Module 3: univariate analysis

BIOSTAT 213: Introduction to Computing in the R Software Environment

Yea-Hung Chen, PhD, MS

Monday, October 7, 2019

Summarizing continuous data

Sums

- total weight, in kg:

```
> sum(ii$bmxwt)
[1] 348645.5
```

Mean, standard deviation, and variance

```
> mean(ii$ridageyr)
[1] 42.93751
> sd(ii$ridageyr)
[1] 15.16127
> var(ii$ridageyr)
[1] 229.8642
```

Mean, standard deviation, and variance

- first measurement of systolic blood pressure:

```
> mean(ii$bpxsy1)
[1] NA
> sd(ii$bpxsy1)
[1] NA
```

Mean, standard deviation, and variance

```
> mean(ii$bpxsy1,na.rm=TRUE)
[1] 122.6199
> sd(ii$bpxsy1,na.rm=TRUE)
[1] 16.86299
```

- not all data-analysis functions have an `na.rm` argument
- be sure to check help pages
- more on missing values on a later day

Mean, standard deviation, and variance

```
> var(ii$ridageyr)
[1] 229.8642
> sum((ii$ridageyr-mean(ii$ridageyr))^2)/(length(ii$ridageyr)-1)
[1] 229.8642
```

- `sd()` and `var()` use a denominator of $n - 1$
- more on this in your statistics classes

Quantiles

```
> median(ii$ridageyr)
[1] 43
> quantile(ii$ridageyr,probs=0.5)
50%
 43
> quantile(ii$ridageyr,0.5)
50%
 43
```

Quantiles

```
> quantile(ii$ridageyr,probs=c(0.25,0.5,0.75))
25% 50% 75%
 30  43  56
> quantile(ii$ridageyr,c(0.25,0.5,0.75))
25% 50% 75%
 30  43  56
```

Quantiles

```
> quantile(ii$ridageyr,seq(0.1,0.9,0.1))
10% 20% 30% 40% 50% 60% 70% 80% 90%
 22  27  32  37  43  48  54  59  64
```

Quantiles

- to remove the labels, use `unname()` or `as.numeric()`:

```
> unname(quantile(ii$ridageyr,seq(0.1,0.9,0.1)))
[1] 22 27 32 37 43 48 54 59 64
> as.numeric(quantile(ii$ridageyr,seq(0.1,0.9,0.1)))
[1] 22 27 32 37 43 48 54 59 64
```

Quantiles

- this highlights that numeric object can have names:

```
> class(quantile(ii$ridageyr,seq(0.1,0.9,0.1)))
[1] "numeric"
> names(quantile(ii$ridageyr,seq(0.1,0.9,0.1)))
[1] "10%" "20%" "30%" "40%" "50%" "60%" "70%" "80%" "90%"
```

Minimums and maximums

```
> min(ii$ridageyr)
[1] 18
> max(ii$ridageyr)
[1] 69
```

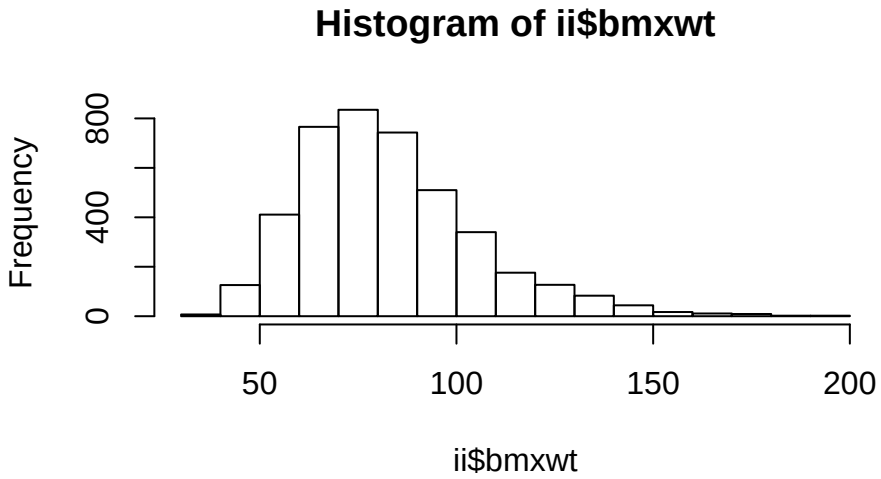
Summaries

```
> summary(ii$ridageyr)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  18.00   30.00   43.00   42.94   56.00   69.00
> summary(ii$bpxsy1)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.    NA's
   82.0   112.0   120.0   122.6   132.0   236.0     186
```

