Lab 1 BIS 505b

Maria Ciarleglio

2/8/2021

- Goal of Lab 1
- Data Preparation
    - Creating Categorical Variables
    - Creating Factor Variables
    - Subsetting a Data Frame (Rows)
    - Subsetting a Data Frame (Columns)
- Quantitative Variables
    - Numerical Summaries
        - Overall
        - By Group
    - Graphical Summaries
        - Overall
        - By Group
- Categorical Variables
    - Numerical Summaries
        - Overall
        - By Group
    - Graphical Summaries
        - Overall
        - By Group
- (Bonus) If there's time during Lab: `tableby()`
    - Quantitative Variables
    - Categorical Variables
    - All Variables

Goal of Lab 1

In **Lab 1**, we will review **(1)** important concepts in data management, **(2)** numerical and graphical summaries for quantitative variables, and **(3)** numerical and graphical summaries for categorical variables.

Data Preparation

Creating Categorical Variables

The first step when conducting a data analysis often involves some form of data management. This frequently involves data cleaning and creating new variables. Using the Framingham Heart Study data, we will begin by creating several categorical variables out of existing quantitative variables in the data set. Logical comparisons allow us to check conditions and logical operators allow us to check if at least one condition out of many is true (**or**, `|`) and check if multiple conditions are true (**and**, `&`).

1. Logical Comparisons

- `<` less than
- `>` greater than

- `<=` less than or equal to
- `>=` greater than or equal to
- `==` equal to
- `!=` not equal to
- `!x` not `x`
- `is.na()` is `NA`
- `!is.na()` is not `NA`

2. Logical Operators

- `|` or
- `&` and

The bracket operator `[ ]` can be used to choose elements of a vector that meet certain conditions. We use this method to identify subsets of a new variable that meet a condition and then assign those entries of that new variable a particular value.

- **`[ ]` Operator**

Creating categorical variables from existing variables often involves checking conditions using the existing variables based on the following logic:

```
newvariable[check condition using existing variable] <- value assigned to newvariable when condition is TRUE
```

For example, to create an Overweight/Obese indicator variable, we would use the existing variable `BMI`. BMI 25.0 to <30 is considered overweight and BMI 30.0 or higher is considered obese. If `BMI` is greater than or equal to 25, then the individual is flagged as being Overweight or Obese. Read the first line of code below as: the variable `OVERWEIGHTOBESE` in `fhs` equals 1 *where* `BMI >= 25`.

```
fhs$OVERWEIGHTOBESE[fhs$BMI >= 25] <- 1  # if BMI>=25 is true, OVERWEIGHTOBESE = 1
fhs$OVERWEIGHTOBESE[fhs$BMI < 25] <- 0   # if BMI<25 is true, OVERWEIGHTOBESE = 0
```

Similar logic can be used to create categorical variables with more than two categories. For example, let's create a new 4-level categorical variable `CIGPDAYGRP` describing number of cigarettes smoked/day: 0, 1-19, 20-39, and 40+. Let's also group BMI into the following categories:

- *Underweight*: BMI less than 18.5
- *Normal*: BMI between 18.5 and <25
- *Overweight*: BMI between 25.0 and <30
- *Obese*: BMI 30 or higher

```
# Grouping cigarettes smoked/day
fhs$CIGPDAYGRP[fhs$CIGPDAY == 0] = 0
fhs$CIGPDAYGRP[fhs$CIGPDAY >= 1 & fhs$CIGPDAY < 20] = 1   # 1-19 cigarettes/day
fhs$CIGPDAYGRP[fhs$CIGPDAY >= 20 & fhs$CIGPDAY < 40] = 2  # 20-39 cigarettes/day
fhs$CIGPDAYGRP[fhs$CIGPDAY >= 40] = 3                     # 40+ cigarettes/day

# Grouping BMI
fhs$BMIGRP[fhs$BMI < 18.5] = 0
fhs$BMIGRP[fhs$BMI >= 18.5 & fhs$BMI < 25] = 1
fhs$BMIGRP[fhs$BMI >= 25 & fhs$BMI < 30] = 2
fhs$BMIGRP[fhs$BMI >= 30] = 3
table(fhs$BMIGRP, useNA = "ifany")   # `useNA = "ifany"` displays NA category
```

```
##
##    0    1    2    3 <NA>
##   57 1936 1845  577   19
```

- **If-Else**

The `ifelse()` function provides a nice shortcut for creating a dichotomous variable based on a condition that is either TRUE or FALSE, as in the case of the variable `OVERWEIGHTOBESE` (i.e., is `BMI >= 25` TRUE or FALSE). The `ifelse(condition checked, value if TRUE, value if FALSE)` function checks the condition specified in the first argument and the function returns the value specified in the second argument if the condition is true. Otherwise, if the condition is false, the function returns the value specified in the third argument.

```
# Equivalent method of coding Overweight/Obese indicator
fhs$OVERWEIGHTOBESE <- ifelse(fhs$BMI >= 25, 1, 0)  # if BMI>=25 is true, OVERWEIGHTOBESE = 1
                                                    # if BMI>=25 is false, OVERWEIGHTOBESE = 0
```

Thankfully, the `ifelse` statement is smart enough to know that we do not want to assign subjects with a missing value of `BMI` an `OVERWEIGHTOBESE` value of 0. `BMI` contains 19 missing values, and the table of `OVERWEIGHTOBESE` shows us that all of these subjects also have a missing value for the `OVERWEIGHTOBESE` indicator variable.

```
summary(fhs$BMI)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.    NA's
##   15.54   23.09   25.45   25.85   28.09   56.80      19
```

```
table(fhs$OVERWEIGHTOBESE, useNA = "ifany")
```

```
##
##    0    1 <NA>
## 1993 2422   19
```

Creating Factor Variables

Below, we are using the data key that was provided for the Framingham Heart Study data set to demonstrate how to turn a categorical variable into a **factor variable**. In **R**, factor variables are categorical variables that can be either character or numeric. All of our categorical variables are coded numerically, so we need to explicitly tell **R** that these variables are categorical. Most of the existing categorical variables in the `fhs` data set are dichotomous Yes/No variables coded numerically as 1/0.

To create a factor variable, we use the `factor()` function. We can also attach labels to the `levels` (i.e., values) of the categorical variables. In addition to creating factor versions of the original variables in `fhs` , we will also turn our newly-created categorical variables, `CIGPDAYGRP` and `OVERWEIGHTOBESE` into factor variables. Ordered factor variables (i.e., ordinal variables) are created using the `ordered=TRUE` option in the `factor()` function. Equivalently, we may use the `ordered()` function instead of the `factor()` function when creating ordinal variables. The `ordered()` function has the same arguments as the `factor()` function. We will apply the `ordered()` option to `CIGPDAYGRP` and `BMIGRP` .

