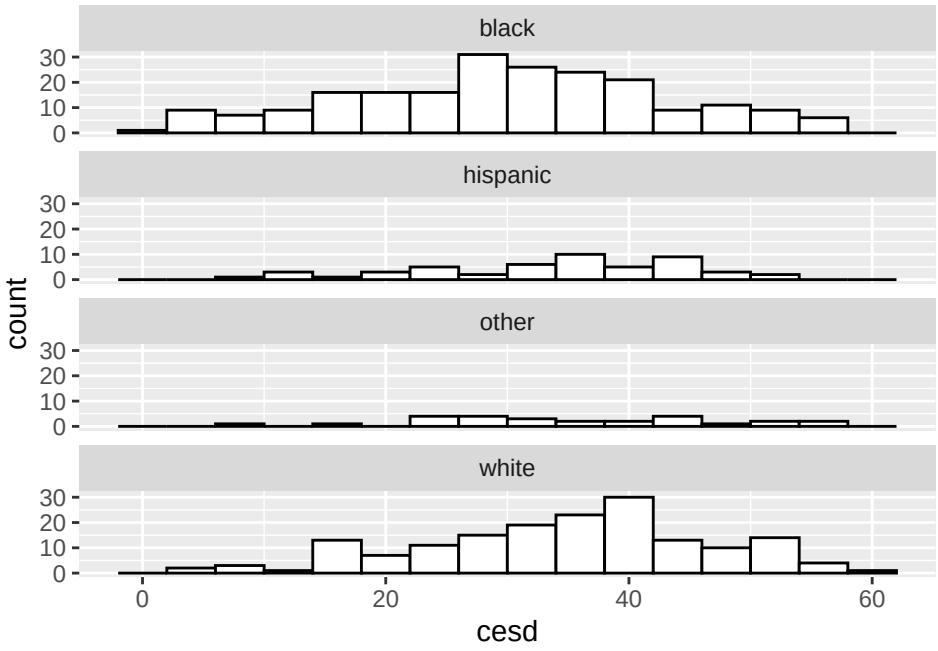
Better! Now, maybe we want to see the distribution of CESD by race group. `ggplot` has a special technique called *faceting* that allows the user to split one plot into multiple plots based on a factor included in the dataset. `facet_wrap` extracts plots into an arbitrary number of dimensions to allow them to cleanly fit on one page. `facet_wrap()` wraps around like words on a page.

```
ggplot(data = helpdata, aes(x = cesd)) +
  geom_histogram(binwidth = 4, fill="white", colour="black") +
  facet_wrap( ~ racegrp)
```



You can specify a number of row or columns with `facet_wrap()`.

```
ggplot(data = helpdata, aes(x = cesd)) +
  geom_histogram(binwidth = 4, fill="white", colour="black") +
  facet_wrap( ~ racegrp, nrow=4)
```

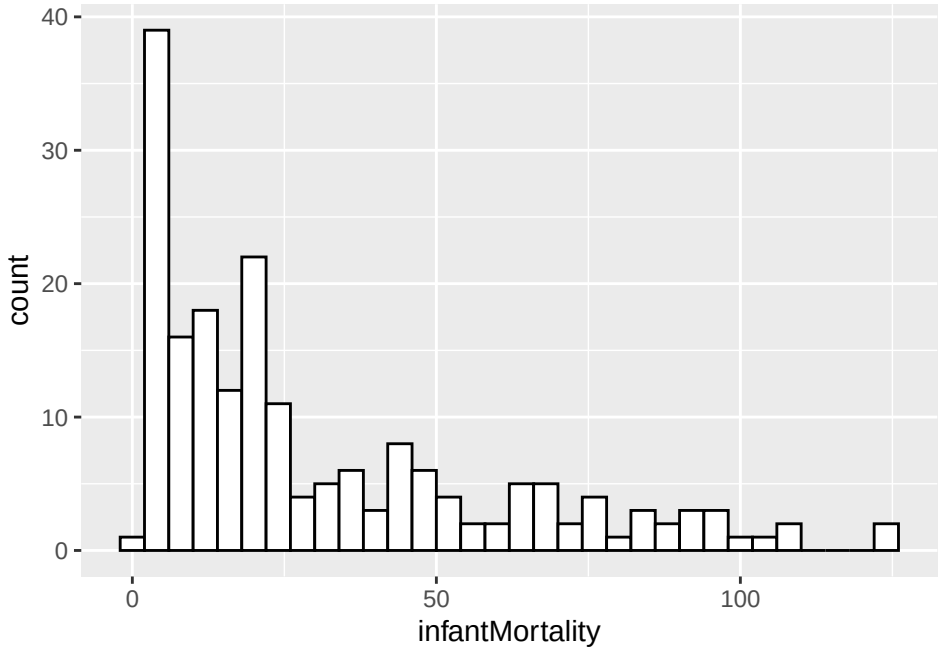


Histograms have several disadvantages.

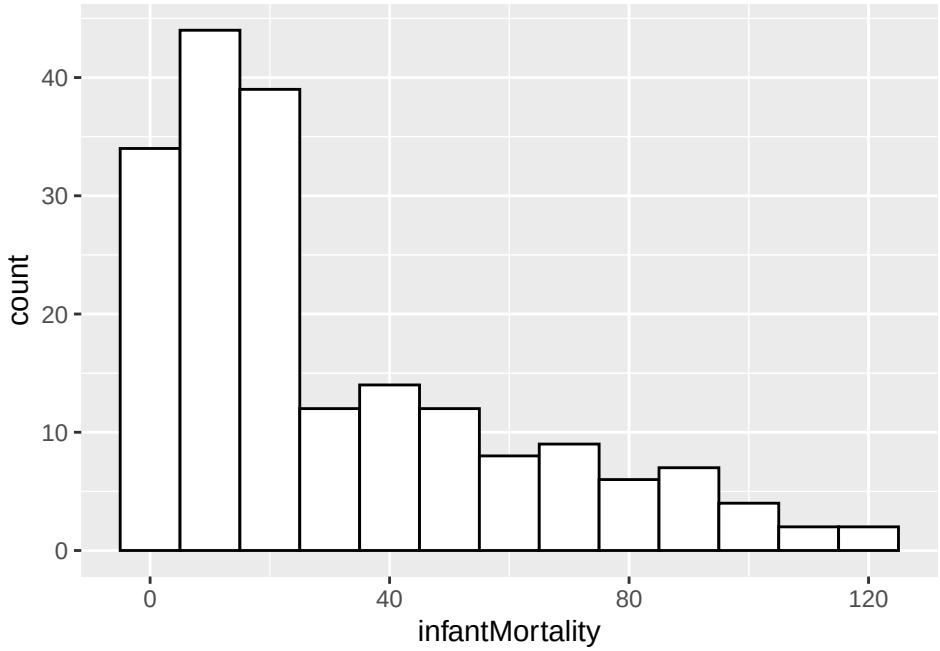
- The visual impression of the data conveyed by a histogram can depend upon the arbitrary origin of the bin system.
- Because the bin system dissects the range of the variable into class intervals, the histogram is discontinuous (i.e., rough) even if the variable is continuous.
- The form of the histogram depends upon the arbitrary width of the bins.