Histograms

```
> hist(ii$bmxwt)
```

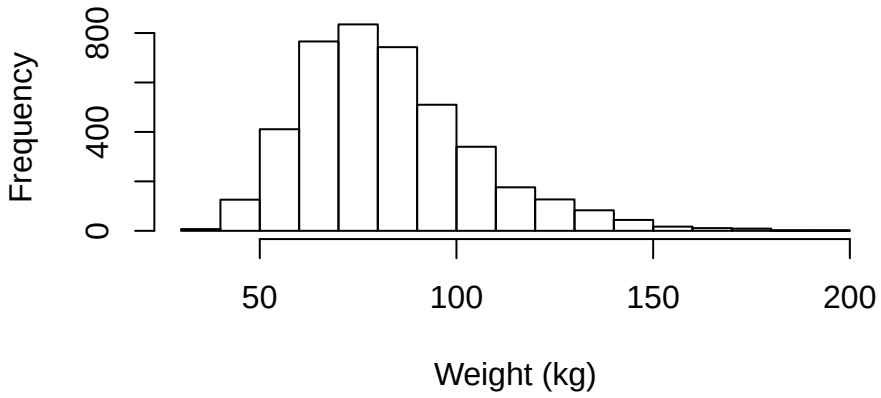
Histograms



Histograms

```
> hist(ii$bmxwt,main='',xlab='Weight (kg)')
```

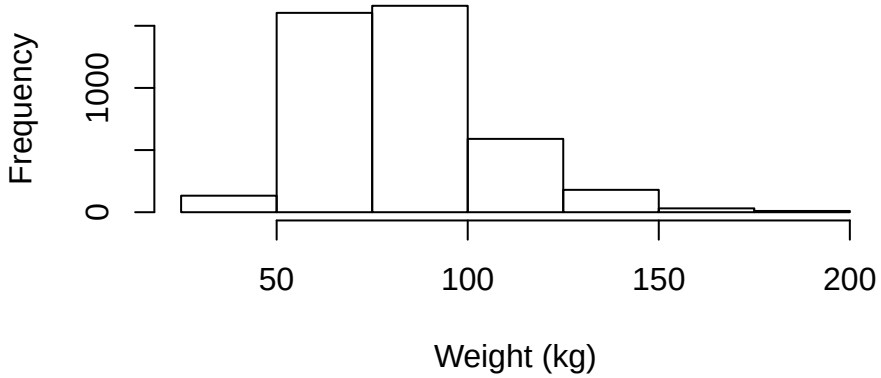
Histograms



Histograms

```
> hist(ii$bmxwt,breaks=seq(25,200,25),main='',xlab='Weight (kg)')
```

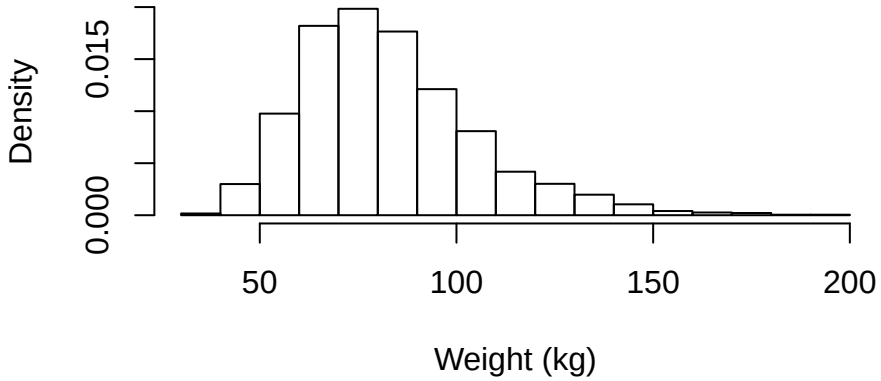
Histograms



Histograms

```
> hist(ii$bmxwt,freq=FALSE,main='',xlab='Weight (kg)')
```

Histograms



Histograms

- we'll discuss an alternate method for creating histograms later today

Factors

Factors

```
> head(ii$gender)
[1] Male   Male   Female Female Male   Female
Levels: Female Male
> class(ii$gender)
[1] "factor"
```

Factors

- a factor is an object that has a limited set of possible values
- the possible values are known as levels
- the levels have order, and it is possible to change the order of the levels
- these orders can be useful for plots and for regression models
- factors are useful for storing categorical variables