The `dplyr` package includes several functions for selecting, filtering, grouping, and arranging data. The `mutate()` function in `dplyr` is useful when creating more than one factor variable in a data frame. Below, rather than overwrite the existing numerical versions of the categorical variables in the data frame `fhs` , we create duplicate factor version of

each variable using the naming convention `variablename_factor`. Creating a factor-version is optional; you can instead over-write the existing numeric version of each variable with its factor version (e.g., over-write `SEX` coded numerically as {1, 2} with `fhs$SEX = factor(fhs$SEX, levels = c(1, 2), labels = c("Male", "Female")`.)

The `unique()` function returns the the unique values of a variable and can be used to check the different values a variable can take. Below, I'm checking that `SEX` is indeed coded numerically as 1 or 2 in the `fhs` data frame.

```
unique(fhs$SEX)
```

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

```
# fhs contains all of the variables in the original data frame, fhs, plus the factor variables below
fhs <- dplyr::mutate(fhs,
                     SEX_factor = factor(SEX,
                                 levels = c(1, 2),
                                 labels = c("Male", "Female")),
                     CURSMOKE_factor = factor(CURSMOKE,
                                      levels = c(0, 1),
                                      labels = c("No", "Yes")),
                     CVD_factor = factor(CVD,
                                 levels = c(0, 1),
                                 labels = c("No", "Yes")),
                     DEATH_factor = factor(DEATH,
                                   levels = c(0, 1),
                                   labels = c("No", "Yes")),
                     CIGPDAYGRP_factor = factor(CIGPDAYGRP,
                                        levels = c(0, 1, 2, 3),
                                        labels = c("0", "1-19", "20-39", "40+"),
                                        ordered = TRUE))
```

We can also create individual factor variables without using the `mutate()` function. Just be sure to add the factor variable to the `fhs` data frame using the `$` (i.e., `fhs$OVERWEIGHTOBESE_factor` is used to add the variable `OVERWEIGHTOBESE_factor` to `fhs`):

```
# Creating a factor not using the `mutate()` function
fhs$BMIGRP_factor <- factor(fhs$BMIGRP,
                            levels = 0:3,
                            labels = c("Underweight", "Normal", "Overweight", "Obese"),
                            ordered = TRUE)

fhs$OVERWEIGHTOBESE_factor <- factor(fhs$OVERWEIGHTOBESE,
                                     levels = c(0, 1),
                                     labels = c("No", "Yes"))
```

The `levels()` function returns the levels of a factor variable; `nlevels()` returns the number of levels.

```
levels(fhs$CIGPDAYGRP_factor)
```

```
## [1] "0"     "1-19"  "20-39" "40+"
```

```
nlevels(fhs$CIGPDAYGRP_factor)
```

```
## [1] 4
```

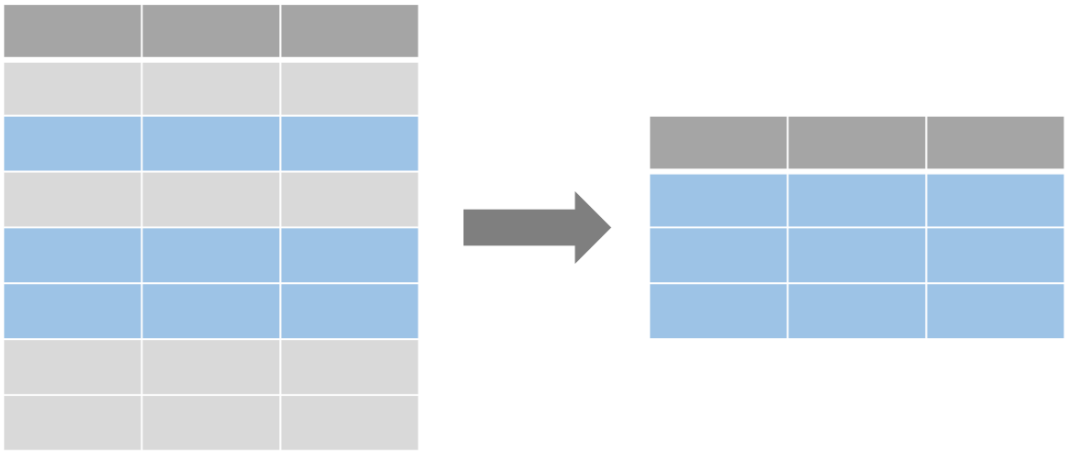
The `table()` function (discussed in detail in the section on **Categorical Variables**) can be used to check that the appropriate label is assigned to a given value or level using a cross-tabulation. It also provides a summary of the number of observations in each category.

```
table(fhs$CIGPDAYGRP_factor, fhs$CIGPDAYGRP)
```

```
##
##            0    1    2    3
##   0     2253    0    0    0
##   1-19     0  907    0    0
##   20-39    0    0 1077    0
##   40+      0    0    0  165
```

Subsetting a Data Frame (Rows)

The `filter()` function in the `dplyr` package allows us to easily subset or extract data frame rows based on certain criteria.



The `filter()` function extracts rows that meet all of the specified logical criteria.

1. Logical Comparisons

- `<` less than
- `>` greater than
- `<=` less than or equal to
- `>=` greater than or equal to
- `==` equal to
- `!=` not equal to
- `%in%` group membership. e.g., `x %in% c(2, 3)` means that `x` is equal to 2 or 3.
- `is.na()` is `NA`
- `!is.na()` is not `NA`

2. Logical Operators

- `|` or. `x == 2|3` means that the `x` is equal to 2 or (`|`) 3. `x %in% c(2, 3)` is equivalent to `x == 2|3` .
- `&` and. `sex == "female" & age > 25`

The `filter()` function uses the `%>%` (**pipe**) operator to feed what is on the left of the pipe to the operation on the right.

