Statistical Literacy for Biologists: Week 1, Part 2
Univariate EDA

Dwight Barry

29 September 2017

Contents

Measurement Scales and Summary Statistics

Summary statistics, such as the mean, median, IQR, or standard deviation, can provide a useful window into a data set for both researchers and readers. But statistical tools, including summary statistics, are predicated on the type of data you have; some statistics just don't work with some types of data.[1]

Further, the types of statistical models you use to analyze your data depend on the measurement scales of the data you are exploring. So any analysis effort must start with being clear on measurement scales.

Statistic / Parameter	Categorical *Nominal*	Ranked *Ordinal*	Discrete/Counts *Interval/Ratio*	Continuous *Interval/Ratio*
Data set size (n)	Y	Y	Y	Y
Percent / frequency	Y	Y	Y	Y
Count or rate	Y	Y	Y	Y
Categories (levels)	Y	Y	Y	Y
Mode	Y	Y	Y	Y
Median	*No*	Y	Y	Y
Interquartile range	*No*	Y	Y	Y
Median absolute deviation	*No*	Y	Y	Y
Range	*No*	Y	Y	Y
Minimum/maximum value	*No*	Y	Y	Y
Quantiles	*No*	Y	Y	Y
Mean (average)	*No*	*No*	Y	Y
Standard deviation	*No*	*No*	Y*	Y*
Coefficient of variation	*No*	*No*	Y*	Y*

* You must use the correct distribution (proper mean-variance relationship) to ensure you get the correct standard deviation; most software defaults to calculating the standard deviation for a normally-distributed sample, which could be incorrect for certain kinds of count, rate, or proportion data, for example.

[1] The most egregious (and common) failure is the use of averages (and standard deviations) on ordinal scale variables, such as Likert scales. We won't cover those in this course, but you can get an overview of better approaches in the mildly-titled e-book, *Do not use averages on Likert scale data.*

What makes this a little trickier than you might think is that the same variable can be measured in many different ways. Take age for example: it could be continuous, a discrete count, ordinal, categorical, even binary.

So, consider the values you have in the data, the context in which those values were gathered, and the purpose of the analysis when deciding which summary statistics to report or which analytic tools to use.

How many decimals should I report?

Just because R's output gives you a bunch of decimals doesn't mean you need to use them. Think instead about how much precision is realistic for your values, and round accordingly.

A decent rule of thumb is to use no additional decimals for medians, no or 1 additional decimal for IQR or quantiles, 1 additional decimal for means, and 2 additional decimals for standard deviation. But these are not hard-and-fast rules; use what makes sense in the context of the values and their meaning in the analysis.

Univariate Exploratory Data Analysis (EDA)

The foundation of all analyses are the individual variables. So understanding each of them, by themselves, is the best place to start any analytics work.

And you should *always* start with visualizations, not summary statistics. Why? Take this example:

```r
# load some data
var1 = anscombe$y1
var2 = anscombe$y2

# calculate mean and sd
mean(var1); sd(var1)
```

```
## [1] 7.500909
```

```
## [1] 2.031568
```

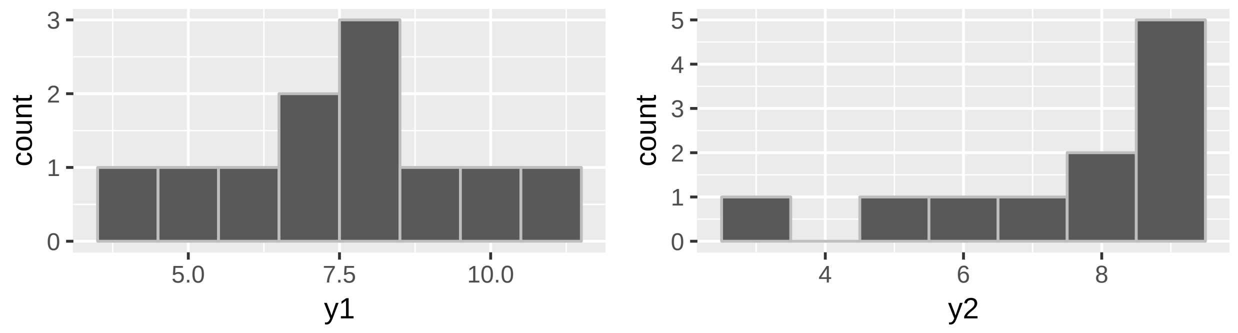
```r
mean(var2); sd(var2)
```

```
## [1] 7.500909
```

```
## [1] 2.031657
```

OK, so the mean and standard deviation are the same in these two variables. Now let's plot the distributions:

```r
p1 = ggplot(anscombe, aes(y1)) + geom_histogram(binwidth = 1, color = "gray")
p2 = ggplot(anscombe, aes(y2)) + geom_histogram(binwidth = 1, color = "gray")
grid.arrange(p1, p2, ncol = 2)
```



Clearly, these two variables *are* different, even though their summary statistics are practically identical.

Often, the use of summary statistics is fine, but it is essential to understand that they are *not necessarily* good descriptors of your data. We'll explore this issue a little later in this workshop.

The most fundamental mistake in data analysis and presentation is confusing *summary statistics* for a variable with the *distribution* of that variable.

Today's data

The 2003 Annual Meeting of the Statistical Society of Canada explored a case study aimed at predicting systolic blood pressure from a variety of physiological and genetic variables. We'll use the physiological variables to illustrate a variety of exploratory data analysis tools.[2]

As we saw earlier, even things as seemingly straightforward as age can be assessed in a variety of ways that influence both the statistical tools you should use as well as the results you obtain. For the purposes of this workshop, we will take the variables measurement scales *as given or implied*, i.e., we will treat `age` as continuous, `income` as categorical, and so on.

```r
# Load the data
data(sbp, package = "sbpdata")

# Let's look at the structure of this data
str(sbp)
```

```
## 'data.frame':    500 obs. of  17 variables:
##  $ sbp                    : int  133 115 140 132 133 138 133 67 138 130 ...
##  $ age                    : int  60 55 18 19 58 55 22 52 46 38 ...
##  $ weight                 : int  159 107 130 230 201 166 188 123 106 166 ...
##  $ height                 : int  56 65 59 57 74 67 66 67 73 72 ...
##  $ bmi                    : int  35 17 26 49 25 25 30 19 13 22 ...
##  $ Gender                 : Factor w/ 2 levels "Female","Male": 1 2 2 2 2 1 1 1 2 2 ...
##  $ Married                : Factor w/ 2 levels "Married","Unmarried": 2 2 2 1 2 2 1 1 1 1 ...
##  $ Smoke                  : Factor w/ 2 levels "Non-smoker","Smoker": 1 2 2 1 1 1 1 1 1 2 ...
##  $ Exercise               : Ord.factor w/ 3 levels "Low"<"Medium"<..: 3 1 1 2 2 3 1 3 1 3 ...
##  $ Overweight             : Ord.factor w/ 3 levels "Normal"<"Overweight"<..: 3 1 2 3 2 2 3 1 1 1 ...
##  $ Alcohol                : Ord.factor w/ 3 levels "Low"<"Medium"<..: 2 2 1 3 3 1 3 2 3 1 ...
##  $ Treatment_Group        : Factor w/ 2 levels "Control","Treatment": 1 1 1 2 1 2 2 1 2 2 ...
##  $ Stress                 : Ord.factor w/ 3 levels "Low"<"Medium"<..: 2 2 3 3 2 2 3 2 2 2 ...
##  $ Salt                   : Ord.factor w/ 3 levels "Low"<"Medium"<..: 2 2 2 3 2 1 1 3 2 2 ...
##  $ Childbearing_Potential: Factor w/ 3 levels "Able Female",..: 1 2 2 2 2 3 3 1 2 2 ...
##  $ Income                 : Ord.factor w/ 3 levels "Low"<"Medium"<..: 2 3 1 1 2 2 1 3 2 1 ...
##  $ Education              : Ord.factor w/ 3 levels "Low"<"Medium"<..: 2 2 3 2 3 3 1 2 1 1 ...
```

[2]The data in this package was created for SCH R workshops and is not available outside of `ratchet`. You can acquire the raw data and background from https://ssc.ca/en/case-studies-2003-annual-meeting.

Visualizing the distribution of variables

Always, *always*, ***always*** start with the visuals.

We'll use the `base` and `ggplot2` libraries to visualize our data, focusing on the `weight` and `Exercise` variables for the purposes of today's workshop.

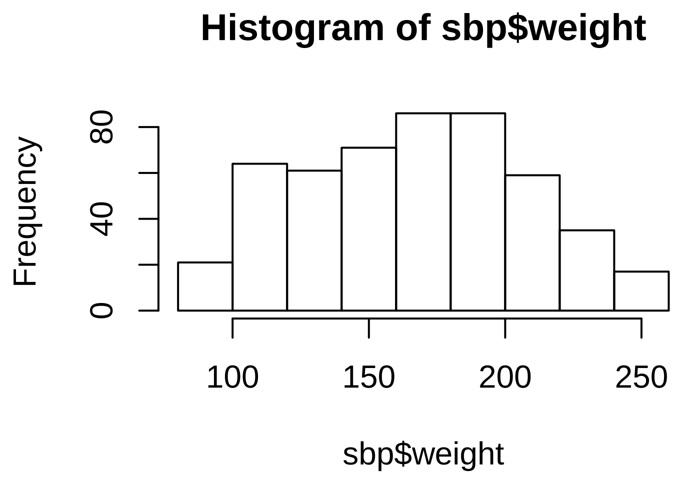
```
library(ggplot2)
```

Quick and Easy: `base` graphics

Numeric distributions in `base` R

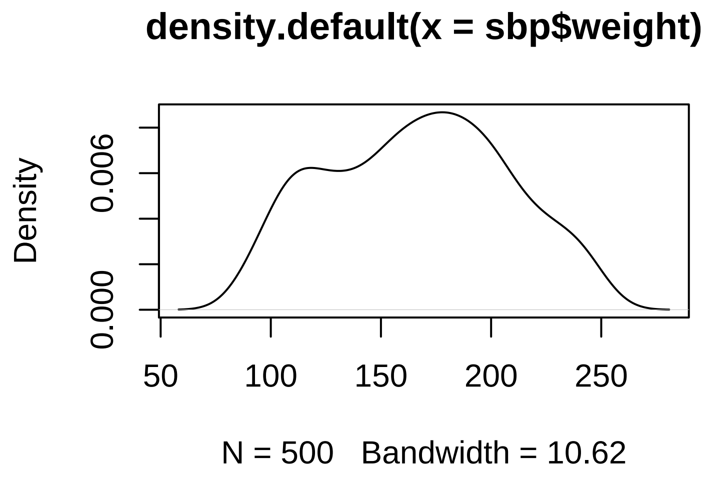
You should always start with a histogram to understand the distribution of your data. The quick-and-dirty way is using the `hist` function from the `base` package. Remember, you can access individual columns in a data frame with the `$` operator.

```
hist(sbp$weight)
```