- If we use bins that are narrow enough to capture detail where data are plentiful — usually near the center of the distribution — then they may be too narrow to avoid 'noise' where data are sparse — usually in the tails of the distribution.

```
UN <- na.omit(UN)  # filter out missing data
ggplot(data = UN, aes(x = infantMortality)) +
  geom_histogram(binwidth = 4, fill="white", colour="black")
```




```
ggplot(data = UN, aes(x = infantMortality)) +
  geom_histogram(binwidth = 10, fill="white", colour="black")
```

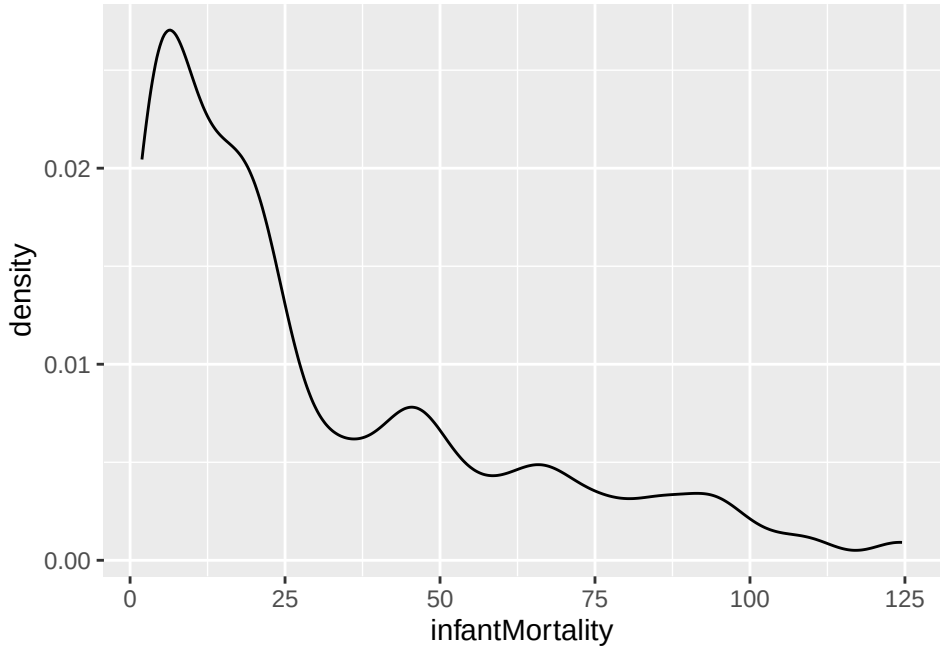


Density plots

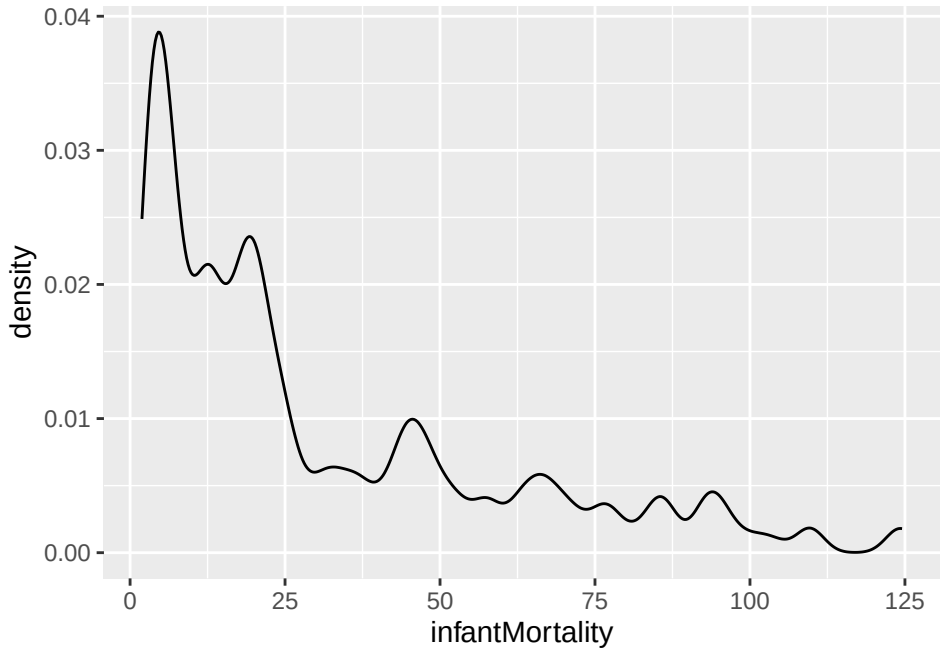
Kernel density plots look similar to frequency polygons but frequency polygons are your actual data, whereas density plots are based on estimates, specifically, kernel density estimates, which you may or may not have been exposed to in your previous classes.

Nonparametric density estimation addresses the deficiencies of traditional histograms by averaging and smoothing. For example:

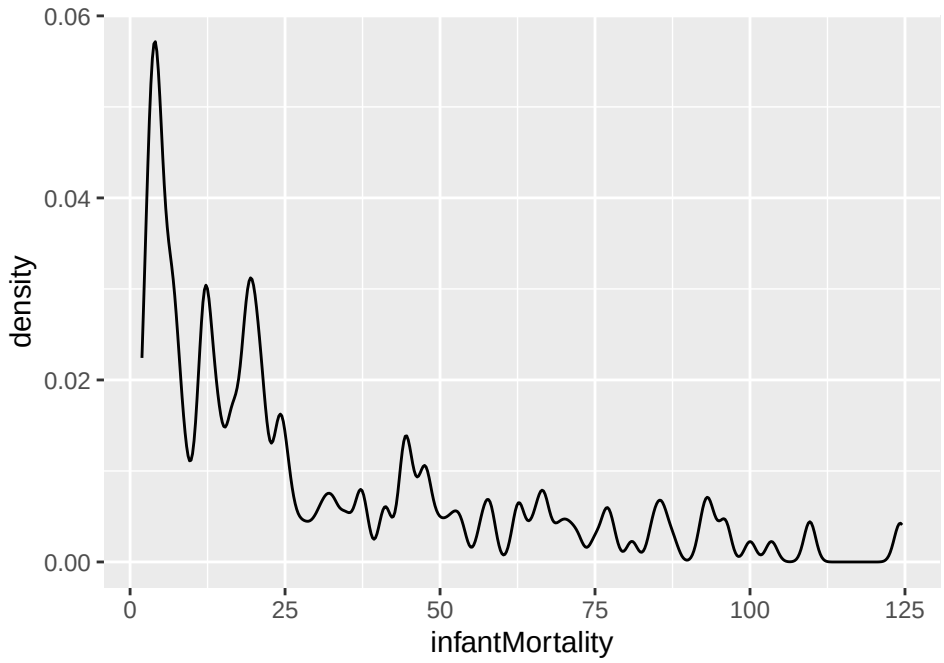
```
ggplot(data = UN, aes(x = infantMortality)) +
   geom_line(stat="density", adjust=.5)
```




```
ggplot(data = UN, aes(x = infantMortality)) +
   geom_line(stat="density", adjust=.25)
```



```
ggplot(data = UN, aes(x = infantMortality)) +
   geom_line(stat="density", adjust=.1)
```



The kernel density estimator continuously moves a window of fixed width across the data, calculating a locally weighted average of the number of observations falling in the window — a kind of running proportion.