- **One Criterion**: Extract rows of `fhs` where `SEX_factor == "Male"` (notice the `==` when testing for equality). This subset is saved as a new data frame called `fhsM`. *Note*: the formatted label is used to check criteria involving a factor variable.

```
fhsM <- fhs %>% filter(SEX_factor == "Male")
dim(fhsM)
```

```
## [1] 1944   46
```

```
table(fhs$SEX_factor, useNA = "ifany", dnn = "Sex")    # original fhs data set (dnn= applies name(s)
to the table dimension(s) )
```

```
## Sex
##   Male Female
##   1944   2490
```

```
table(fhsM$SEX_factor, useNA = "ifany", dnn = "Sex")   # new subset data set fhsM
```

```
## Sex
##   Male Female
##   1944      0
```

Extract rows where `TOTCHOL` is missing (`NA`) and save these rows as a new data frame called `fhsTOTCHOLNA`.

```
fhsTOTCHOLNA <- fhs %>% filter(is.na(TOTCHOL))
summary(fhs$TOTCHOL)      # checking number of NAs
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.    NA's
##     107     206     234     237     264     696      52
```

```
dim(fhsTOTCHOLNA)         # number of rows should equal number of NAs
```

```
## [1] 52 46
```

Extract rows where `TOTCHOL` is *not* missing.

```
fhsTOTCHOLCOMPLETE <- fhs %>% filter(!is.na(TOTCHOL))
sum(!is.na(fhs$TOTCHOL))   # number of non-missing values of TOTCHOL in fhs
```

```
## [1] 4382
```

```
dim(fhsTOTCHOLCOMPLETE)    # number of rows should equal number of non-missing values of TOTCHOL in
 fhs
```

```
## [1] 4382   46
```

- **Two+ Criteria**: Multiple conditions are separated by a comma.

Extract rows of `fhs` where `SEX_factor == "Male"` and `AGE` between 40 and 50 (notice the `&` when checking the two conditions on `AGE`).

```
fhsM4050 <- fhs %>% filter(SEX_factor == "Male", AGE >= 40 & AGE <= 50)
dim(fhsM4050)
```

```
## [1] 814  46
```

```
head(fhsM4050)[,c("SEX_factor", "AGE")]   # print columns "SEX_factor" and "AGE" in data frame fhsM4
050
```

```
##   SEX_factor AGE
## 1       Male  48
## 2       Male  43
## 3       Male  46
## 4       Male  48
## 5       Male  41
## 6       Male  44
```

- **`[ ]` Operator and `subset()` function**

Although this section is focusing on `dplyr` package functions, recall that we discussed subsetting a data frame using using the `[ ]` operator and using the `subset()` function in the **R-Review** document (**Section III.5**). Again, notice that the formatted label is used when specifying the values of `SEX_factor` that are of interest. **Note of Caution**: When creating a subset using the bracket operator `[ ]` , **NAs are included in the subset**. This usually does not cause problems since we use the option `na.rm = TRUE` when producing numerical summary statistics. But it can cause some problems when producing certain frequency tables. To avoid these NA rows from being included when using the bracket operator `[ ]` , one option is apply the `which()` function within the bracket operator `[ ]` . `which()` returns the indices where the condition is TRUE; these indices are the row numbers extracted from `fhs` .

```
# Subset function
fhsM <- subset(fhs, SEX_factor == "Male")
dim(fhsM)
```

```
## [1] 1944   46
```

```
# Bracket operator
fhsM <- fhs[which(fhs$SEX_factor == "Male"),]
table(fhsM$SEX_factor, useNA = "ifany")
```

```
##
##   Male Female
##   1944      0
```

```
dim(fhsM)
```

```
## [1] 1944   46
```

Subsetting on more than one condition:

```
# Subset function
fhsM4050 <- subset(fhs, SEX_factor == "Male" & AGE >= 40 & AGE <= 50)
dim(fhsM4050)
```

```
## [1] 814  46
```

```
# Bracket operator
fhsM4050 <- fhs[which(fhs$SEX_factor == "Male" & fhs$AGE >= 40 & fhs$AGE <= 50),]
dim(fhsM4050)
```

```
## [1] 814  46
```

Exercise: Create the data frame `fhs2plus` that contains only those with `CIGPDAYGRP_factor` equal to `"40+"` . Check that you created `fhs2plus` correctly.

▶ Answer:

Subsetting a Data Frame (Columns)

The `select()` function in the `dplyr` package allows us to select variables (i.e., columns) in a data frame.

Below, we are selecting the outcome variables of interest from the `fhs` data frame and storing them in the new data frame `fhsoutcomes` :

```
fhsoutcomes <- fhs %>% select(CVD_factor, DEATH_factor)
head(fhsoutcomes)
```

```
##   CVD_factor DEATH_factor
## 1        Yes           No
## 2         No           No
## 3         No           No
## 4        Yes          Yes
## 5         No           No
## 6        Yes           No
```

Equivalently, using the `[ ]` operator (i.e., not using `dplyr` package):

```
# Bracket operator
fhsoutcomes2 <- fhs[,c("CVD_factor", "DEATH_factor")]
head(fhsoutcomes2)
```

```
##   CVD_factor DEATH_factor
## 1        Yes           No
## 2         No           No
## 3         No           No
## 4        Yes          Yes
## 5         No           No
## 6        Yes           No
```

Quantitative Variables

Quantitative variables are summarized using summary statistics such as mean, median, standard deviation, IQR, and range. We typically present summary statistics for the overall study sample and within subgroups of interest. Graphical summaries such as histograms and boxplots allow us to see the shape of the distribution of the variable and can also be presented by subgroup.

Numerical Summaries

Overall

The built-in functions below provide numerical summaries of quantitative variables. The option `na.rm = TRUE` is required in many of the functions if the variable analyzed contains missing values (NAs). The `na.rm = TRUE` option tells the function to remove NAs before performing any calculations.

Function	Description
`sum(!is.na(x))`	Number of non-missing values
`sum(is.na(x))`	Number of missing values
`mean(x)`	Mean; `na.rm = TRUE`
`sd(x)`	Standard deviation; `na.rm = TRUE`
`var(x)`	Variance; `na.rm = TRUE`
`min(x)`	Minimum; `na.rm = TRUE`
`max(x)`	Maximum; `na.rm = TRUE`