Ugly, but easy. Ditto with density histograms:

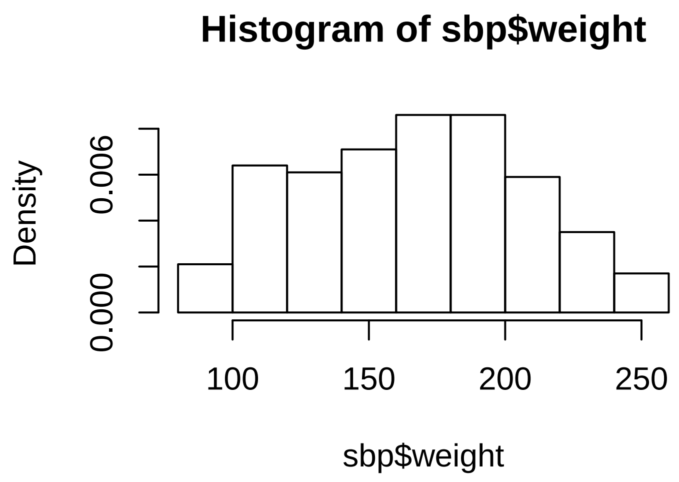
```
plot(density(sbp$weight))
```



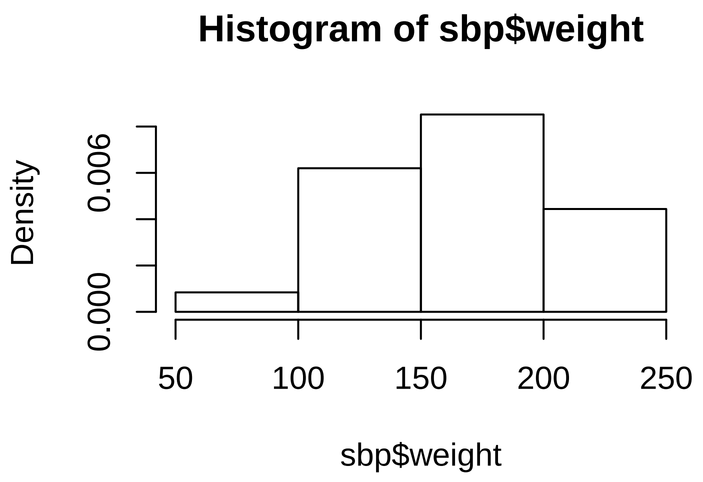
Why use density histograms? Well, because they're basically "more objective" histograms. The impression of the distribution can be modified substantially by changing the width of the bars, but the density histogram will maintain the same shape.

For example,

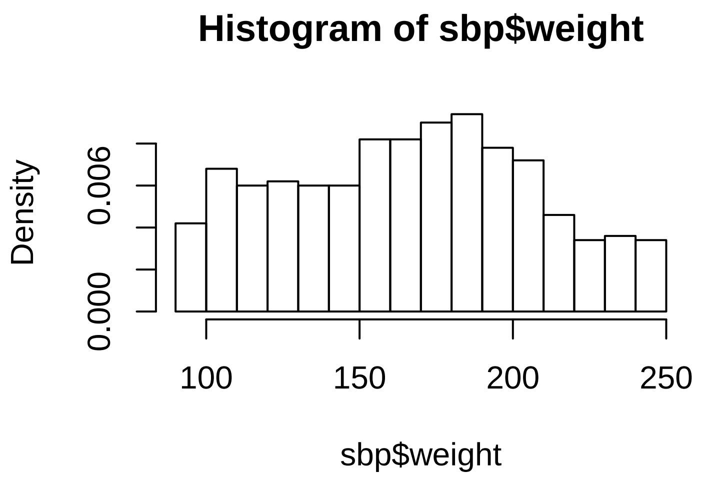
```
hist(sbp$weight, freq = F)
```



```
hist(sbp$weight, freq = F, breaks = 5)
```

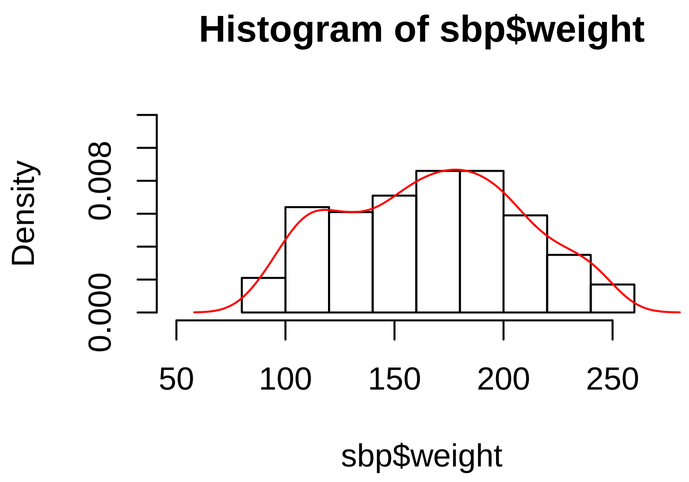


```
hist(sbp$weight, freq = F, breaks = 20)
```

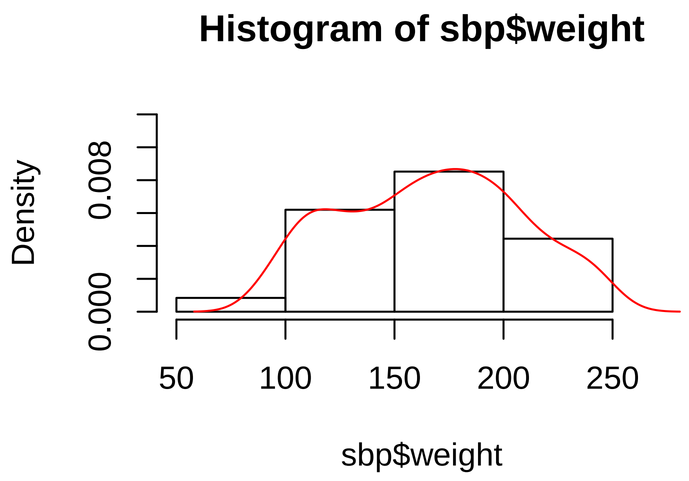


These definitely look different. But adding the density trace shows that they're from the same distribution:

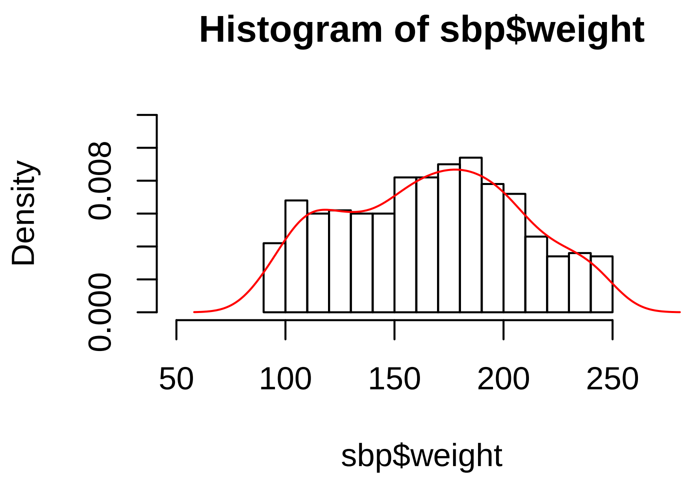
```r
hist(sbp$weight, freq = F, xlim = c(50, 275), ylim = c(0, 0.012))
lines(density(sbp$weight), col = "red")
```



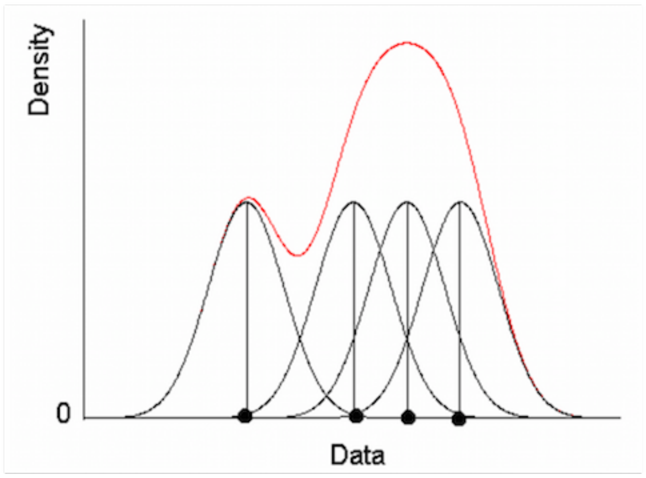
```r
hist(sbp$weight, freq = F, breaks = 5, xlim = c(50, 275), ylim = c(0, 0.012))
lines(density(sbp$weight), col = "red")
```



```r
hist(sbp$weight, freq = F, breaks = 20, xlim = c(50, 275), ylim = c(0, 0.012))
lines(density(sbp$weight), col = "red")
```