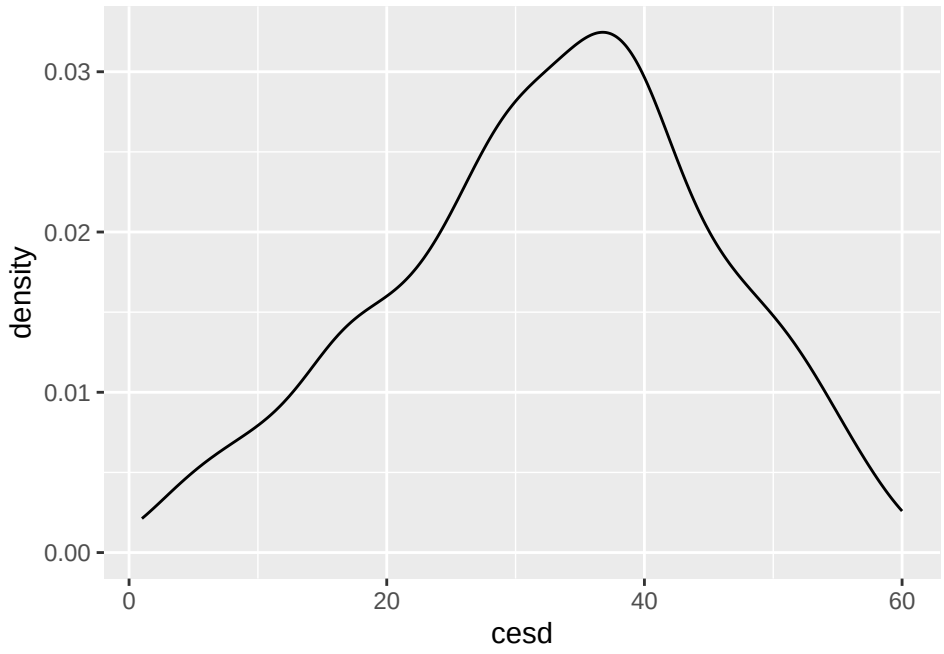
The smoothed plot is scaled so that it encloses an area of one.

Selecting the window width for the kernel estimator is primarily a matter of trial and error — we want a value small enough to reveal detail but large enough to suppress random noise.

The adaptive kernel estimator is similar, except that the window width is adjusted so that the window is narrower where data are plentiful and wider where data are sparse.

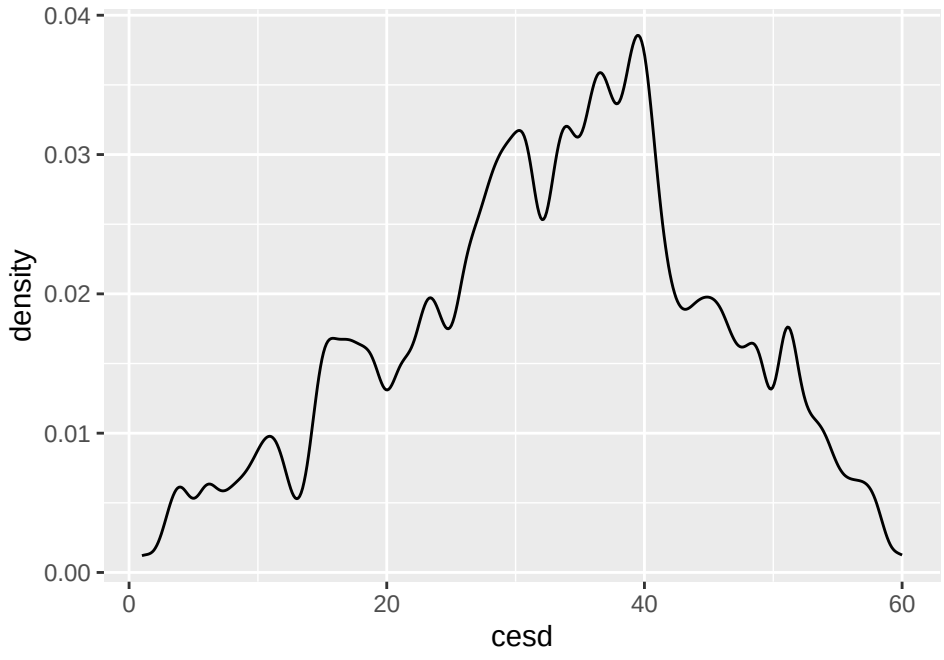
Returning to the HELP data:

```
ggplot(data = helpdata, aes(x = cesd)) +
  geom_density()
```



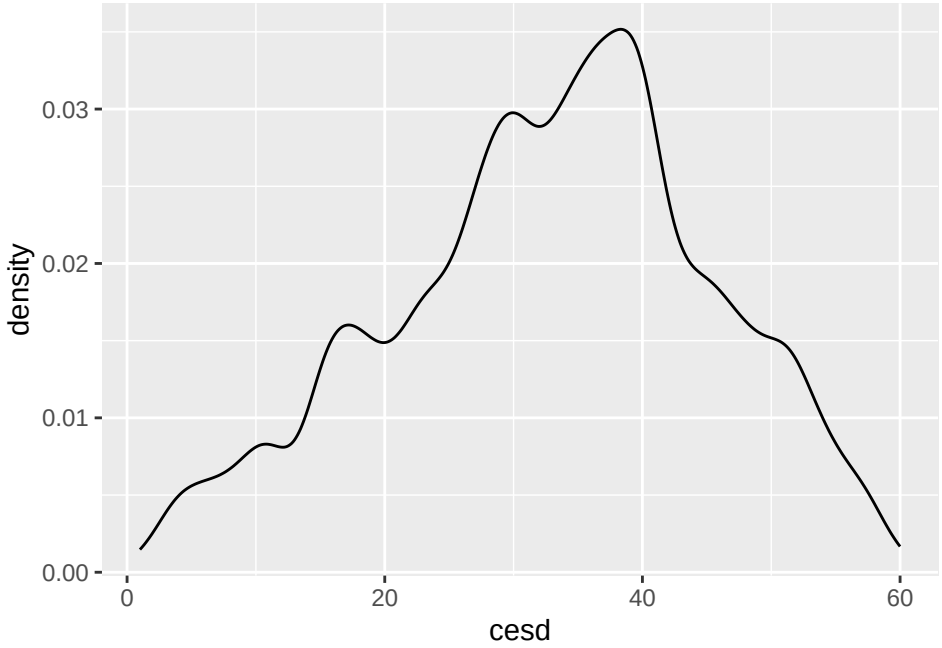
That's a bit over-smoothed. In order to control the smoothing, we have to change it to `geom_line(stat="density")` (instead of `geom_density`) and then use `adjust=`.

```
ggplot(data = helpdata, aes(x = cesd)) +
  geom_line(stat="density", adjust=.25)
```



Perhaps that was too much of adjustment. The `geom_density()` uses a default of `adjust=1` so we probably want something in between .25 and 1.

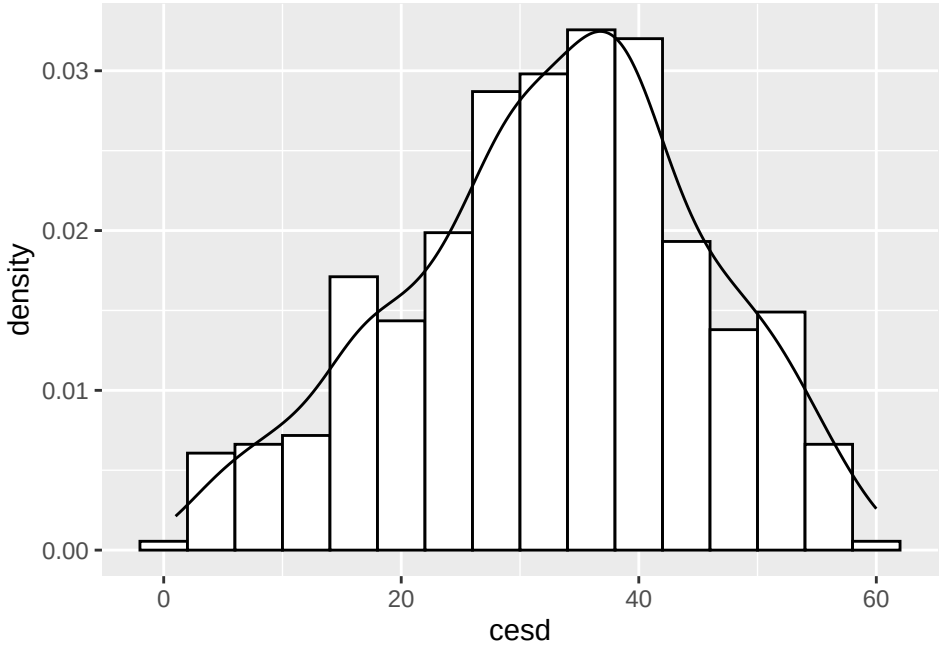
```
ggplot(data = helpdata, aes(x = cesd)) +
  geom_line(stat="density", adjust=.5)
```