Function	Description
`median(x)`	Median; `na.rm = TRUE`
`range(x)`	Range of values (minimum and maximum); `na.rm = TRUE`
`quantile(x)`	0, 25, 50, 75 and 100th percentile; `quantile(x, probs = 0.25)` returns specific percentile; `type = 2` matches class method of computing Q_1 and Q_3; `na.rm = TRUE`
`IQR(x)`	Interquartile range ($Q_3 - Q_1$); `type =` option available, `na.rm = TRUE`
`summary(x)`	Minimum, Q_1, median, mean, Q_3, maximum, and number of missing values (if any); `quantile.type =` option available
`fivenum(x)`	Five number summary (minimum, Q_1, median, Q_3, maximum); `na.rm = TRUE`

```
sum(!is.na(fhs$TOTCHOL))
```

```
## [1] 4382
```

```
mean(fhs$TOTCHOL)
```

```
## [1] NA
```

```
mean(fhs$TOTCHOL, na.rm=TRUE)
```

```
## [1] 236.9843
```

```
sd(fhs$TOTCHOL, na.rm=TRUE)
```

```
## [1] 44.6511
```

```
var(fhs$TOTCHOL, na.rm=TRUE)
```

```
## [1] 1993.721
```

```
min(fhs$TOTCHOL, na.rm=TRUE)
```

```
## [1] 107
```

```
max(fhs$TOTCHOL, na.rm=TRUE)
```

```
## [1] 696
```

```
range(fhs$TOTCHOL, na.rm=TRUE)
```

```
## [1] 107 696
```

```
median(fhs$TOTCHOL, na.rm=TRUE)
```

```
## [1] 234
```

```
quantile(fhs$TOTCHOL, type=2, na.rm=TRUE)
```

```
##   0%  25%  50%  75% 100%
##  107  206  234  264  696
```

```
quantile(fhs$TOTCHOL, type=2, probs=0.75, na.rm=TRUE)  # request specific percentile with probs=
```

```
## 75%
## 264
```

```
IQR(fhs$TOTCHOL, na.rm=TRUE)
```

```
## [1] 58
```

```
summary(fhs$TOTCHOL)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.    NA's
##     107     206     234     237     264     696      52
```

```
fivenum(fhs$TOTCHOL, na.rm=TRUE)
```

```
## [1] 107 206 234 264 696
```

There is no built-in function to find the mode of a variable. However, we can simply find the most frequently occurring value of a variable by creating a table of its unique values using the `table()` function (discussed more in the section on **Categorical Variables**) and sorting the number of occurrences from highest to lowest using the sort()` function.

```
tab <- table(fhs$TOTCHOL)               # number of occurrences for each unique value
names(sort(tab, decreasing = TRUE)[1])  # sort highest to lowest, pick highest (i.e., first elemen
t)
```

```
## [1] "240"
```

Recall from **Lab 0** that it is possible to embed **R** code in a line of markdown text. For example:

> The average total cholesterol level at exam 1 in the 4382 subjects in the Framingham Heart Study is 236.98 mg/dL with a standard deviation 44.65. The range of cholesterol values is 107 - 696.

Manual pipe tables can also be combined with inline **R** code to create summary tables:

Variable	**Mean (SD)**
Age	49.93 (8.68)
BMI	25.85 (4.1)
Cigarettes smoked/day	8.97 (11.93)
Diastolic BP	83.08 (12.06)
Systolic BP	132.91 (22.42)

By Group

In addition to describing the overall sample, it is useful to compare data between two or more defined (categorical) groups. For example, we may be interested in comparing the characteristics above (`AGE` , `BMI` , `CIGPDAY` , etc.) in those who are overweight/obese vs. those who are not overweight/obese.

- **Option 1:** The `by()` function allows us to separate the data frame into subgroups defined by the second variable (`OVERWEIGHTOBESE_factor`) and applies the function specified (e.g., `mean` , `sd` , `var` , `min` , `median` , `max` , `summary`) to each subgroup.

```
by(fhs$AGE, fhs$OVERWEIGHTOBESE_factor, mean, na.rm=TRUE)
```

```
## fhs$OVERWEIGHTOBESE_factor: No
## [1] 48.63221
## ------------------------------------------------------------
## fhs$OVERWEIGHTOBESE_factor: Yes
## [1] 50.97688
```

```
by(fhs$AGE, fhs$OVERWEIGHTOBESE_factor, sd, na.rm=TRUE)
```

```
## fhs$OVERWEIGHTOBESE_factor: No
## [1] 8.613389
## ------------------------------------------------------------
## fhs$OVERWEIGHTOBESE_factor: Yes
## [1] 8.565386
```

```
by(fhs$AGE, fhs$OVERWEIGHTOBESE_factor, min, na.rm=TRUE)
```

```
## fhs$OVERWEIGHTOBESE_factor: No
## [1] 33
## ------------------------------------------------------------
## fhs$OVERWEIGHTOBESE_factor: Yes
## [1] 32
```

```
by(fhs$AGE, fhs$OVERWEIGHTOBESE_factor, summary)
```

```
## fhs$OVERWEIGHTOBESE_factor: No
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   33.00   41.00   47.00   48.63   55.00   69.00
## ------------------------------------------------------------
## fhs$OVERWEIGHTOBESE_factor: Yes
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   32.00   44.00   51.00   50.98   58.00   70.00
```

Exercise: Produce some summary statistics for `AGE` by `SEX_factor` in those who smoke 40+ cigarettes/day. Recall that we created `fhs2plus` in the previous exercise.

▶ Answer:

- **Option 2:** We can apply summary statistic functions on subsets using the bracket operator `[ ]` or the `subset()` function. The `filter()` function in the `dplyr` package also allows us to create the data frames for each subgroup.

`[ ]` operator:

```
# [ ] operator broken into two steps
grp1 = fhs[which(fhs$OVERWEIGHTOBESE_factor == "No"),]    # new data frame containing subgroup 1
mean(grp1$AGE, na.rm=TRUE)                                # compute mean using grp1 data frame
```

```
## [1] 48.63221
```

```
grp2 = fhs[which(fhs$OVERWEIGHTOBESE_factor == "Yes"),]   # new data frame containing subgroup 2
mean(grp2$AGE, na.rm=TRUE)                                # compute mean using grp2 data frame
```

```
## [1] 50.97688
```

```
# [ ] operator all in one step
mean(fhs$AGE[which(fhs$OVERWEIGHTOBESE_factor == "No")], na.rm=TRUE)
```

```
## [1] 48.63221
```

```
mean(fhs$AGE[which(fhs$OVERWEIGHTOBESE_factor == "Yes")], na.rm=TRUE)
```

```
## [1] 50.97688
```

```
# [ ] operator all in one step (use of which() is optional, here, since we drop missings with na.rm=
TRUE)
mean(fhs$AGE[fhs$OVERWEIGHTOBESE_factor == "No"], na.rm=TRUE)    # same answer as above (using which
())
```

```
## [1] 48.63221
```

```
mean(fhs$AGE[fhs$OVERWEIGHTOBESE_factor == "Yes"], na.rm=TRUE)   # same answer as above (using which
())
```

```
## [1] 50.97688
```

`subset()` function:

```
# subset function broken into two steps
grp1 = subset(fhs, OVERWEIGHTOBESE_factor == "No")   # new data frame containing subgroup 1
mean(grp1$AGE, na.rm=TRUE)                           # compute mean using grp1 data frame
```

```
## [1] 48.63221
```

```
grp2 = subset(fhs, OVERWEIGHTOBESE_factor == "Yes")  # new data frame containing subgroup 2
mean(grp2$AGE, na.rm=TRUE)                           # compute mean using grp2 data frame
```

```
## [1] 50.97688
```

```
# subset function all in one step
mean(subset(fhs, OVERWEIGHTOBESE_factor == "No")$AGE, na.rm=TRUE)
```

```
## [1] 48.63221
```

```
mean(subset(fhs, OVERWEIGHTOBESE_factor == "Yes")$AGE, na.rm=TRUE)
```

```
## [1] 50.97688
```

Graphical Summaries

Histograms and boxplots are the two primary graphical measures used to summarize quantitative variables.