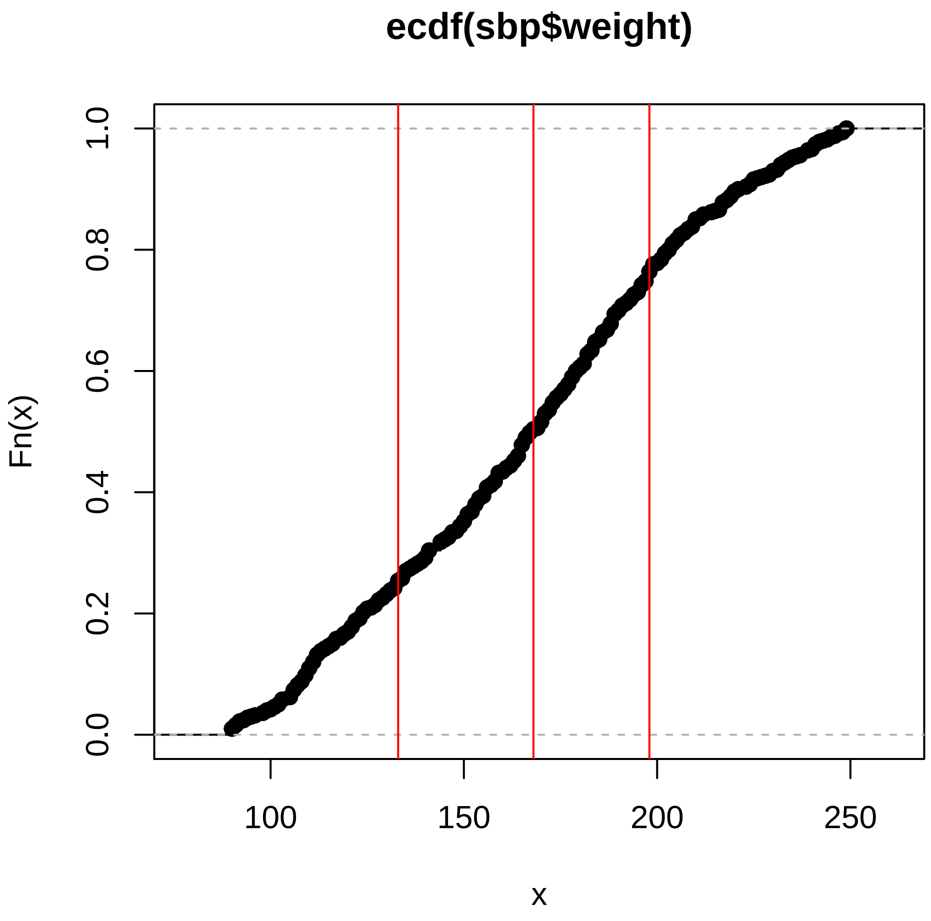


In case you're curious, here's how a density shape is created:



Another `base` package quick visualization for the distribution of continuous data is the empirical cumulative distribution function (`ecdf`), which plots the values of the distribution against the percentiles (median and IQR lines added in red for reference):

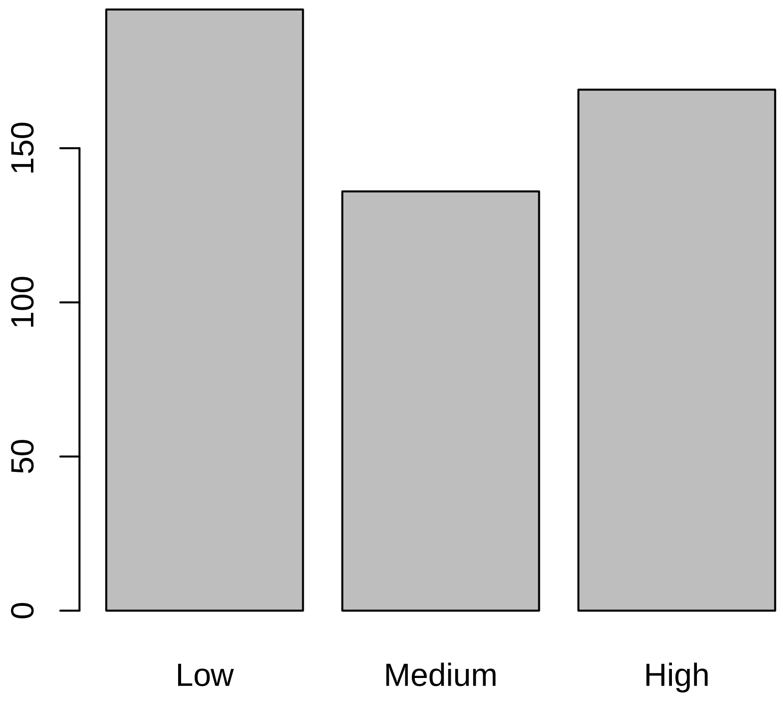
```r
plot(ecdf(sbp$weight))
abline(v = quantile(sbp$weight, probs = c(0.25, 0.5, 0.75)), col = "red")
```



Categorical distributions in `base` R

The `barplot` function provides the quick-and-dirty way to visualize categorical distributions:

```r
barplot(table(sbp$Exercise))
```



Exercise R1

1. Create a histogram of the `sbp` variable (systolic blood pressure).
2. Create a bar plot of the `Salt` variable.
3. *Extra credit:* add a density line to the `sbp` histogram.

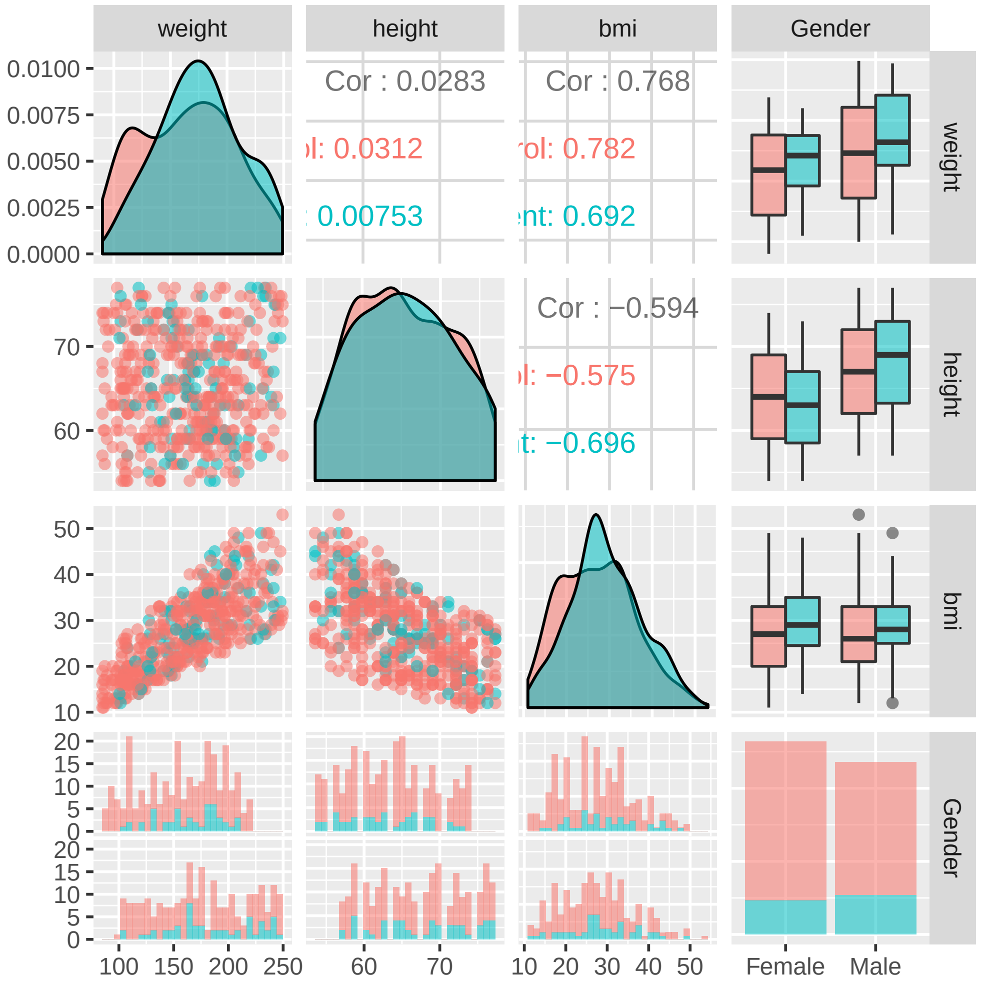
---

ggplot2 graphics

It's worth having the `base` functions in your toolbox simply because they are quick to type out during initial exploration. But you probably want to use `ggplot2` for most anything requiring more than a cursory glance.

Why? Let's start with the end: a multi-panel data viz summary.

```r
# We'll use the GGally package to create a "master EDA overview"
library(GGally)

# ggpairs is slow, so we'll only use a subset of data
ggpairs(data = sbp, columns = c(3:6), aes(color = Treatment_Group, alpha = 0.5))
```



The power of `ggplot2` comes from its immense ecosystem of add-ons and customization options, as well as its seamless integration with other key R packages (e.g., `dplyr`, `tidyr`, `forcats`, etc.). This, of course, can come at the cost of complication, but we think the learning curve is worth it: you can get publication quality graphs that you'd simply be unable to create in other tools.

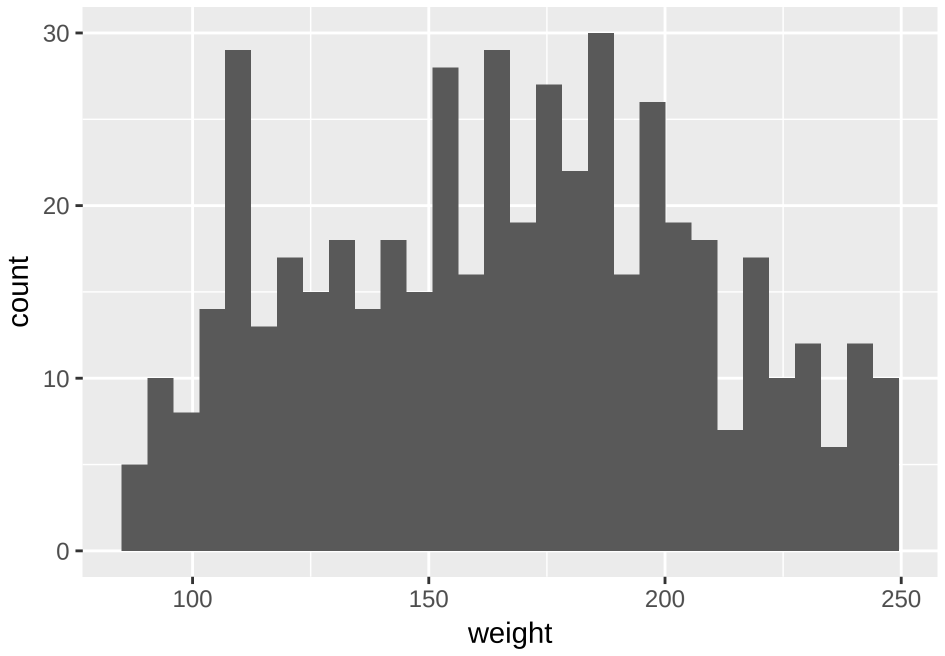
We'll now create the same graphs as we did with `base` using `ggplot2` instead.