Maybe we want to overlay this on our histogram. This is an example of layering. You need to add `y=..density..` and an additional layer.

```
ggplot(data = helpdata, aes(x = cesd, y=..density..)) +
  geom_histogram(binwidth = 4, fill="white", colour="black") +
  geom_density()
```

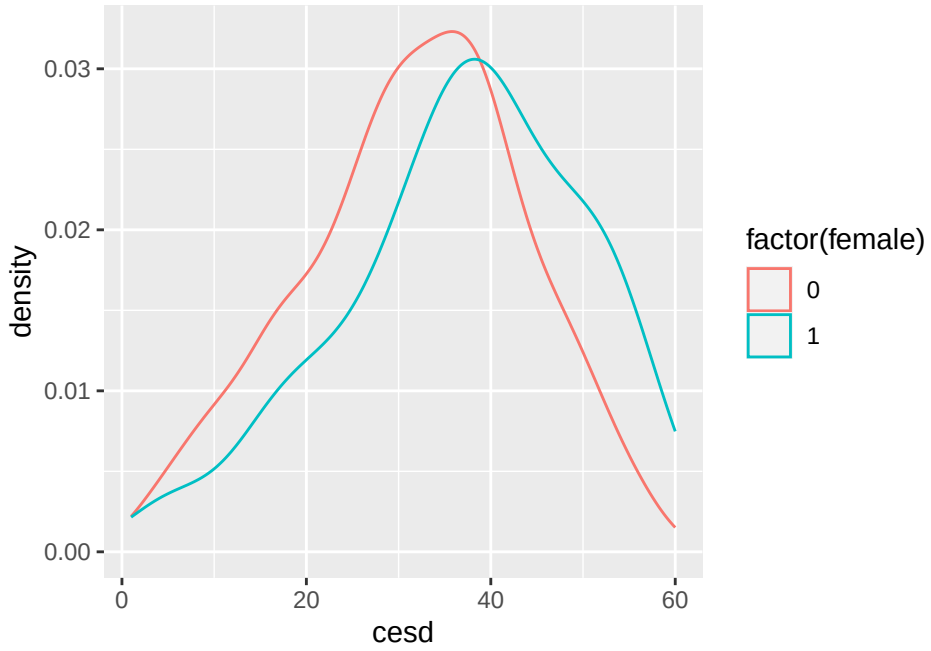
```
## Warning: The dot-dot notation (`..density..`) was deprecated in ggplot2 3.4.0.
## i Please use `after_stat(density)` instead.
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_lifecycle_warnings()` to see where this warning was
## generated.
```



Perhaps we would like to see the densities by gender. Note, the variable assigned to colors needs to be of

type `factor`.

```
ggplot(data = helpdata, aes(x = cesd, color = factor(female))) +
  geom_density()
```

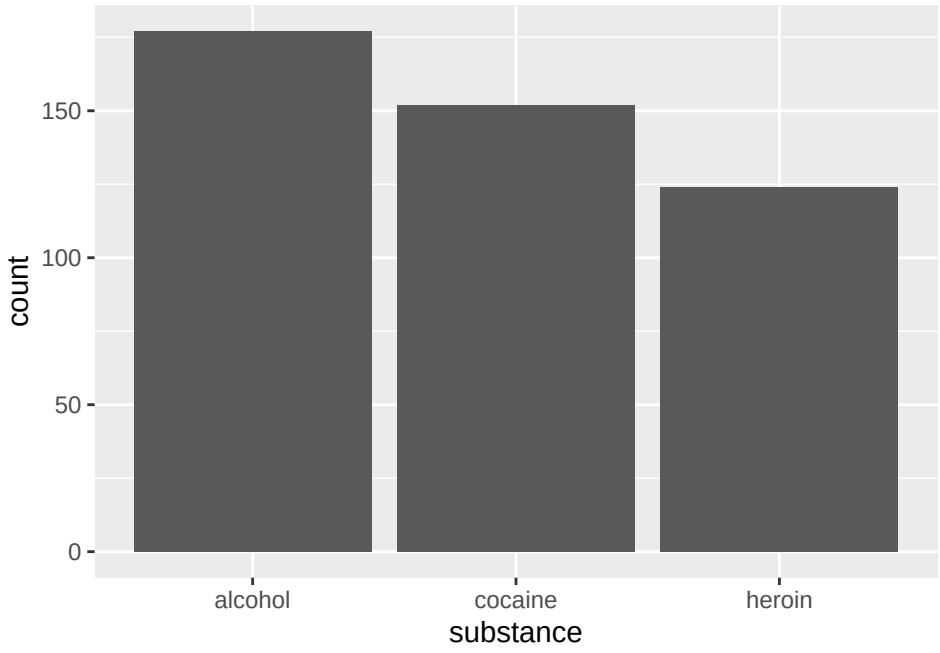


Probably oversmoothed again but I will leave that to you to adjust.

Bar graphs

There are two types of bar charts: `geom_bar()` makes the height of the bar proportional to the number of cases in each group.

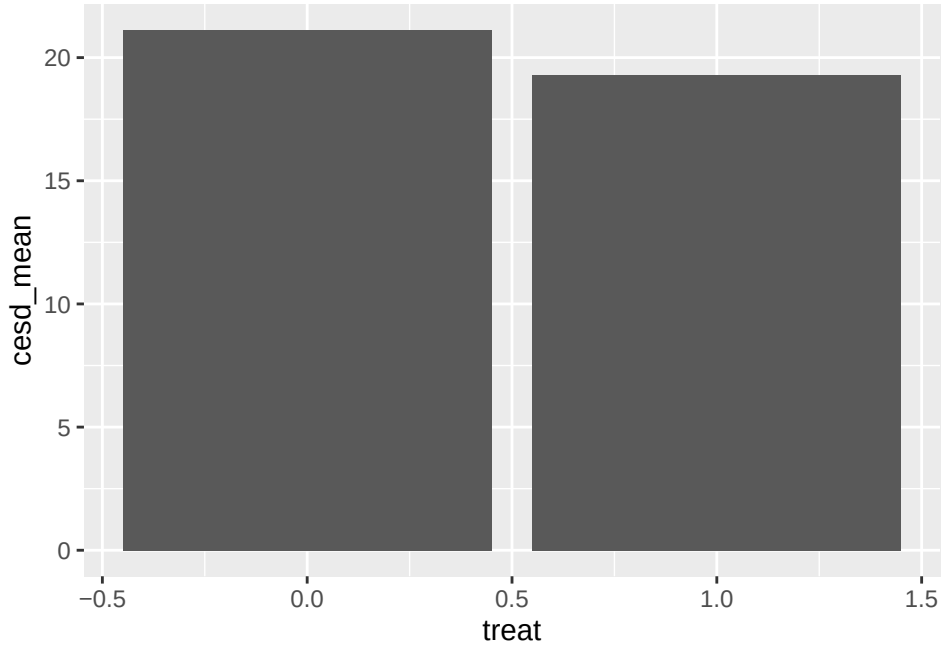
```
ggplot(data = helpdata, aes(x = substance)) +
  geom_bar()
```



If you want the heights of the bars to represent, say the mean of some variable, use `geom_col()` instead. First though, you have to create a data set that has the mean levels by the grouping variable, in this case `treat`.