Factors

```
> head(ii$gender)
[1] Male   Male   Female Female Male   Female
Levels: Female Male
> class(ii$gender)
[1] "factor"
> levels(ii$gender)
[1] "Female" "Male"
```

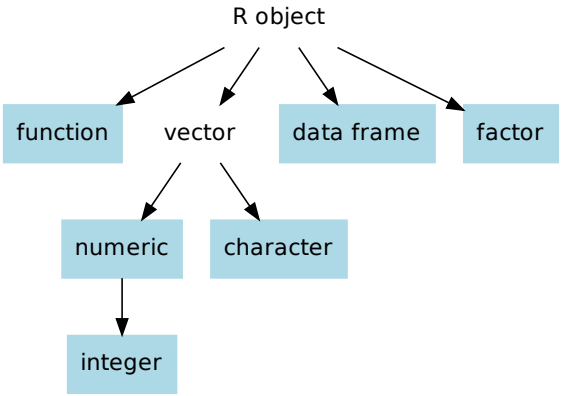
- gender has 2 levels: Female and Male
- the first level is Female and the second level is Male

Factors

```
> length(ii$gender)
[1] 4209
> is.vector(ii$gender)
[1] FALSE
```

- minor note: factors are not vectors, even though they look like vectors

Factors



Reordering levels

- to change the first level, use `relevel()`:

```
> levels(ii$gender)
[1] "Female" "Male"
> ii$gender<-relevel(ii$gender,ref='Male')
> levels(ii$gender)
[1] "Male"   "Female"
```

- the `ref` argument is the "reference": the level we want to be first
- all other levels are shifted over

Reordering levels

```
> levels(ii$gender)
[1] "Male"   "Female"
> ii$gender<-relevel(ii$gender,'Female')
> levels(ii$gender)
[1] "Female" "Male"
```

Reordering levels

```
> levels(ii$bmicat)
[1] "Class I obesity"                "Class II obesity"
[3] "Class III obesity"              "Overweight"
[5] "Underweight or normal weight"
```

- for greater control of level order, use `factor()`:

```
ii$bmicat<-factor(ii$bmicat,
                  levels=c('Underweight or normal weight',
                           'Overweight','Class I obesity',
                           'Class II obesity',
                           'Class III obesity'))
```

Reordering levels

```
> levels(ii$bmicat)
[1] "Underweight or normal weight" "Overweight"
[3] "Class I obesity"              "Class II obesity"
[5] "Class III obesity"
```

Renaming levels

```
> levels(ii$hs)
[1] "History of smoking" "No"
> levels(ii$hs)<-c('Smoking','No')
> levels(ii$hs)
[1] "Smoking" "No"
```

- this should be done cautiously

Factor versus character

- factor objects are useful for categorical variables (for example, US state)
- character objects are more useful for free response (for example, street address)

`read.csv()` and factors

- by default, if a variable in the data file has a character string in it, `read.csv()` will treat the variable as a factor
- to override that default, use the `stringsAsFactors=FALSE` argument
- the default ordering of the levels is alphabetical

Factor to character

```
> class(ii$bmicat)
[1] "factor"
> levels(ii$bmicat)
[1] "Underweight or normal weight" "Overweight"
[3] "Class I obesity"              "Class II obesity"
[5] "Class III obesity"
> ii$bmicat<-as.character(ii$bmicat)
> class(ii$bmicat)
[1] "character"
```

Character to factor

```
> ii$bmicat<-as.factor(ii$bmicat)
> class(ii$bmicat)
[1] "factor"
> levels(ii$bmicat)
[1] "Class I obesity"                "Class II obesity"
[3] "Class III obesity"              "Overweight"
[5] "Underweight or normal weight"
```

- notice that `as.character()` put the levels in alphabetical order again

Counts and percents

Counts

- categories of blood pressure:

```
> table(ii$bpcat)

    Elevated Hypertension       Normal
         743         1518         1762
```

Counts

```
> table(ii$bpcat,useNA='always')

    Elevated Hypertension        Normal          <NA>
              743                1518            1762             186
```

- use the `useNA='always'` argument to show missing values
- not all data-analysis functions have an `useNA` argument

Counts

- `table()` works with numeric and character objects, not just factors:

```
> class(ii$phq9)
[1] "integer"
> table(ii$phq9)

  1   2   3   4   5   6   7   8   9  10  11  12  13  14  15  16
622 565 358 277 223 173 139  93  97  67  57  45  39  27  22  16
 17  18  19  20  21  22  23  24  25  26  27  28  29  30
 21  14  15  15  14  10   7  11   5   1   7   1   2   1
```