Note the small and large changes in the code as we progress through these examples. They are there on purpose, to introduce different options and ideas for customizing plots. Like most things R, it will take practice and use in real projects to instill familiarity with `ggplot2`. But these examples can help serve as a leap-off point for your own work.

Numeric distributions in `ggplot2`

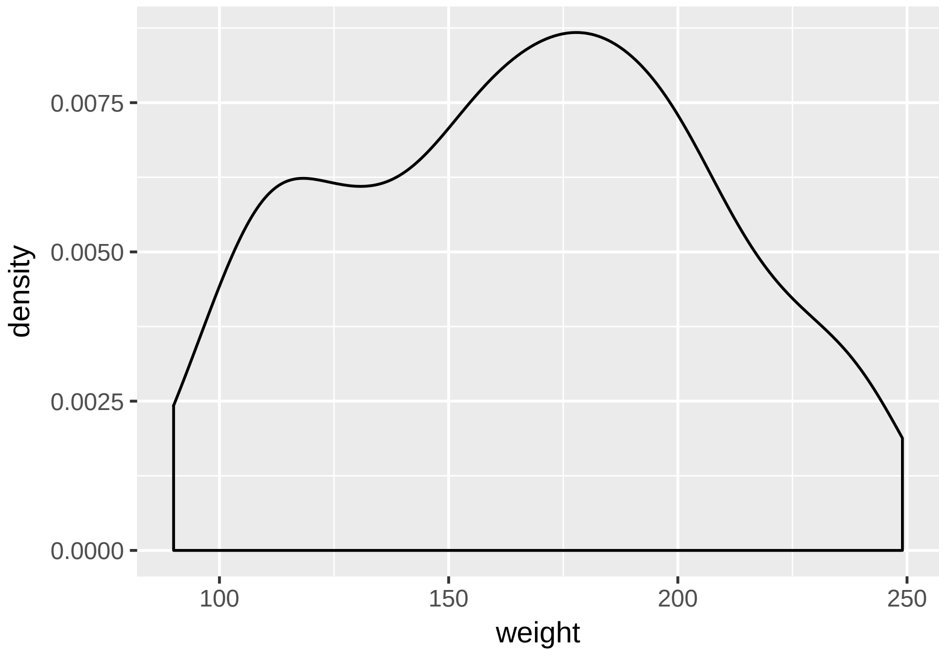
Histogram

```r
# Histogram
ggplot(data = sbp, aes(x = weight)) +
    geom_histogram()
```



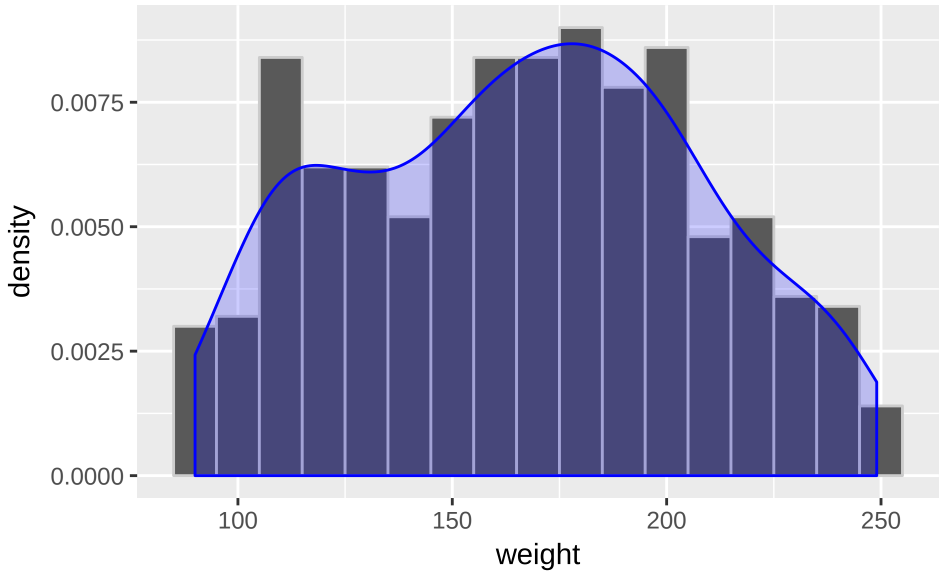
Density histogram

```r
# Density
ggplot(sbp, aes(weight)) +
    geom_density()
```



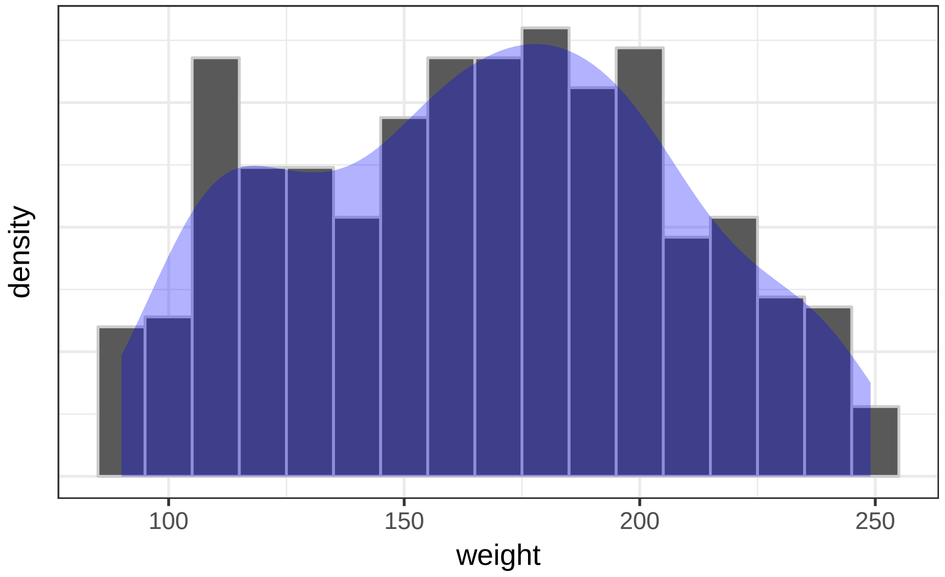
Combination histogram

```
# Histogram and density, with a custom bin width and some color/fill options
# Note the ..density.. option, which puts the histogram on the density scale
ggplot(sbp, aes(weight)) +
    geom_histogram(aes(y = ..density..), color = "gray80", binwidth = 10) +
    geom_density(color = "blue", fill = "blue", alpha = 0.2)
```



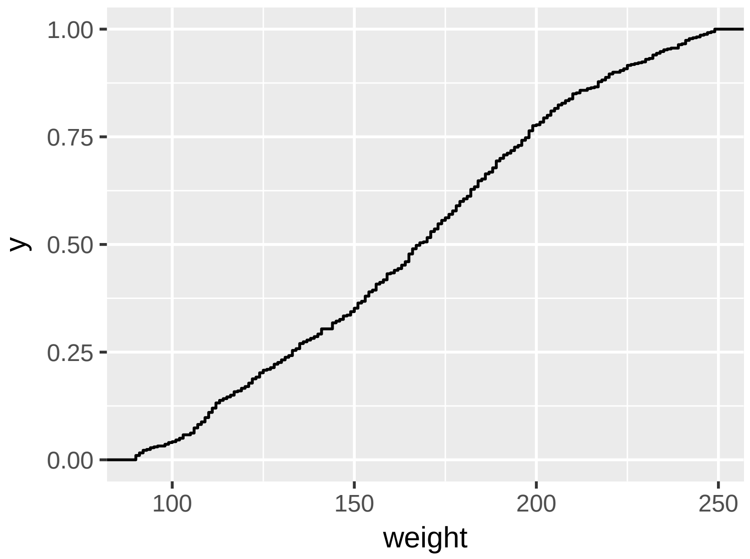
Combination histogram, cleaned up a little

```
# Histogram and density, with a few color and fill options, and a plain theme
# and the actual density value is meaningless, so we'll remove it
ggplot(sbp, aes(weight)) +
      geom_histogram(aes(y = ..density..), color = "gray80", binwidth = 10) +
      geom_density(color = "transparent", fill = "blue", alpha = 0.3) +
      theme_bw() +
      # note: you must use theme options after theme type
      theme(axis.ticks.y = element_blank(), axis.text.y = element_blank())
```



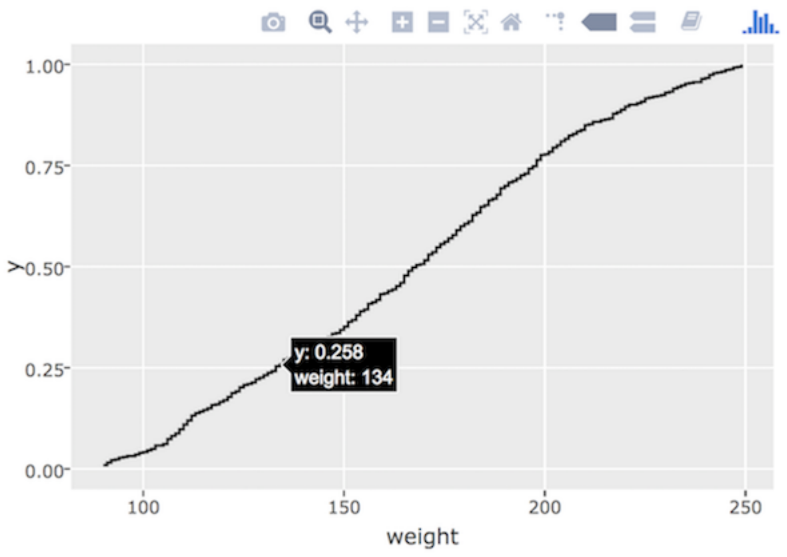
`ggplot2` also has built-in statistical functions, so you can create, e.g., an ecdf plot, error bars, overlays of theoretical distributions, and many other tasks (see `?stat_function` and `?stat_`). Here, we'll create an ecdf plot:

```r
ggplot(sbp, aes(x = weight)) +
    stat_ecdf()
```



If we assign the ggplot to an object, we can see how other packages can use ggplots; in this example, we'll make our edcf plot interactive using the `plotly` package:

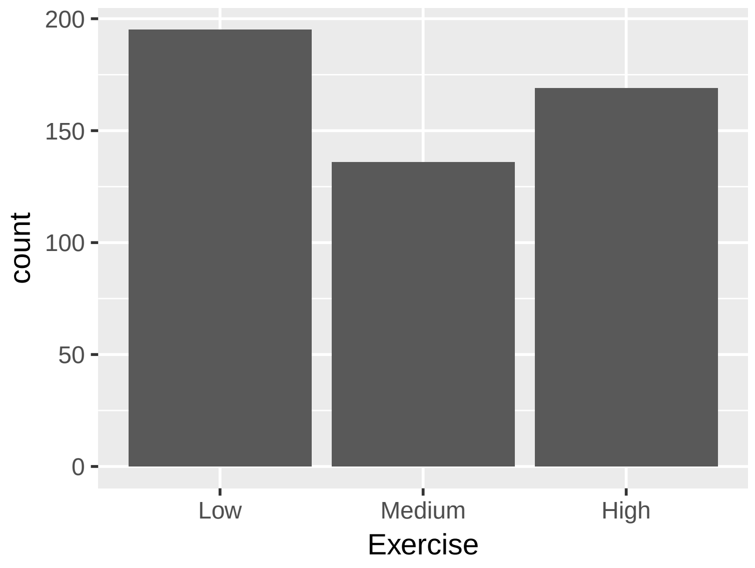
```r
# First, order the data by the variable you want to plot
sbp = arrange(sbp, weight)

# Then create a ggplot object
ecdf_plot_weight = ggplot(sbp, aes(x = weight)) +
    stat_ecdf()

# Finally, pass the ggplot object to another package, e.g., plotly
plotly::ggplotly(ecdf_plot_weight)
```



Categorical distributions in `ggplot2`

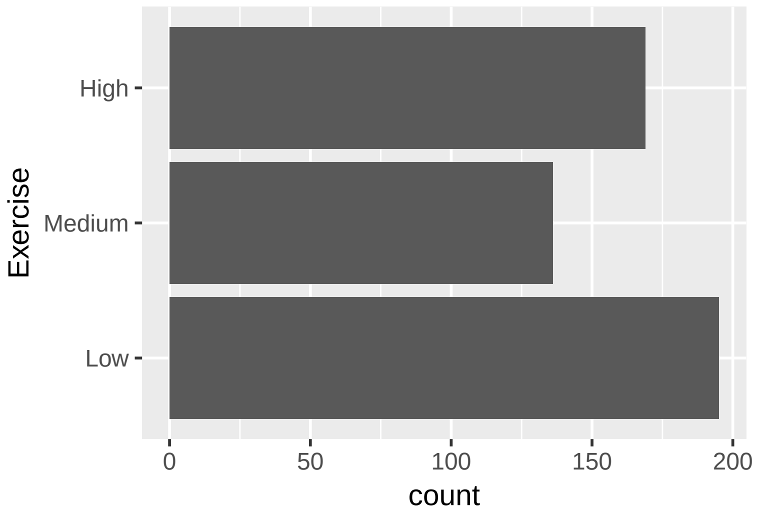
Barplot

```r
# Barplot
ggplot(sbp, aes(x = Exercise)) +
    geom_bar()
```



We can turn it (or any other ggplot) sideways with `coord_flip()`:

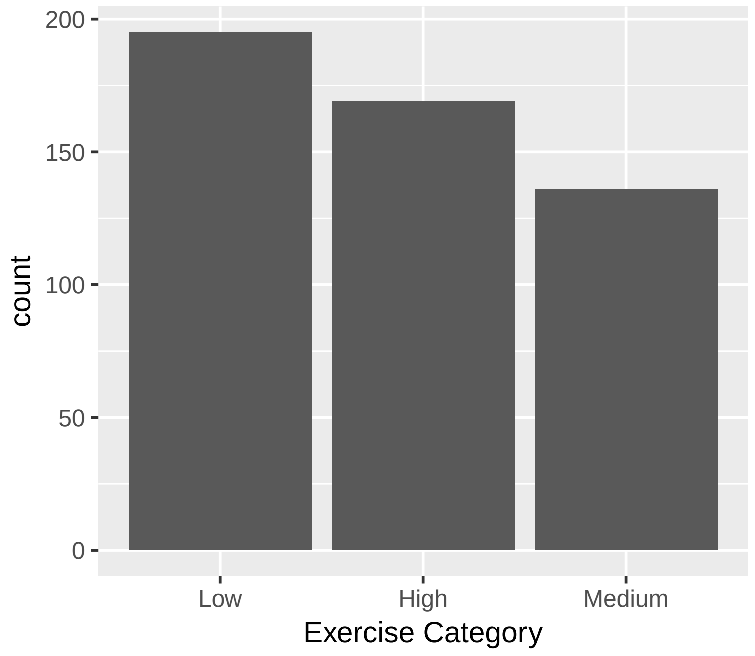
```r
# Barplot
ggplot(sbp, aes(x = Exercise)) +
    geom_bar() +
    coord_flip()
```



Let's say we want it ordered by count, from top to bottom.

```r
# We need to load the forcats library (helps with categorical variables) first
library(forcats)
```

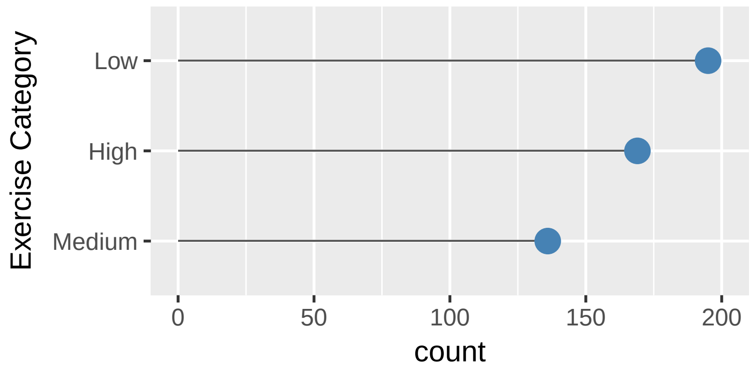
```r
# Barplot with values ordered by count, from origin
# fct_infreq orders by frequency
ggplot(sbp, aes(x = fct_infreq(Exercise))) +
    geom_bar() +
    xlab("Exercise Category")
```



Many people now prefer a dot or “lollipop” graph to bars:

Dot / “Lollipop” graph

```r
# fct_rev reverses the order so most frequent is on top
ggplot(sbp, aes(x = fct_rev(fct_infreq(Exercise)), y = ..count..)) +
    # Make a very skinny bar for the stem
    geom_bar(stat = "count", width = 0.02) +
    geom_point(stat = "count", size = 4, color = "steelblue") +
    coord_flip() +
    # We flipped the graph but ggplot uses original aes
    ylim(0, 200) +
    xlab("Exercise Category")
```



Saving ggplots

We can save a publication-ready/hi-res plot with `ggsave`, which saves the current plot unless otherwise specified. Save any ggplot in the current environment by specifying its name in the `plot` option, and use the `filename` extension for the file type you want, e.g., for a `png` file you'd use this approach:

```r
# This will appear in your current working directory; use getwd() if needed
ggsave(filename = "ecdf_plot_weight.png",
       plot = ecdf_plot_weight,
       dpi = 600,
       width = 4,
       height = 4,
       units = "in")
```

Exercise R2

Using `ggplot2`:

1. Create a histogram/density plot of the `sbp` variable (systolic blood pressure), using a custom color fill for density (Google "r color names" or try #A30134 if you're stumped).

2. Create a bar plot of the `Salt` variable, ordered with the highest value closest to the x-axis.

3. *Extra credit:* make one of these graphs interactive.

Summary statistics

The simplest way to get basic summary stats in `base` R is to use the `summary` function.

```r
summary(sbp)
```

```
##       sbp              age            weight           height
##  Min.   : 67.0   Min.   :18.0   Min.   : 90.0   Min.   :54.00
##  1st Qu.:130.0   1st Qu.:28.0   1st Qu.:133.0   1st Qu.:60.00
##  Median :140.5   Median :40.0   Median :168.0   Median :65.00
##  Mean   :145.0   Mean   :40.2   Mean   :166.6   Mean   :65.33
##  3rd Qu.:162.2   3rd Qu.:52.0   3rd Qu.:198.0   3rd Qu.:70.00
##  Max.   :224.0   Max.   :64.0   Max.   :249.0   Max.   :77.00
##       bmi            Gender          Married            Smoke
##  Min.   :11.00   Female:264   Married  :239   Non-smoker:234
##  1st Qu.:21.00   Male  :236   Unmarried:261   Smoker    :266
##  Median :27.00
##  Mean   :27.66
##  3rd Qu.:33.00
##  Max.   :53.00
##    Exercise          Overweight    Alcohol      Treatment_Group    Stress
##  Low   :195   Normal     :187   Low   :160   Control  :399    Low   :151
##  Medium:136   Overweight:109   Medium:167   Treatment:101    Medium:175
##  High  :169   Obese      :204   High  :173                    High  :174
##
##
##
##      Salt        Childbearing_Potential    Income      Education
##  Low   :166   Able Female  :143         Low   :176   Low   :171
##  Medium:157   Male         :236         Medium:167   Medium:159
##  High  :177   Unable Female:121         High  :157   High  :170
##
##
##
```

You get counts for categorical variables, and the 5-number summary plus the arithmetic mean for numeric variables.

For numeric data, you can use `quantile` to obtain the quantiles of your choice in the `probs` option.

```r
quantile(sbp$age, probs = c(0.025, 0.20, 0.50, 0.80, 0.975))
```

```
##  2.5%   20%   50%   80% 97.5%
##    18    26    40    54    62
```

The `ecdf` function allows you to go the other way and find out what quantile a certain number is at—assign it to an object and call the object with the value you want to see the quantile for:

```r
what_quantile = ecdf(sbp$weight)
what_quantile(134)
```

```
## [1] 0.258
```

There are several packages that provide more useful summary functions than `base`; one I've found useful for quantitative variables is the `psych` package, particularly for conditional summary stats (which we'll explore next week). Here, we'll use its version of `summary`, which is called `describe`.

```
# Load psych package
library(psych)
```

`describe` only works on numeric variables, and will *treat your categorical variables as numeric* if you try to use it on those variables. It does provide an * on those variables, but for clarity's sake, use the `is.numeric` function with `sapply` to alleviate this potential problem.

```
# Not run:
# describe(sbp)

# Summary stats for numeric variables only
describe(sbp[ , sapply(sbp, is.numeric)])
```

```
##        vars   n   mean    sd median trimmed   mad min max range skew
## sbp       1 500 144.95 27.99  140.5  143.79 21.50  67 224   157 0.36
## age       2 500  40.20 13.30   40.0   40.16 17.79  18  64    46 0.02
## weight    3 500 166.64 40.90  168.0  166.24 45.96  90 249   159 0.01
## height    4 500  65.33  6.19   65.0   65.28  7.41  54  77    23 0.09
## bmi       5 500  27.66  8.56   27.0   27.33  8.90  11  53    42 0.29
##        kurtosis   se
## sbp        0.17 1.25
## age       -1.19 0.59
## weight    -0.91 1.83
## height    -1.02 0.28
## bmi       -0.47 0.38
```

Counts and tables

You can use the `table` function to get counts for categorical variables; pre-pending `prop.table` gives you percentages.

```
table(sbp$Exercise)
```

```
## 
##    Low Medium   High 
##    195    136    169
```

```
prop.table(table(sbp$Exercise))
```

```
## 
##    Low Medium   High 
##  0.390  0.272  0.338
```

Wrapping a table in the `addmargins` function gives you a sum as well:

```
addmargins(table(sbp$Exercise))
```

```
## 
##    Low Medium   High    Sum 
##    195    136    169    500
```

Aside: **But R console output is ugly!**

Yes. Yes it is.