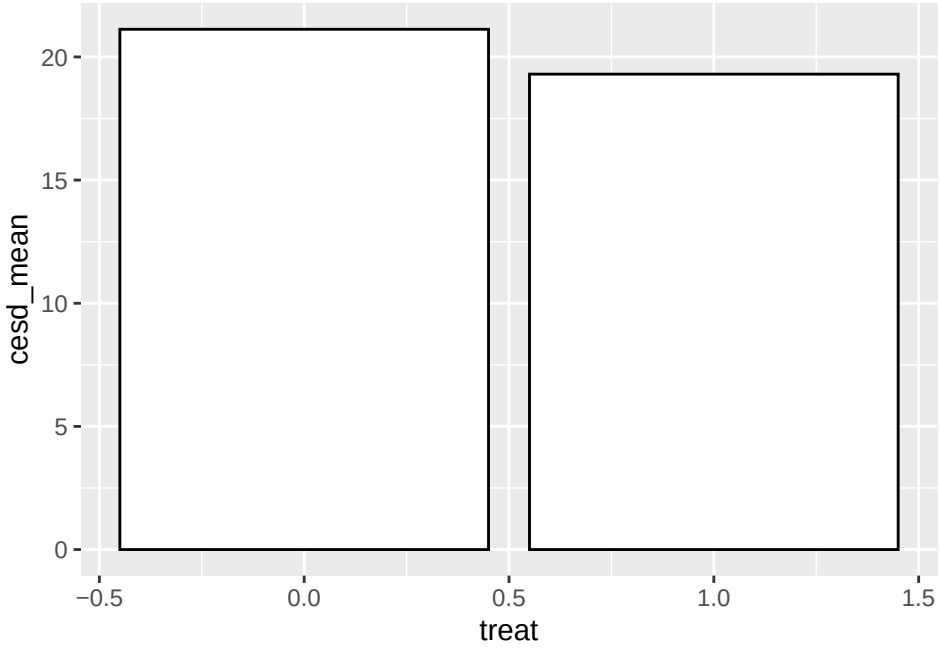
```
help_agg <- helpdata %>%
  group_by(treat) %>%
  summarise(cesd_mean = mean(cesd4, na.rm=TRUE))
```

```
ggplot(data = help_agg, aes(x = treat, y = cesd_mean)) +
  geom_col()
```



If you want to change the color fill of the bars, use `fill=` and to change the color of the outline around the boxes, use `color=`.

```
ggplot(data = help_agg, aes(x = treat, y = cesd_mean)) +
  geom_col(fill="white", color="black")
```

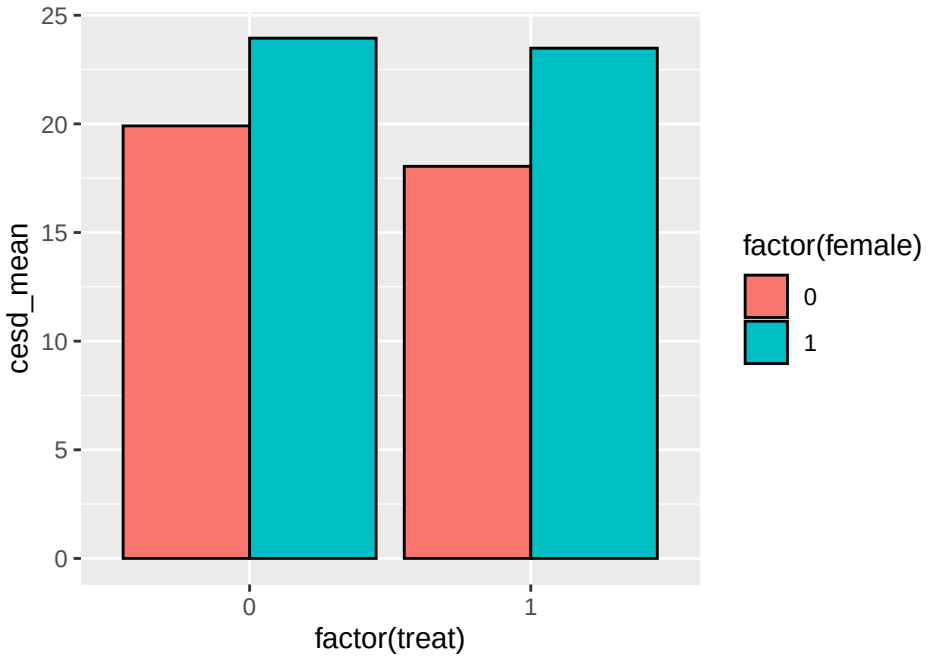


We can include another variable as a factor. We can also convert `treat` to a factor. Notice what happens to the x-axis.

```
help_agg <- helpdata %>%
  group_by(treat, female) %>%
  summarise(cesd_mean = mean(cesd4, na.rm=TRUE))
```

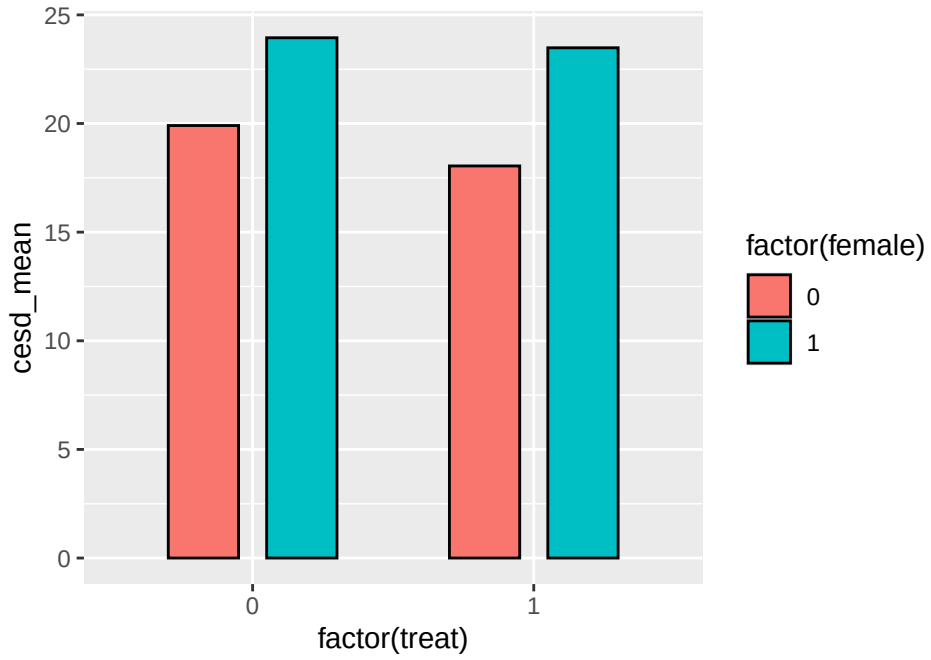
```
## `summarise()` has grouped output by 'treat'. You can override using the
## `.groups` argument.
```

```
ggplot(data = help_agg, aes(x = factor(treat), y = cesd_mean,
                            fill = factor(female))) +
  geom_col(position="dodge", color="black")
```