Overall

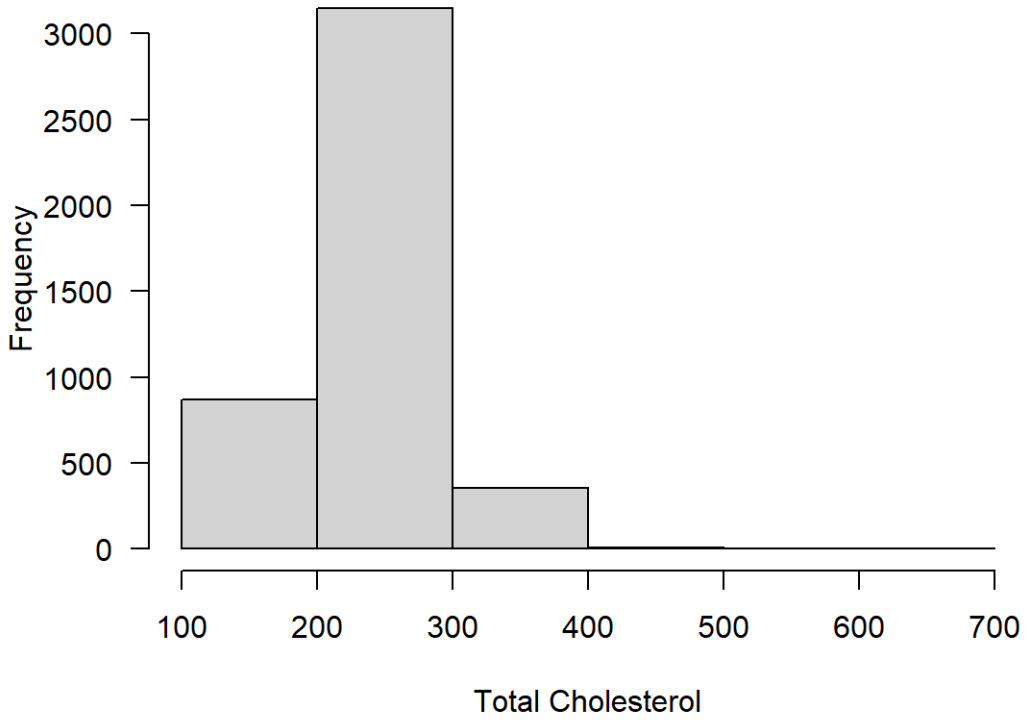
Histograms are created using the `hist()` function. By default, **R** displays the frequency on the y-axis (i.e., the number of observations in each bin). **R** automatically chooses the bin size and the bin breaks for the histogram. You can manually specify either the number of bins or a vector of the break points between histogram bins. By default, the bins

are **right-closed** (left open) intervals (i.e., (lower, upper]). Request `right = FALSE` gives **left-closed** intervals, [lower, upper). The option `las = 1` changes the orientation of the text on the y-axis so that it is perpendicular to the axis.

```
summary(fhs$TOTCHOL)  # helps us define the range of our axis
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.    NA's
##     107     206     234     237     264     696      52
```

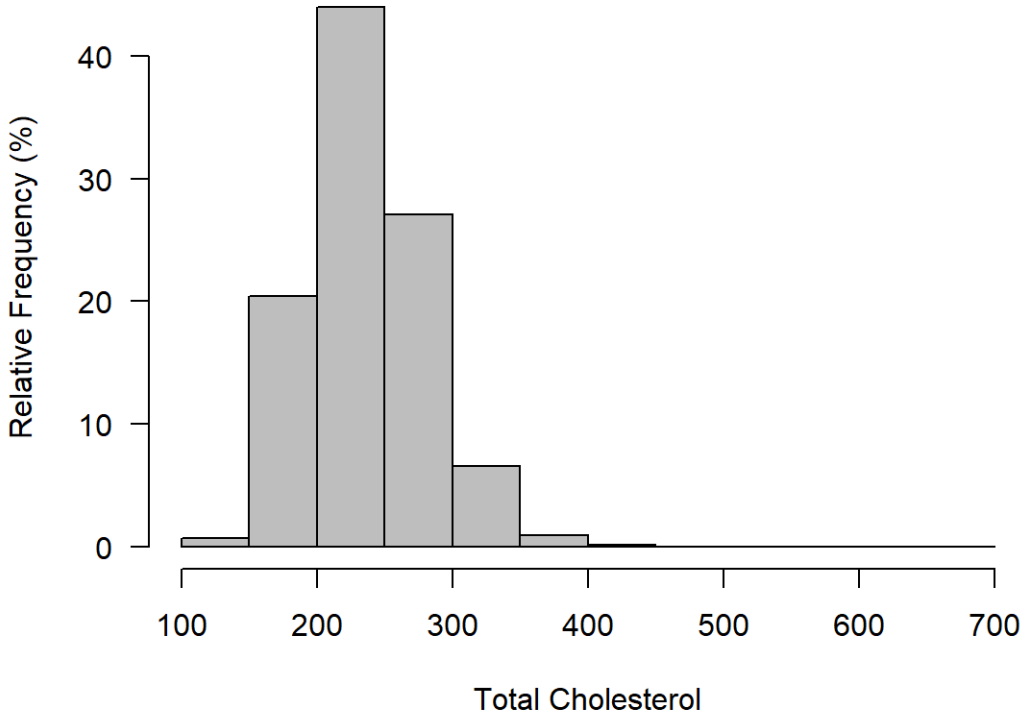
```
hist(fhs$TOTCHOL,
     breaks = seq(100, 700, by=100),
     right = FALSE,   # left-closed intervals (preferred)
     main = "",       # no figure title because we are requesting a figure caption in code chunk hea
der
     xlab = "Total Cholesterol",
     ylab = "Frequency",
     las = 1)         # rotate y-axis text
```



Total Cholesterol at Exam 1 in Framingham Heart Study

We can change the y-axis to display relative frequency by first saving the histogram to an object (called `h`, here). This will not plot the histogram because we specified `plot = FALSE` in the `hist()` function. `h` contains a vector called `counts` that gives the number of observations in each bin. We access this vector using `h$counts`. Next, we redefine `h$counts` as the relative frequency (`100 * h$counts / sum(h$counts)`) and plot the histogram by running `plot(h)`.