An easy way to make it pretty in an R markdown file is with `kable` from the `knitr` package. There are a variety of other packages that can create pre-formatted tables with more customization, e.g., `htmlTable` and xtable; here, we use describe output with kable for its nice default look in pdf:

```
# Create a describe object
sbp_summary = describe(sbp[ , sapply(sbp, is.numeric)], skew = FALSE, IQR = TRUE)

# Output describe object to a table, rounded to 1 place
knitr::kable(sbp_summary, digits = 1)
```

	vars	n	mean	sd	min	max	range	se	IQR
sbp	1	500	145.0	28.0	67	224	157	1.3	32.2
age	2	500	40.2	13.3	18	64	46	0.6	24.0
weight	3	500	166.6	40.9	90	249	159	1.8	65.0
height	4	500	65.3	6.2	54	77	23	0.3	10.0
bmi	5	500	27.7	8.6	11	53	42	0.4	12.0

Mode

Despite its name, R's mode function does not give you the mode of a distribution (it actually gives you how something is stored in R). There is no built-in function in R to get the mode (or modes), but a function cribbed from Stack Overflow can get us this information:

```
# Function to calculate mode(s)
Mode <- function(x, na.rm = FALSE) {
    if(na.rm){
      x = x[!is.na(x)]
    }
  x <- sort(x)
  u <- unique(x)
  y <- lapply(u, function(y) length(x[x==y]))
  u[which(unlist(y) == max(unlist(y)))]
}
```

Which will work for numeric…

```
Mode(sbp$bmi)
```

```
## [1] 31 32
```

…as well as categorical variables.

```
Mode(sbp$Exercise)
```

```
## [1] Low
## Levels: Low < Medium < High
```

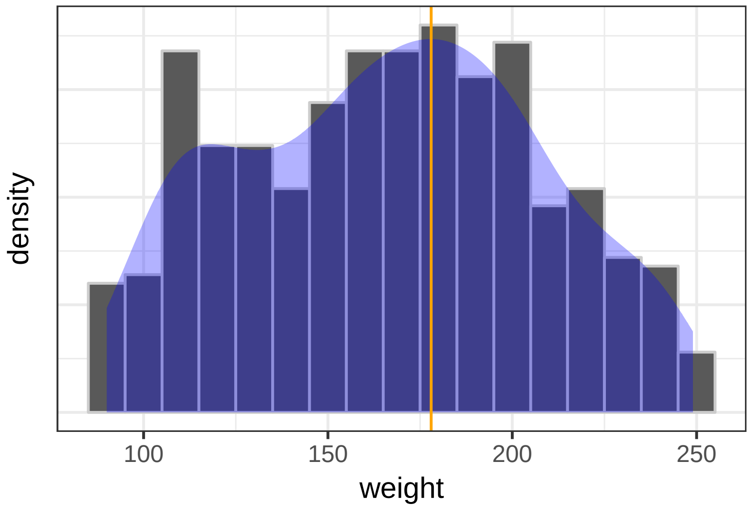
We don't often need the exact value of the mode in continuous data, but it sometimes can be a useful value to show in plots. We can get the value of maximum density with the which.max function, which can be used alone or in a plot, either as-is or in an object.

```
# Calculate density and place into a data frame
weight_dens = data.frame(weight = density(sbp$weight)$x,
                         density = density(sbp$weight)$y)

# Obtain the weight value at which density is maximized
weight_dens[which.max(weight_dens[, 2]), ]
```

```
##        weight     density
## 276 177.9995 0.008674805
```

Now we can plot that as a line in a combined histogram + density plot using `geom_vline`:

```
ggplot(sbp, aes(weight)) +
    geom_histogram(aes(y = ..density..), color = "gray80", binwidth = 10) +
    geom_density(color = "transparent", fill = "blue", alpha = 0.3) +
    geom_vline(aes(xintercept = weight_dens[which.max(weight_dens[, 2]), 1]),
               color = "orange") +
    theme_bw() +
    theme(axis.ticks.y = element_blank(), axis.text.y = element_blank())
```



There are packages with functions to calculate modes (e.g., `modeest`), but they come with additional complexity that is not usually necessary for typical analytic needs.

Exercise R3

1. Create a table of counts for the `Salt` variable that includes a sum value.

2. Use the `describe` function to obtain summary stats for the `sbp` variable *only* (hint: think back to how R identifies columns).

3. *Extra credit:* Plot the mean, median, and mode as vertical lines of different colors on a histogram + density ggplot for the `sbp` variable.

Table 1 (the horror!)

The `tableone` package has a simple interface for creating the typical *Table 1* that you find in many medical research papers.

```r
# Load tableone package
library(tableone)

# Create a variable list for Table 1
table_vars = c("age", "height", "weight", "bmi", "Gender", "Married",
               "Smoke", "Exercise", "Stress")

# Specify the categorical variables
fct_vars = c("Gender", "Married", "Smoke", "Exercise", "Stress")

# Create the table, without using NHST, stratified by treatment group
table_1_lies = CreateTableOne(table_vars, sbp, fct_vars,
                strata = "Treatment_Group", test = FALSE)

# Put the table into an object so it will show up in Rmd files
table_1_lies_object = print(table_1_lies, showAllLevels = TRUE, quote = FALSE,
                noSpaces = TRUE, printToggle = FALSE)

# Print the table
knitr::kable(table_1_lies_object, align = 'crr')
```

	level	Control	Treatment
n		399	101
age (mean (sd))		39.96 (13.13)	41.12 (13.98)
height (mean (sd))		65.29 (6.18)	65.50 (6.28)
weight (mean (sd))		164.15 (41.71)	176.48 (36.09)
bmi (mean (sd))		27.26 (8.62)	29.24 (8.17)
Gender (%)	Female	217 (54.4)	47 (46.5)
	Male	182 (45.6)	54 (53.5)
Married (%)	Married	195 (48.9)	44 (43.6)
	Unmarried	204 (51.1)	57 (56.4)
Smoke (%)	Non-smoker	193 (48.4)	41 (40.6)
	Smoker	206 (51.6)	60 (59.4)
Exercise (%)	Low	151 (37.8)	44 (43.6)
	Medium	113 (28.3)	23 (22.8)
	High	135 (33.8)	34 (33.7)
Stress (%)	Low	125 (31.3)	26 (25.7)
	Medium	140 (35.1)	35 (34.7)
	High	134 (33.6)	40 (39.6)

Awesome, right? That saves a few hours of cut, paste, and Word table formatting!

Yes, it does, but be careful: you don't want to lie with *Table 1*.

So why can *Table 1* be a problem?

In the olde days, creating graphs was extremely difficult, but calculating summary stats was fairly trivial. The thinking went that although the reader didn't have the full set of information that a distribution plot would provide, they at least had something to go on that would help them mentally visualize the approximate (normal) distribution. Further, it was compact: a bunch of summary information gathered into a single space.

But then some researchers started generating p-values on each variable, not understanding how they might be violating the principles of good stats use (experiment-wise error rate, for one).

Others used it for data that was clearly non-normal, rendering the standard deviation values pointless, at best, and misleading, at worst.

Why?

For example, imagine having a small mean and using a normal distribution on a variable that cannot go below zero, such as age. Using the normal (Gaussian) mean and standard deviation can imply that impossible values are actually part of your data, and that values near the mean are more common than those further away from the mean. We'll pretend we have some data on the counts of tumors on a model organism, and see what a traditional *Table 1* would imply.

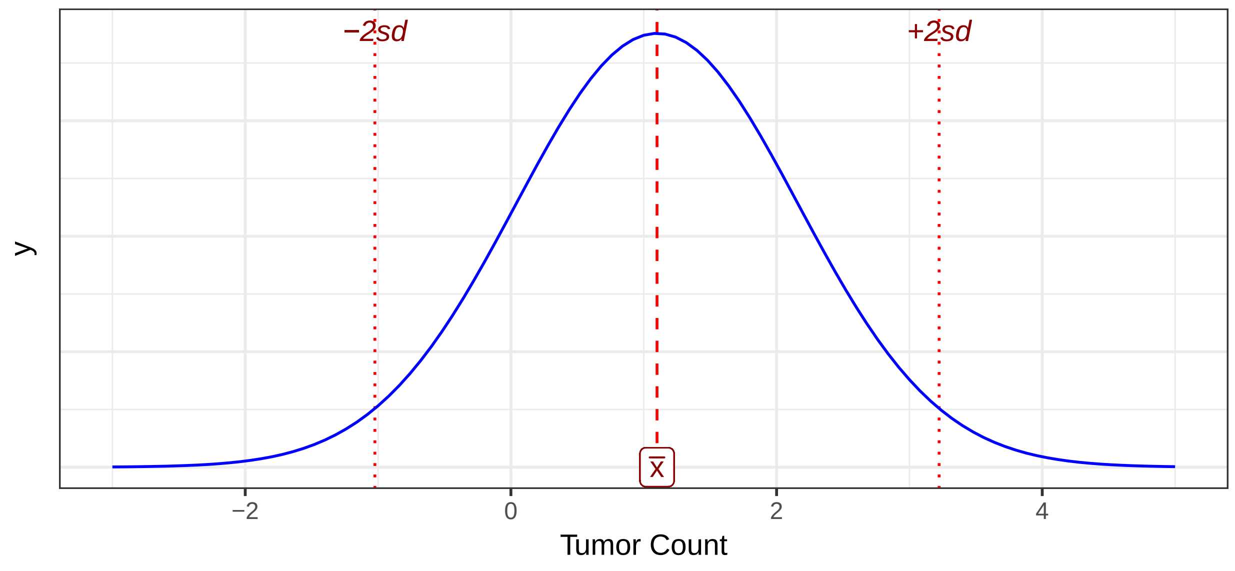
```r
# Create some random Poisson count data with mean = 1
set.seed(54)
tumor_count = data.frame(x = rpois(30, 1))

# Calculate mean
mean(tumor_count$x)
```

```
## [1] 1.1
```