Percents

```
> prop.table(table(ii$bpcat))

    Elevated Hypertension       Normal
   0.1846880    0.3773304    0.4379816
```

- you must include `table()` inside `prop.table()`

Percents

```
> prop.table(table(ii$bpcat))*100

    Elevated Hypertension       Normal
    18.46880     37.73304     43.79816
```

Univariate hypothesis tests

Univariate *t*-tests

```
> t.test(ii$bmxbmi,mu=23)
```

- use the `mu` argument to specify the null hypothesis
- in other words, this tests the null hypothesis that the mean body mass index (BMI) among American adults is 23 kg/m^2

Univariate *t*-tests

```
> t.test(ii$bmxbmi,mu=23)

    One Sample t-test

data:  ii$bmxbmi
t = 58.652, df = 4208, p-value < 2.2e-16
alternative hypothesis: true mean is not equal to 23
95 percent confidence interval:
 29.42332 29.86758
sample estimates:
mean of x
 29.64545
```

Univariate *t*-tests

```
> tt<-t.test(ii$bmxbmi,mu=23)
> names(tt)
[1] "statistic"   "parameter"   "p.value"     "conf.int"
[5] "estimate"    "null.value"  "alternative" "method"
[9] "data.name"
```

- it is often useful to save hypothesis tests
- the `names()` function provides an easy way to see accessible parts of the stored object

Univariate *t*-tests

```
> names(tt)
[1] "statistic"   "parameter"   "p.value"     "conf.int"
[5] "estimate"    "null.value"  "alternative" "method"
[9] "data.name"
> tt$statistic
       t
58.65219
> tt$parameter
  df
4208
> tt$p.value
[1] 0
```

Univariate *t*-tests

```
> tt$estimate
mean of x
 29.64545
> tt$conf.int
[1] 29.42332 29.86758
attr(,"conf.level")
[1] 0.95
```

Univariate χ^2-tests

```
> prop.table(table(ii$hs))

  Smoking        No
0.4011891 0.5988109
```

Univariate χ^2-tests

```
> chisq.test(table(ii$hs),p=c(0.5,0.5))
```

- use the p argument to specify the null hypothesis
- this tests the null hypothesis that 50% of American adults have a history of smoking

Univariate χ^2-tests

```
> chisq.test(table(ii$hs),p=c(0.5,0.5))

    Chi-squared test for given probabilities

data:  table(ii$hs)
X-squared = 164.22, df = 1, p-value < 2.2e-16
```

Univariate χ^2-tests

```
> tt<-chisq.test(table(ii$hs),p=c(0.5,0.5))
> names(tt)
[1] "statistic" "parameter" "p.value"   "method"    "data.name"
[6] "observed"  "expected"  "residuals" "stdres"
> tt$statistic
X-squared
 164.2238
> tt$parameter
df
 1
> tt$p.value
[1] 1.351685e-37
```

Installing and using packages

Installing packages

- as mentioned in the first week, there are many, many, many add-on packages available for R

Installing packages

- ggplot2 is a popular package for statistical plots
- to install the ggplot2 package:

```
> install.packages('ggplot2')
```

- you should only need to do this once ever

Installing packages

- to specify the repository, use the `repos` argument:

```
> install.packages('ggplot2',repos='https://cran.rstudio.com')
```

- the repository is the server
- they all host the same packages (they are mirrors of each other)
- other possible repositories include:
  - `https://cran.cnr.berkeley.edu`
  - `http://cran.stat.ucla.edu`
- a full list is available here

Using packages

- to use the ggplot2 package:

```
> library(ggplot2)
```

- you only need to do this once per session

Introduction to ggplot2

The ggplot2 approach

- ggplot2 uses a modular approach to build plots
- the basic form of ggplot2 code requires 2 parts:

variables or other dimensions + type of plot

Histograms

- let's focus first on the first part, which uses the `ggplot()` function:

```
> gg<-ggplot(data=ii,mapping=aes(x=bmxwt))
```

- the `data` argument specifies the data
- the `mapping` argument specifies the variables
- the `mapping` argument uses the `aes()` function, short for aesthetics
- here, we are saying that we want our `x` variable to be `bmxwt`