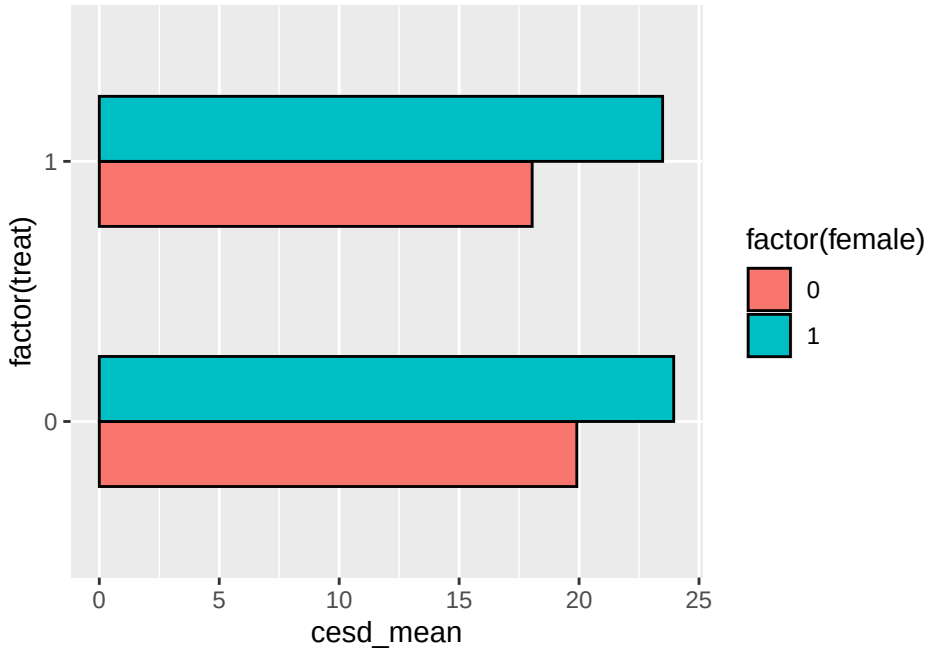
You can adjust the bar width and spacing. The default value of `width` is 0.9.

```
ggplot(data = help_agg, aes(x = factor(treat), y = cesd_mean,
                            fill = factor(female))) +
  geom_col(position=position_dodge(0.7), color="black", width=0.5)
```



Finally, you can swap the axes for any plot using `coord_flip()`. In this case, resulting in a horizontal bar graph.

```
ggplot(data = help_agg, aes(x = factor(treat), y = cesd_mean,
                            fill = factor(female))) +
  geom_col(position="dodge", color="black", width=0.5) +
  coord_flip()
```



QQ plots

Quantile-comparison plots are useful for comparing an empirical sample distribution with a theoretical distribution, such as the normal distribution. A strength of the display is that it does not require the use of arbitrary bins or windows.

Let $P(x)$ represent the theoretical cumulative distribution function (CDF) to which we wish to compare the data.

1. Order the data values from smallest to largest, denoted $X_{(1)}, X_{(2)}, ..., X_{(n)}$. The $X_{(i)}$ are called the order statistics.
2. The cumulative proportion of the data 'below' $X_{(i)}$ is given by

$$P_i = \frac{i - 1/2}{n}$$

3. Use the inverse of the CDF (the quantile function) to find the value z_i corresponding to the cumulative probability P_i:

$$z_i = P^{-1}\left(\frac{i - 1/2}{n}\right)$$

4. Plot the z_i as horizontal coordinates against the $X_{(i)}$ as vertical coordinates.
5. It is often helpful to place a comparison line on the plot to facilitate the perception of departures from linearity/ normality.
6. We expect some departure from normality because of sampling variation; it therefore assists interpretation to display the expected degree of sampling error in the plot. The standard error of the order statistic is:

$$SE(X_{(i)}) = \frac{\widehat{\sigma}}{p(z_i)}\sqrt{\frac{P_i(1 - P_i)}{n}}$$

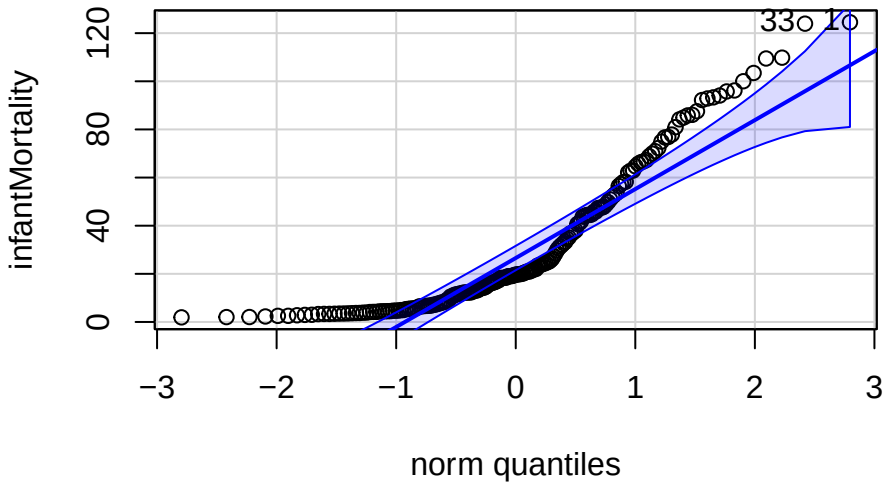
where $p(z)$ is the pdf (probability density function) corresponding to the CDF $P(z)$.

The values along the fitted line are given by $\widehat{X}_{(i)} = \widehat{\mu} + \widehat{\sigma}x_i$. An approximate 95 percent confidence 'envelope' around the fitted line is then $\widehat{X}_{(i)} \pm 2 \times SE(X_{(i)})$.

QQ plots highlight the tails of distributions. This is important, because the behavior of the tails is often problematic for standard estimation methods like least-squares, but it is useful to supplement QQ plots with other displays. QQ plots are usually used not to plot a variable directly but for derived quantities, such as residuals from a regression model.

The `qqPlot` function in the **car** package is useful:

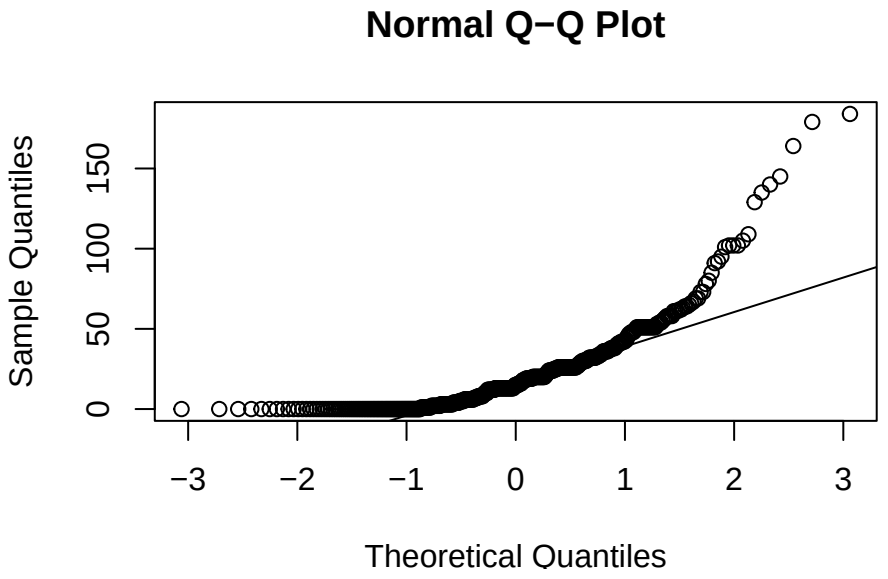
```
with(UN, qqPlot(infantMortality))
```




```
## [1]  1 33
```

Here, we will use **base graphics**.

```
qqnorm(helpdata$i2)
qqline(helpdata$i2)
```



Is average drinks/day normally distributed?

Plotting Options

`ggplot2` themes

Usually plots with white background look more readable when printed. We can set the background to white using the function `theme_bw()`. In addition to `theme_bw()`, `ggplot2` comes with several other themes which can be useful to quickly change the look of your visualization. The complete list of themes is available at https://ggplot2.tidyverse.org/reference/ggtheme.html. `theme_minimal()` and `theme_light()` are popular, and `theme_void()` can be useful as a starting point to create a new hand-crafted theme.