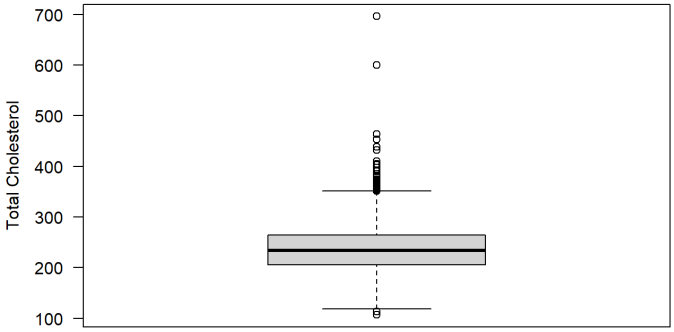
```
h <- hist(fhs$TOTCHOL, breaks = seq(100, 700, by=50), plot = FALSE)
h$counts <- 100 * h$counts / sum(h$counts)   # relative frequency
plot(h,
     main = "",        # no figure title
     xlab = "Total Cholesterol",
     ylab = "Relative Frequency (%)",
     col = "gray",     # bar colors
     las = 1)          # rotate y-axis text
```

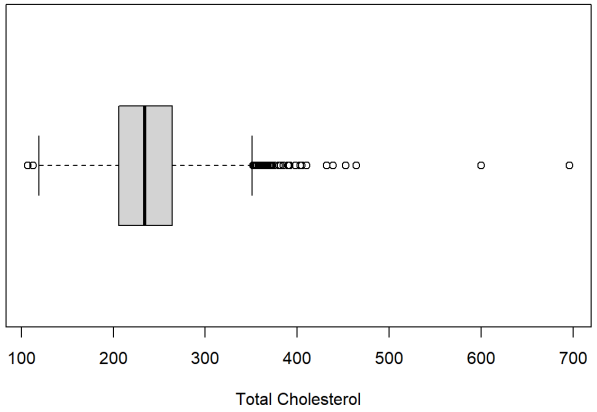


Total Cholesterol at Exam 1 in Framingham Heart Study

Boxplots are created using the `boxplot()` function. By default, vertical boxplots are produced. Request a horizontal boxplot using the `horizontal = TRUE` option.

```
boxplot(fhs$TOTCHOL,
        ylab = "Total Cholesterol",
        las = 1)

boxplot(fhs$TOTCHOL,
        xlab = "Total Cholesterol",
        horizontal = TRUE)
```





By Group

Creating plots for each subgroup allows us to visually compare the distributions. When plotting the data for each subgroup, it is important that the axes have the same range so that the graphs are directly comparable.

We begin by plotting **histograms** by group. We first create two data frames that contain non-overweight/obese participants `grp1` and overweight/obese participants `grp0`, respectively, using the `subset()` function. In order to set the range of the x-axis, we use the `range()` function to find the minimum and maximum of the quantitative variable being plotted (`TOTCHOL`, here). We will use this information to set the range of the x-axis when specifying the upper and lower limits of the the vector of histogram bin breaks (`breaks = c()`) in the histograms for each subgroup.

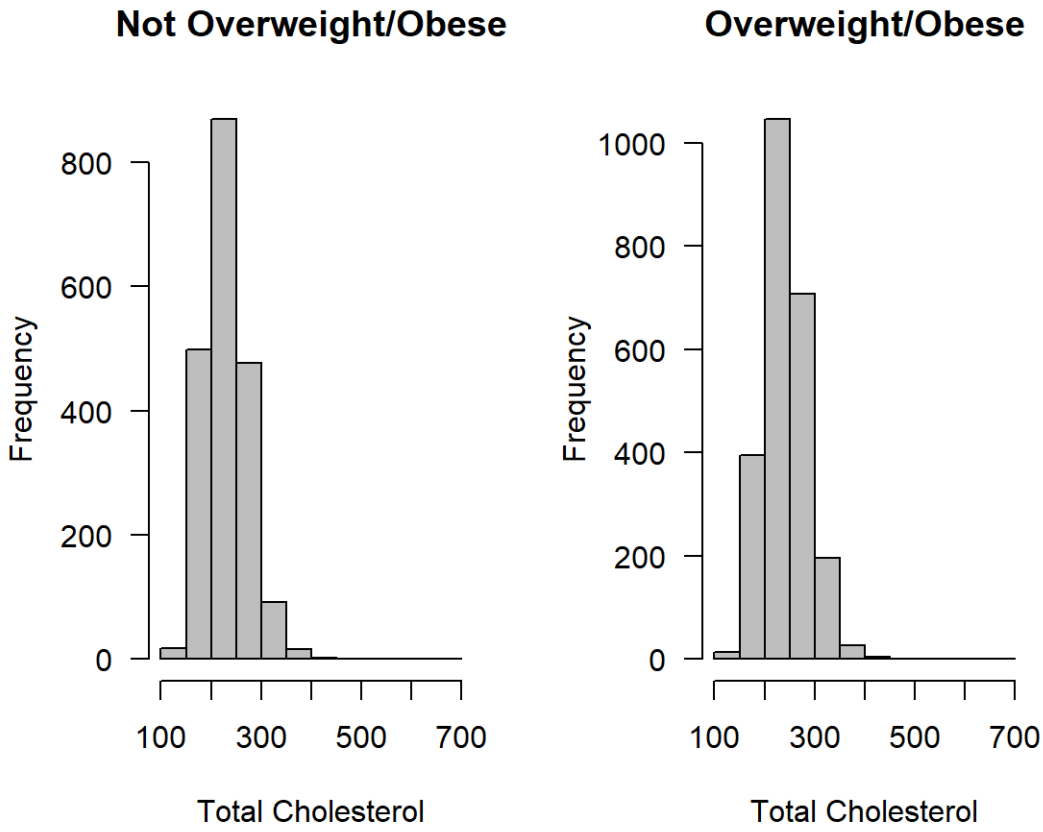
```
grp1 <- subset(fhs, OVERWEIGHTOBESE_factor == "No")  # new data frame containing subgroup 1
grp2 <- subset(fhs, OVERWEIGHTOBESE_factor == "Yes") # new data frame containing subgroup 2

range(grp1$TOTCHOL, grp2$TOTCHOL, na.rm=TRUE)  # Want x-axis range to be the same in both histograms
```

```
## [1] 113 696
```

We see that the smallest value of `TOTCHOL` in the two subgroups is 113 and the largest value is 696. Using this information, we will set the lower and upper limits of the histogram bin breaks in both histograms at 100 and 700, respectively.