Histograms

```
> gg<-ggplot(data=ii,mapping=aes(x=bmxwt))
```

- the object assignment saves the initial plot
- as with other examples, gg is an arbitrary name
- the object assignment is optional, but allows for brevity and flexibility later

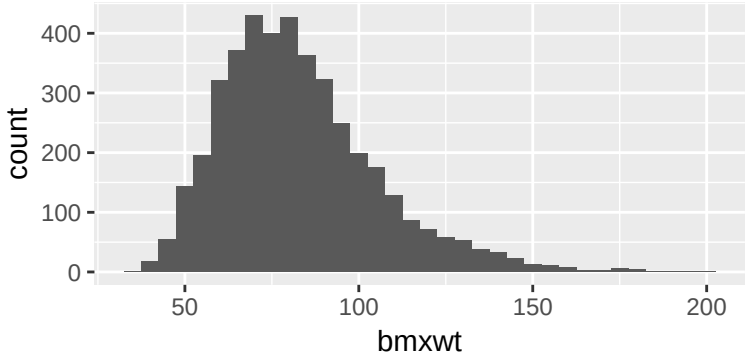
Histograms

- we're now ready to add the second part:

```
> gg+geom_histogram(binwidth=5)
```

- the addition of the `geom_histogram()` to the initial plot, gg, indicates that we want a histogram
- the `binwidth` argument specifies the bin widths

Histograms



Histograms

- previously, saving the ggplot object (the first part of the 2-part form):

```
gg<-ggplot(data=ii,mapping=aes(x=bmxwt))
gg+geom_histogram(binwidth=5)
```

- equivalently, without first saving the first part:

```
ggplot(data=ii,mapping=aes(x=bmxwt))+
  geom_histogram(binwidth=5)
```

- saving the ggplot object can be useful for easy modification, however

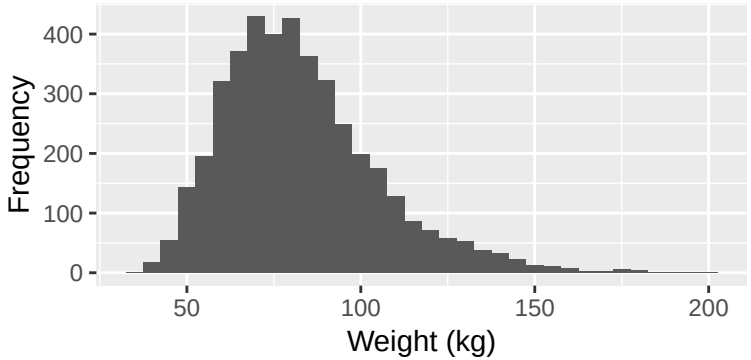
Histograms

- specifying labels for the axes:

```
gg+
  geom_histogram(binwidth=5)+
  xlab('Weight (kg)')+
  ylab('Frequency')
```

- it's not necessary to put each part of the sum on a new line

Histograms



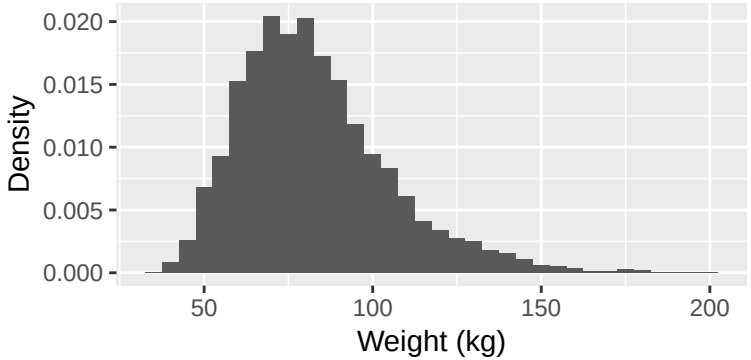
Histograms

- for density on the y axis instead of counts, we can update the mapping within `geom_histogram()`, using `aes()`:

```
gg+
  geom_histogram(aes(y=..density..),binwidth=5)+
  xlab('Weight (kg)')+
  ylab('Density')
```