The `ggthemes` package provides a wide variety of options (including an STATA theme).

The `ggplot2` extensions website provides a list of packages that extend the capabilities of `ggplot2`, including

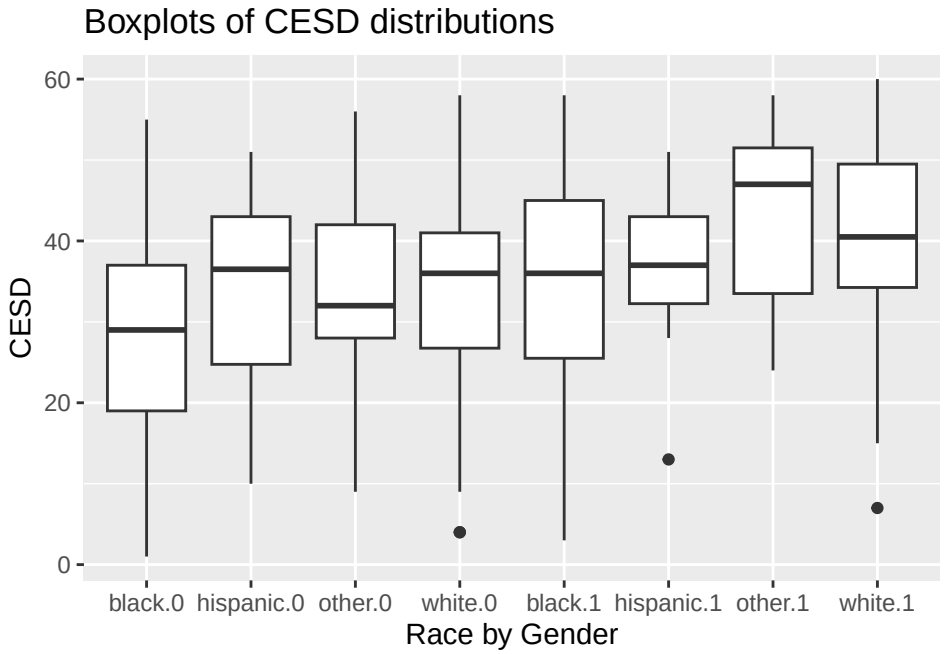
additional themes.

Note that it is also possible to change the fonts of your plots. If you are on Windows, you may have to install the **extrafont** package, and follow the instructions included in the README for this package.

Creating your own theme

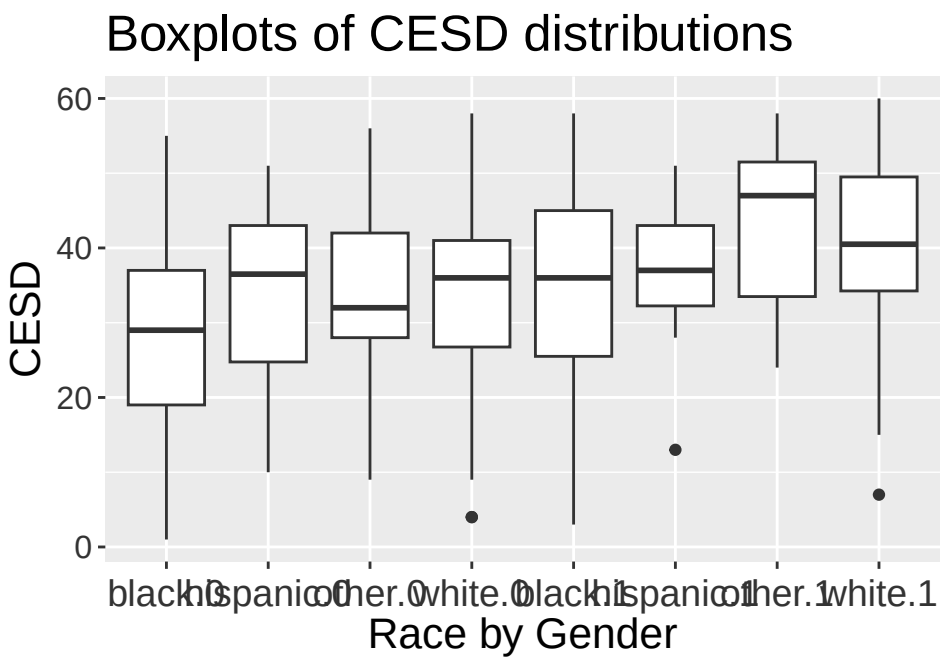
Let's begin with the boxplot we created earlier.

```
ggplot(helpdata, aes(x = interaction(racegrp, female), y = cesd)) +
    geom_boxplot() +
    labs(title="Boxplots of CESD distributions",
         x="Race by Gender", y="CESD")
```



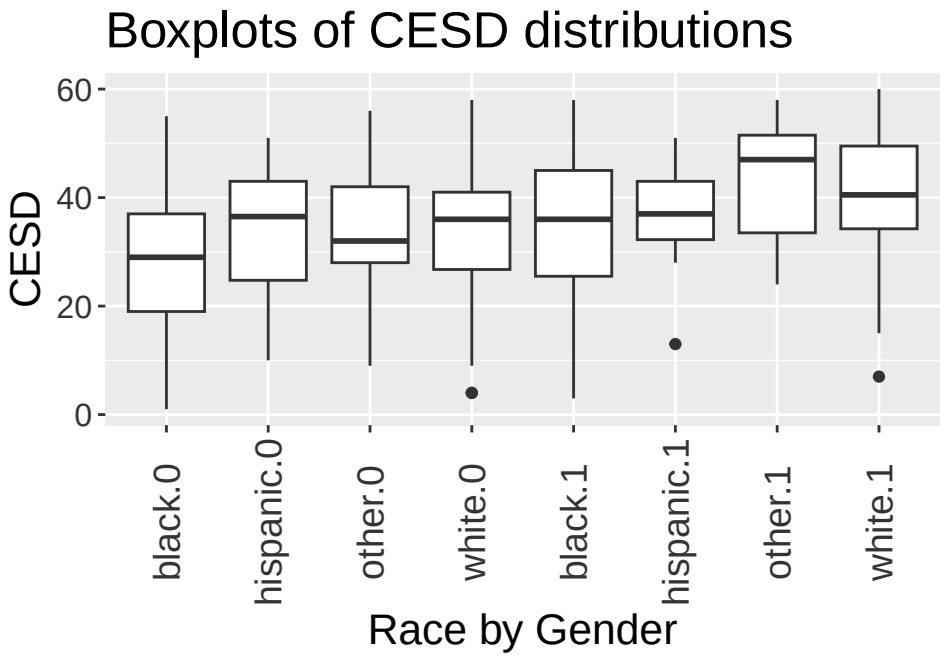
But let's now increase the font size to improve readability.

```
ggplot(helpdata, aes(x = interaction(racegrp, female), y = cesd)) +
    geom_boxplot() +
    labs(title="Boxplots of CESD distributions",
         x="Race by Gender", y="CESD") +
    theme(axis.text.x = element_text(colour = "grey20", size = 14,
                                     hjust = 0.5, vjust = 0.5),
          axis.text.y = element_text(colour = "grey20", size = 12),
                                      text = element_text(size = 16))
```