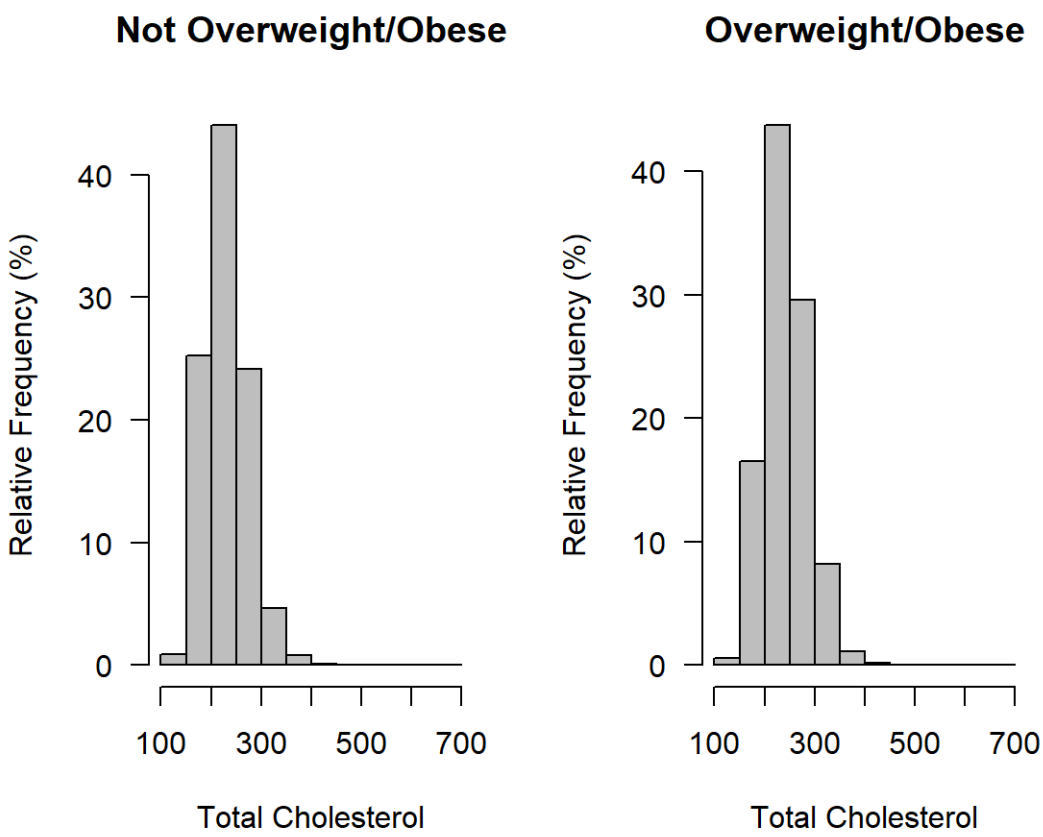
`par(mfrow=c(rows, cols))` allows us to put multiple plots in the same plot area. For example, to create two plots side-by-side, request a plot layout of 1 row and 2 columns using `par(mfrow=c(1,2))`. Reset the plotting area to the traditional 1x1 plot using `par(mfrow=c(1,1))`.

```
par(mfrow = c(1,2))   # figures will be plotted in 1 row, 2 columns

hist(grp1$TOTCHOL,
     breaks = seq(100, 700, by=50),
     main = "Not Overweight/Obese",
     xlab = "Total Cholesterol",
     ylab = "Frequency",
     col = "gray",
     las = 1)

hist(grp2$TOTCHOL,
     breaks = seq(100, 700, by=50),
     main = "Overweight/Obese",
     xlab = "Total Cholesterol",
     ylab = "Frequency",
     col = "gray",
     las = 1)
```



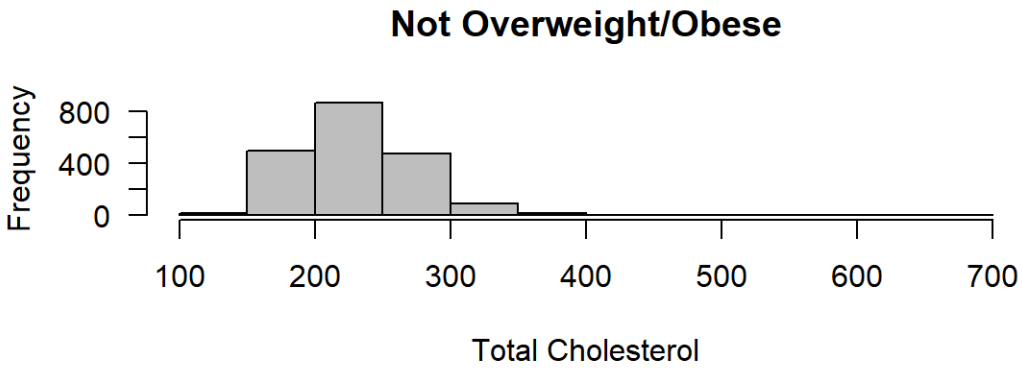
Just as when plotting one overall histogram, we can change the y-axis to display the relative frequency within each subgroup.