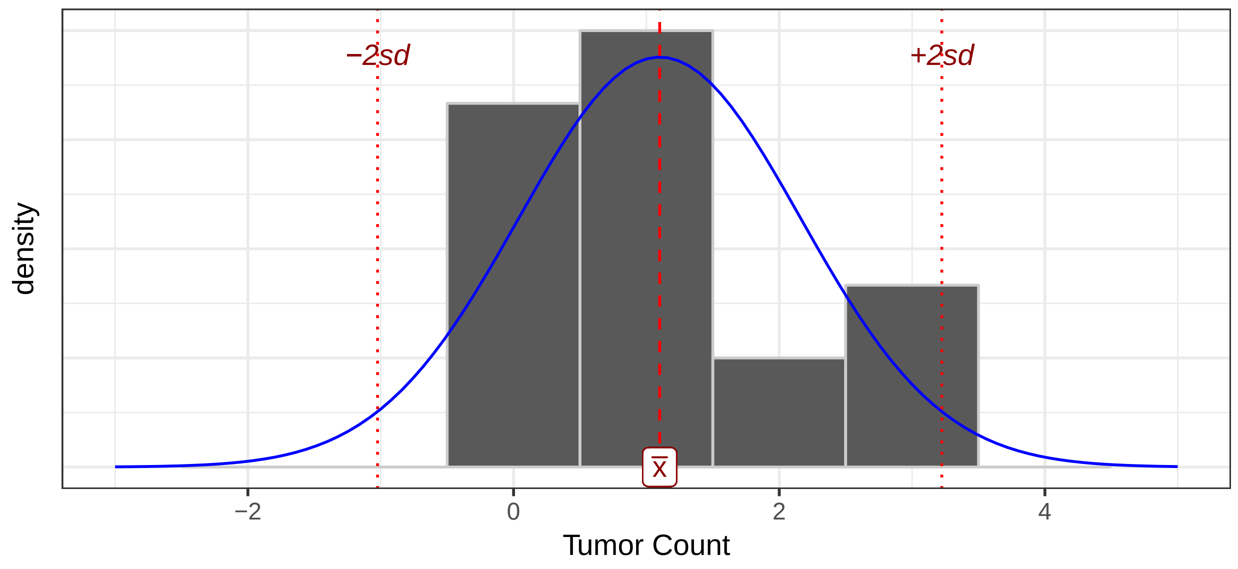
```r
# Calculate normal distribution standard deviation
sd(tumor_count$x)
```

```
## [1] 1.061879
```

So given only a mean of 1.1 and a standard deviation of 1.06—as you would see in a typical *Table 1*—you are implying that your data follow this distribution:



Clearly, a negative tumor count is impossible, which is indeed the case when you plot the *actual* distribution:



Summary statistics for a count variable are often better described by the Poisson distribution, for which the mean is still the arithmetic mean, but the standard deviation is the square root of the mean. This was the reason for the footnote on the *Summary Statistics* table on page 1 of this document: you need to use the proper mean/variance relationship to ensure you are providing the correct values. With this example, the mean would be the same, but the (Poisson) standard deviation would be 1.05, the theoretical distribution would stop at 0—and you'd need to specify somewhere (e.g., in a footnote) that you used the Poisson distribution to obtain this value.

There are many instances where a basic *Table 1* is just fine, especially if the continuous variables conform to at least an approximately normal distribution. But there are others when it could be obvious that it's wrong, as in the example above, and others in which it could be outright misleading. If you lose credibility in *Table 1*, it's going to be a much harder sell to get readers to trust your *Results* section.

And if you're a clinician, and you're considering using a study in practice, given only the mean and standard deviation for (say) the ages of patients, would you be confident that their sample represents the same population you are interested in?

~ *End of class* ~

Homework: Summary Stats, Data Viz, & the (insidious?) *Table 1*

1. Use either a data set of your choice (perhaps something you're working on now) or data on post-neurosurgery cognitive outcomes available on GitHub[3] to obtain summary stats and a visual of the distribution of at least *one numeric* and/or *one categorical* variable.

2. Create a simple *Table 1* summary from your own data or from the neurosurgery data linked in question 1.

3. *Extra credit:* create an "advanced" *Table 1* (see below) from your own data or from the neurosurgery data linked in question 1, using `Phase` as the strata.

Extra Credit: Advanced *Table 1*

Let's look at the Table 1 and implied distributions idea by returning to our original Stats Canada case study data, and take the study group as a whole, instead of being stratified into *control* and *treatment* groups.

```
mean(sbp$age)
```

```
## [1] 40.196
```

```
sd(sbp$age)
```

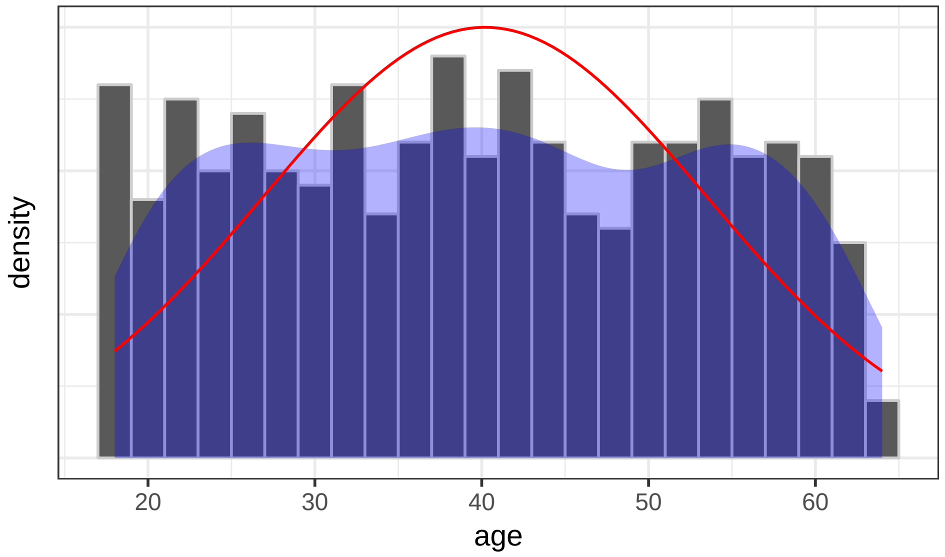
```
## [1] 13.29854
```

So given this in a *Table 1*, it would be suggesting that ~67% of these patients are between 27 and 53 years old, and that ~95% of them are between 14 and 67 years old. Is that true for this data?

Let's look at it. (Always start with visuals!)

```
# Histogram/density with normal curve overplotted
ggplot(sbp, aes(x = age)) +
    geom_histogram(aes(y = ..density..), color = "gray80", binwidth = 2) +
    geom_density(color = "transparent", fill = "blue", alpha = 0.3) +
    stat_function(fun = dnorm, color = "red",
        args = list(mean = mean(sbp$age), sd = sd(sbp$age))) +
    theme_bw() +
    theme(axis.ticks.y = element_blank(), axis.text.y = element_blank())
```

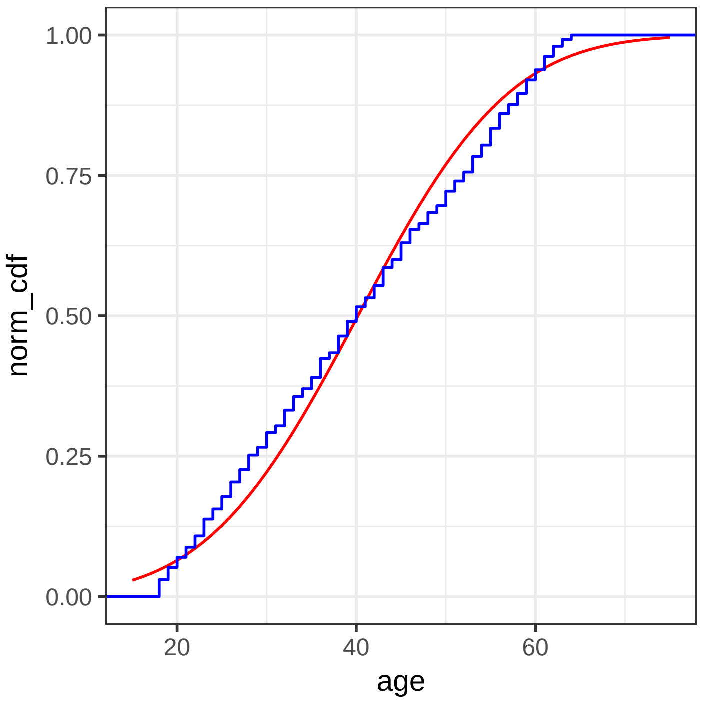
[3]Data: https://raw.githubusercontent.com/Rmadillo/SCH_R_Training/master/Diff_Inf/Shurtleffetal2015_episurg_data.csv; metadata: https://github.com/Rmadillo/SCH_R_Training/blob/master/Diff_Inf/README.md.



```r
# quantiles for 2.5, 33, 67, 95
quantile(sbp$age, probs = c(0.025, 0.33, 0.67, 0.975))
```

```
##  2.5%   33%   67% 97.5%
##    18    32    48    62
```

```r
# ecdf with normal cdf overplotted
age_cdf = data.frame(x = seq(15, 75, 1), norm_cdf = pnorm(seq(15, 75, 1),
                     mean = mean(sbp$age), sd = sd(sbp$age)))

ggplot(sbp, aes(x = age)) +
    geom_line(data = age_cdf, aes(x = x, y = norm_cdf), color = "red") +
    stat_ecdf(color="blue") +
    theme_bw()
```



So just by looking at the distribution itself, we can see that our (implied normally-distributed) summary stats are lying to us, and seeing the quantiles as well as over-plotting the normal curve confirms that.

What if we only saw the summary stats in a paper, as would be the typical case? We'd have to take it on faith that the variable's distribution would at least approximate a normal curve. In some cases, it's irrelevant, e.g., perhaps age in a genetic study. In others, it's essential, e.g., perhaps age in pediatric neurosurgery. A little critical thought is required to determine the relative importance of summary stats in *Table 1* to the study and your intended use of that study.