- `..density..` is a shortcut indicating we want the density scale

Histograms

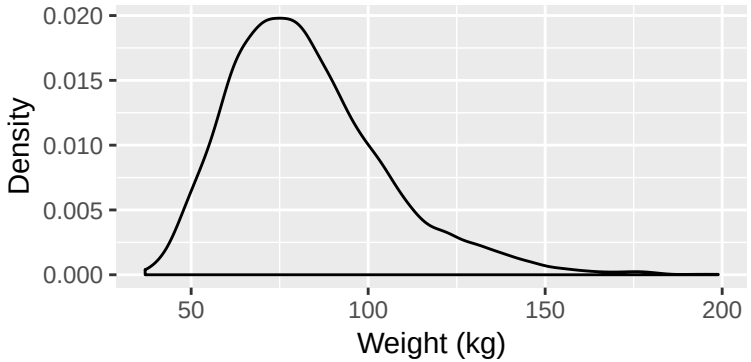


Density curves

- for density curves, use `geom_density()`:

```
gg+
  geom_density()+
  xlab('Weight (kg)')+
  ylab('Density')
```

Density curves

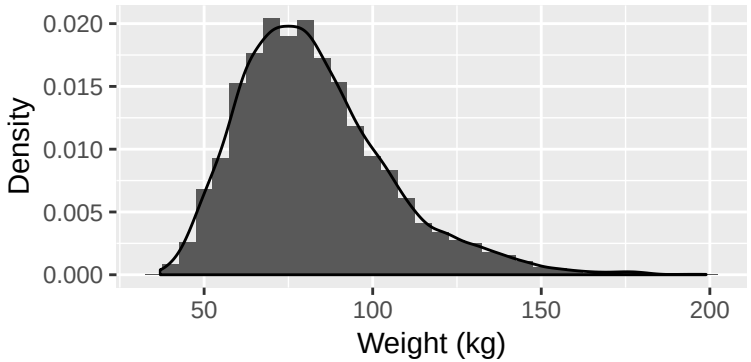


Density curves

- it's possible to combine visualization types:

```
gg+
  geom_histogram(aes(y=..density..),binwidth=5)+
  geom_density()+
  xlab('Weight (kg)')+
  ylab('Density')
```

Density curves



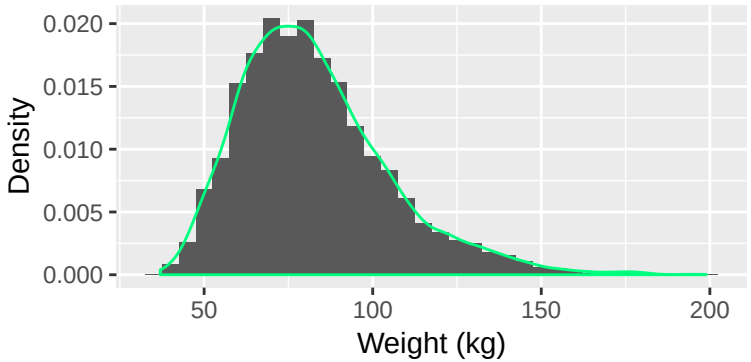
Density curves

- specifying color:

```
gg+
  geom_histogram(aes(y=..density..),binwidth=5)+
  geom_density(color='springgreen')+
  xlab('Weight (kg)')+
  ylab('Density')
```

- a list of predefined colors in R is here
- in general, however, I recommend minimizing manual specification of color in ggplot2, since the package will generally pick sensible defaults
- we'll see examples of that next week

Density curves



Exercise

Exercise

You may work in groups of 2–3. Save your code in a script.

1. Find the mean and standard deviation of high-density lipoprotein, HDL (`lbdhdd`).
2. Find the 20th, 40th, 60th, and 80th percentiles of HDL (`lbdhdd`).
3. Use `hist()` to produce a histogram of HDL (`lbdhdd`).

More on the next slide...

Exercise

4. Install ggplot2. Use `library()` to load the package.
5. Use ggplot2 to produce a histogram of HDL (`lbdhdd`).
6. Produce a histogram of HDL (`lbdhdd`).

More on the next slide...

Exercise

7. Reorder the levels of categories of HDL (`hdlcat`), using the following order: `Low`, `Medium`, and `High`.
8. Find the number of individuals in each `hdlcat` group.
9. Find the percent of individuals in each `hdlcat` group.

More on the next slide...

Exercise

10. Use a t-test to test the null hypothesis that the mean HDL (`lbdhdd`) in the population is equal to 60 mg/dL, the cutoff for high HDL.
11. Save the t-test above and write code that will output just the t statistic.

More on the next slide...

Exercise

12. The `round()` function rounds numbers. For example, `round(pi,3)` will round π to 3 decimal places. Using one line of code, use `round()` and `paste()` to generate the following summary of the test results: `t=164.2, df=1`. Here, we've rounded the t statistic to 1 decimal place.