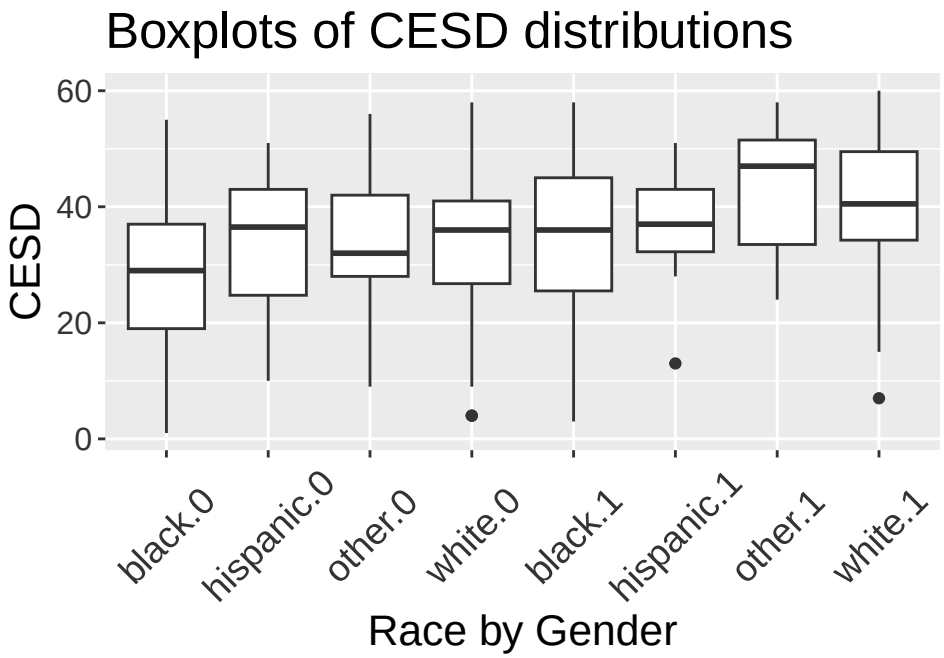
You change the orientation of the labels and adjust them vertically and horizontally.

```
ggplot(helpdata, aes(x = interaction(racegrp, female), y = cesd)) +
    geom_boxplot() +
    labs(title="Boxplots of CESD distributions",
         x="Race by Gender", y="CESD") +
    theme(axis.text.x = element_text(colour = "grey20", size = 14, angle = 90,
                                     hjust = 0.5, vjust = 0.5),
          axis.text.y = element_text(colour = "grey20", size = 12),
                                      text = element_text(size = 16))
```



You can use a 90 degree angle, or experiment to find the appropriate angle for diagonally oriented labels:

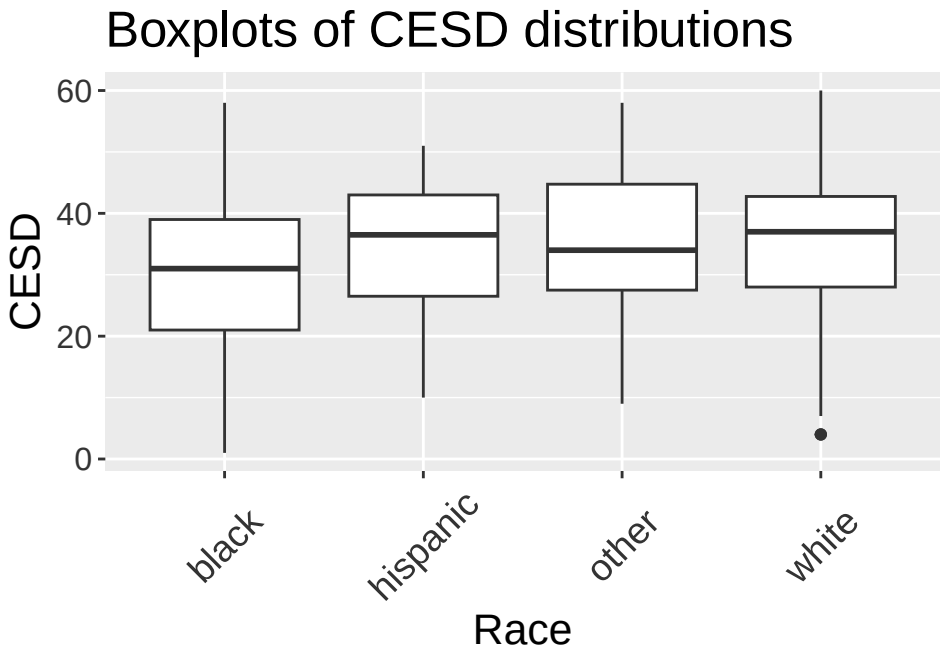
```
ggplot(helpdata, aes(x = interaction(racegrp, female), y = cesd)) +
    geom_boxplot() +
    labs(title="Boxplots of CESD distributions",
         x="Race by Gender", y="CESD") +
    theme(axis.text.x = element_text(colour = "grey20", size = 14, angle = 45,
                                     hjust = 0.5, vjust = 0.5),
          axis.text.y = element_text(colour = "grey20", size = 12),
                                     text = element_text(size = 16))
```



If you like the changes you created better than the default theme, you can save them as an object to be able to easily apply them to other plots you may create:

```
my_theme <- theme(axis.text.x = element_text(colour = "grey20", size = 14,
                                             angle = 45, hjust = 0.5, vjust = 0.5),
                  axis.text.y = element_text(colour = "grey20", size = 12),
                                             text = element_text(size = 16))


ggplot(helpdata, aes(x = racegrp, y = cesd)) +
    labs(title="Boxplots of CESD distributions",
         x="Race", y="CESD") +
    geom_boxplot() +
    my_theme
```

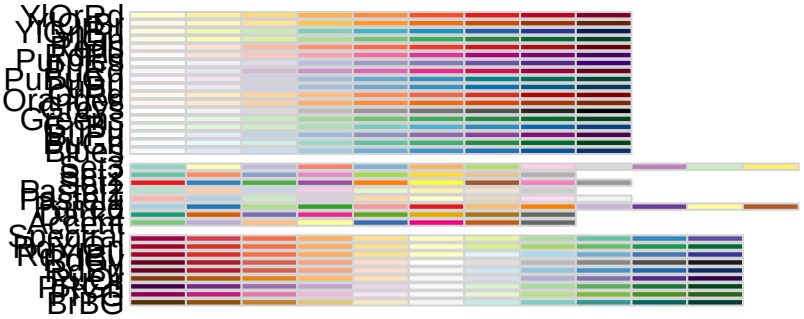


Colors

Try using different color palettes (see http://www.cookbook-r.com/Graphs/Colors_(ggplot2)/).

There are two packages that contain various color palettes. The first is `RColorBrewer`.

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



The other package is `colorspace`. Below are examples of some of the color palettes available in the `colorspace` package.

Be careful with colors because some people are colorblind. In fact, there are colorblind palettes. Here are two; the first with grey and the second with black.

```
cbPalette <- c("#999999", "#E69F00", "#56B4E9", "#009E73", "#F0E442", "#0072B2", "#D55E00", "#CC79A7")
cbbPalette <- c("#000000", "#E69F00", "#56B4E9", "#009E73", "#F0E442", "#0072B2", "#D55E00", "#CC79A7")
```

Summary

Statistical graphs are central to effective data analysis, both in the early stages of an investigation and in statistical modeling.

Nonparametric density estimation may be employed to smooth a histogram.

QQ plots are useful for comparing data with a theoretical probability distribution.

Boxplots summarize some of the most important characteristics of a distribution, including center, spread, skewness, and the presence of outliers. Parallel boxplots can be employed to display the relationship between a quantitative response variable and a discrete explanatory variable.

The bivariate scatterplot is a natural graphical display of the relationship between two quantitative variables. Interpretation of a scatterplot can often be assisted by graphing a nonparametric regression, which summarizes the relationship between the two variables.

Visualizing multivariate data is intrinsically difficult because we cannot directly examine higher-dimensional scatterplots. Effective displays project the higher-dimensional point cloud onto two or three dimensions.

MOST IMPORTANT RULE - LOOK AT YOUR DATA!