What would be an alternative to this? Of course, some authors use medians and interquartile (IQR) ranges for non-normal data, which is a great alternative, but we then lose the inferential power that summary statistics can provide when a particular distribution matches our data, such as in the Poisson example, above. On the other hand, that adds a level of complexity to a publication that may not be necessary or warranted.

Ideally, one would like to see a more modern *Table 1*, where you see summary statistics alongside a sparkline-type of mini-graph in the table as well, something that compactly shows the observed distribution. Unfortunately, it is not currently easy to do this in an automated fashion entirely inside R. There are a few workarounds possible, including creating the graphs, saving them to disk, then loading the saved pictures inside the table, as seen below.

```r
# Load dplyr for its filter command (like SQL's "where")
library(dplyr)

# Create ggplot objects and save to local file
p1 = ggplot(filter(sbp, Treatment_Group == "Control"), aes(x = age)) +
  geom_histogram(fill="darkblue", color = "white", size = 0.1, binwidth = 5) +
  xlim(10, 70) +
  theme_void()
ggsave("p1.png", p1, width = 0.25, height = 0.15, units = "in")

p2 = ggplot(filter(sbp, Treatment_Group == "Treatment"), aes(x = age)) +
  geom_histogram(fill="darkblue", color = "white", size = 0.1, binwidth = 5) +
  xlim(10, 70) +
  theme_void()
ggsave("p2.png", p2, width = 0.25, height = 0.15, units = "in")

p3 = ggplot(filter(sbp, Treatment_Group == "Control"), aes(x = age)) +
  geom_density(fill="darkblue", color = "white") +
  xlim(10, 70) +
  theme_void()
ggsave("p3.png", p3, width = 0.25, height = 0.15, units = "in")

p4 = ggplot(filter(sbp, Treatment_Group == "Treatment"), aes(x = age)) +
  geom_density(fill="darkblue", color = "white") +
  xlim(10, 70) +
  theme_void()
ggsave("p4.png", p4, width = 0.25, height = 0.15, units = "in")

p5 = ggplot(filter(sbp, Treatment_Group == "Control"), aes(x = age)) +
  geom_histogram(aes(y = ..density..), color = "white", fill="darkblue",
                 size = 0.1, binwidth = 5) +
  geom_density(fill="#DEEBF7", color = "black", size = 0.1, alpha = 0.3) +
  xlim(10, 70) +
  theme_void()
ggsave("p5.png", p5, width = 0.25, height = 0.15, units = "in")
```

```r
p6 = ggplot(filter(sbp, Treatment_Group == "Treatment"), aes(x = age)) +
  geom_histogram(aes(y = ..density..), color = "white", fill="darkblue",
                 size = 0.1, binwidth = 5) +
  geom_density(fill="#DEEBF7", color = "black", size = 0.1, alpha = 0.2) +
  xlim(10, 70) +
  theme_void()
ggsave("p6.png", p6, width = 0.25, height = 0.15, units = "in")

p7 = ggplot(filter(sbp, Treatment_Group == "Control"),
            aes(x = factor(1), fill = Exercise)) +
  geom_bar() +
  theme_void() +
  theme(legend.position = "none") +
  scale_fill_brewer() # 3 colors hex codes: brewer.pal(3, "Blues")
ggsave("p7.png", p7, width = 0.10, height = 0.25, units = "in")

p8 = ggplot(filter(sbp, Treatment_Group == "Treatment"),
            aes(x = factor(1), fill = Exercise)) +
  geom_bar() +
  theme_void() +
  theme(legend.position = "none") +
  scale_fill_brewer()
ggsave("p8.png", p8, width = 0.10, height = 0.25, units = "in")

p9 = ggplot(filter(sbp, Treatment_Group == "Control"),
            aes(x = factor(1), fill = Exercise)) +
  geom_bar(width = 1) +
  theme_void() +
  theme(legend.position = "none") +
  coord_polar(theta = "y") +
  scale_fill_brewer()
ggsave("p9.png", p9, width = 0.5, height = 0.5, units = "in")

p10 = ggplot(filter(sbp, Treatment_Group == "Treatment"),
             aes(x = factor(1), fill = Exercise)) +
  geom_bar(width = 1) +
  theme_void() +
  theme(legend.position = "none") +
  coord_polar(theta = "y") +
  scale_fill_brewer()
ggsave("p10.png", p10, width = 0.5, height = 0.5, units = "in")
```

The `tableone` package also has some useful summary functions, which we'll use to create a more "advanced" *Table 1*.

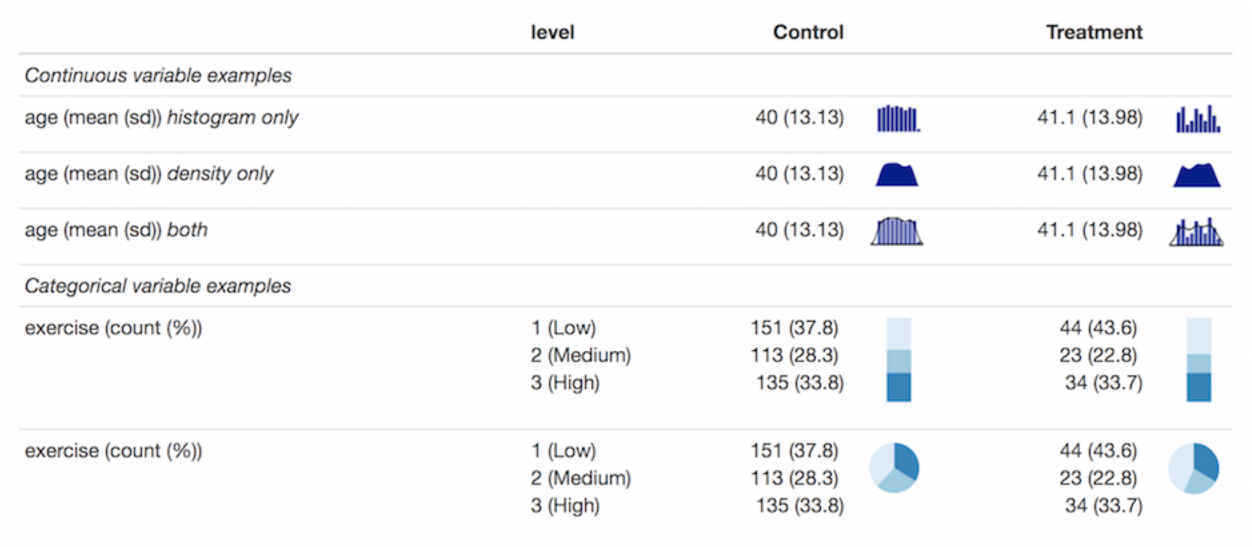
```r
# You can see the summary stats and object structure using summary, but str helps
# Not run: str(table_1_lies)
summary(table_1_lies)
```

```
##           Length Class     Mode
## ContTable 2      ContTable list
## CatTable  2      CatTable  list
## MetaData  5      -none-    list
```

And with these pieces inserted into an `rmarkdown` table, it might look something like this (see the `.Rmd` file for a complete view of the table code):

```
|  | level | Control |     | Treatment |     |
|------------------------|:-------|--------:|:--:|----------:|:--:|
| *Continuous variable examples*  |   |   |   |   |   |
| age (mean (sd)) *histogram only* | | `r round(table_1_lies$ContTable$Control[1,4], 1)`  (`r round(tab
| age (mean (sd)) *density only* | | `r round(table_1_lies$ContTable$Control[1,4], 1)`  (`r round(table
| age (mean (sd)) *both* |  | `r round(table_1_lies$ContTable$Control[1,4], 1)`  (`r round(table_1_lies
| *Categorical variable examples*  |   |   |   |   |   |
| exercise (count (%)) | 1 (Low) <br> 2 (Medium) <br> 3 (High) | `r table_1_lies$CatTable$Control$exerc
| exercise (count (%)) | 1 (Low) <br> 2 (Medium) <br> 3 (High) | `r table_1_lies$CatTable$Control$exerc
```

Table 1. This is a test. This is only a test. Do not adjust your set.



	level	Control		Treatment	
Continuous variable examples					
age (mean (sd)) *histogram only*		40 (13.13)		41.1 (13.98)	
age (mean (sd)) *density only*		40 (13.13)		41.1 (13.98)	
age (mean (sd)) *both*		40 (13.13)		41.1 (13.98)	
Categorical variable examples					
exercise (count (%))	1 (Low)<br>2 (Medium)<br>3 (High)	151 (37.8) 113 (28.3) 135 (33.8)		44 (43.6) 23 (22.8) 34 (33.7)	
exercise (count (%))	1 (Low)<br>2 (Medium)<br>3 (High)	151 (37.8) 113 (28.3) 135 (33.8)		44 (43.6) 23 (22.8) 34 (33.7)	

It's an open question on how and when such advances might make it into medical journals.

Table 1 has been a staple of medical research for decades. Hopefully, you'll treat them with a more critical eye now that you've seen ways it can be either helpful or harmful. But perhaps most importantly, the possible discrepancies between summary statistics and the actual data should illustrate how important it is to do EDA work *every time* you analyze data.

Exercise and Homework Answers (code in the .Rmd file)

Exercise R1

1. Create a histogram/density plot of the `sbp` variable using `ggplot2`, using a custom color fill for density (try #A30134).
2. Create a horizontal bar plot of the `Salt` variable, ordered with the highest value closest to the x-axis.

Exercise R2

1. Create a table of counts for the `Salt` variable that includes a sum value.
2. Use the `describe` function to obtain summary stats for the `sbp` variable only.
3. Plot the mean, median, and mode as vertical lines of different colors on a histogram + density ggplot for the `sbp` variable.

Extra credit: replace `describe` with `multi.hist` (also from the `psych` package) in the code chunk that introduces the `psych` package above, and see what happens.

Homework (using the neurosurg data)

1. Use either a data set of your choice (perhaps something you're working on now) or data on post-neurosurgery cognitive outcomes available on GitHub to obtain summary stats and a visual of the distribution of at least *one numeric* and/or *one categorical* variable.
2. Create a simple *Table 1* summary from your own data or from the neurosurgery data linked in question 1.
3. *Extra credit:* create an "advanced" *Table 1* (see below) from your own data or from the neurosurgery data linked in question 1, using `Phase` as the strata.

~ *End of file* ~