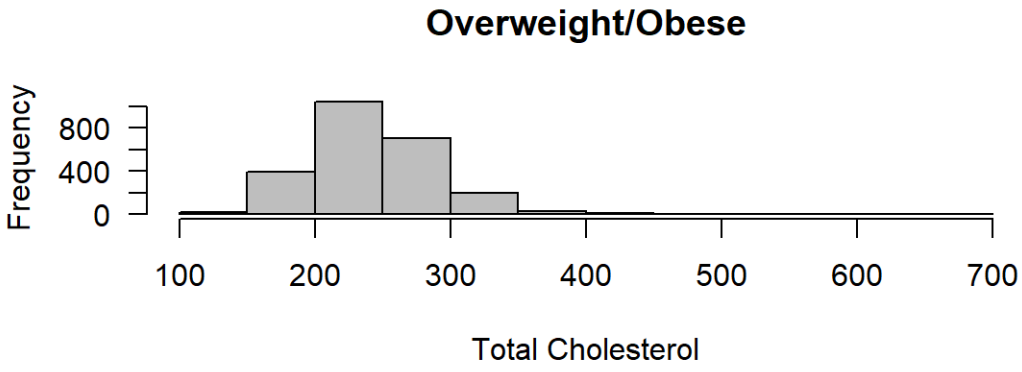
```
h1 <- hist(grp1$TOTCHOL,
           breaks = seq(100, 700, by=50),
           plot = FALSE)
h1$counts <- 100 * h1$counts / sum(h1$counts)   # relative frequency
h2 <- hist(grp2$TOTCHOL,
           breaks = seq(100, 700, by=50),
           plot = FALSE)
h2$counts <- 100 * h2$counts / sum(h2$counts)   # relative frequency

par(mfrow = c(1,2))   # figures will be plotted in 1 row, 2 columns

plot(h1,
     main = "Not Overweight/Obese",
     xlab = "Total Cholesterol",
     ylab = "Relative Frequency (%)",
     col = "gray",
     las = 1)

plot(h2,
     main = "Overweight/Obese",
     xlab = "Total Cholesterol",
     ylab = "Relative Frequency (%)",
     col = "gray",
     las = 1)
```



To request one plot on top of the other, request a plot layout of 2 rows and 1 column using `par(mfrow=c(2,1))` .

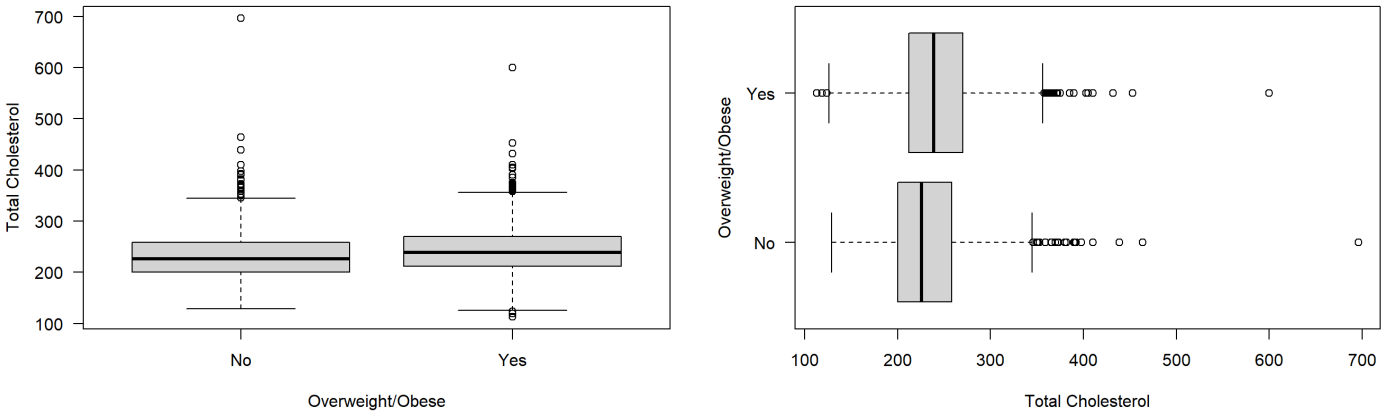
```
par(mfrow = c(2,1))   # figures will be plotted in 2 rows, 1 column

hist(grp1$TOTCHOL,
     breaks = seq(100, 700, by=50),
     main="Not Overweight/Obese",
     xlab = "Total Cholesterol",
     ylab="Frequency",
     col = "gray",
     las = 1)

hist(grp2$TOTCHOL,
     breaks = seq(100, 700, by=50),
     main="Overweight/Obese",
     xlab = "Total Cholesterol",
     ylab="Frequency",
     col = "gray",
     las = 1)
```





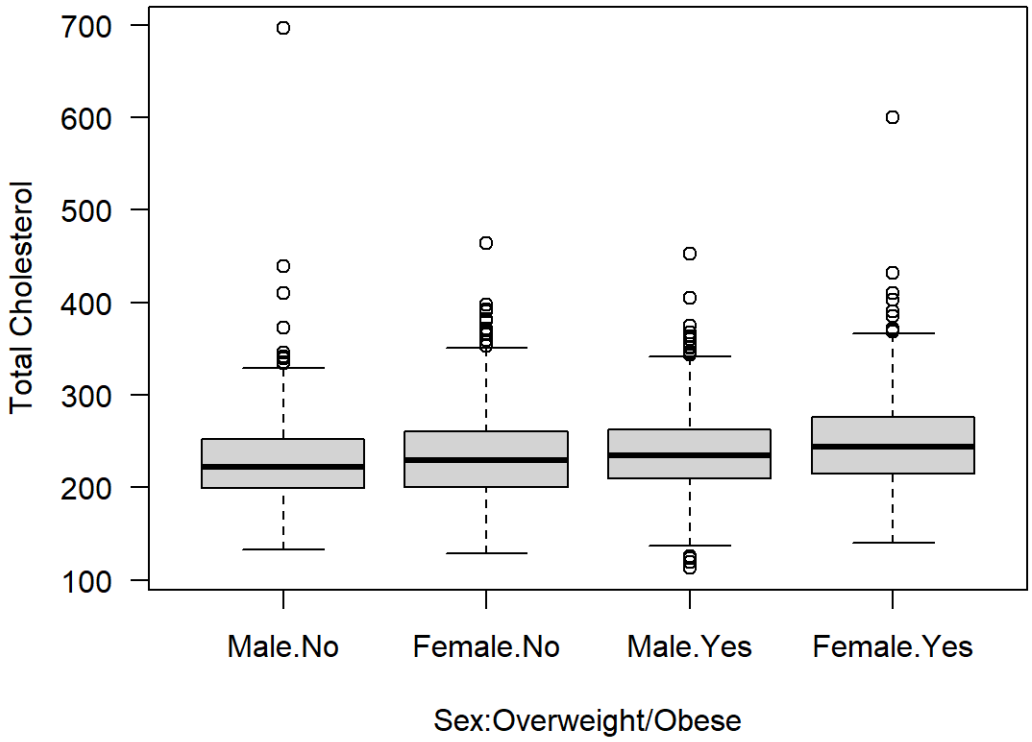
We can plot **boxplots** separately by group using a formula statement to specify the quantitative variable being plotted followed by ~ and the grouping variable (a factor).

```
boxplot(TOTCHOL ~ OVERWEIGHTOBESE_factor, data=fhs,
        xlab = "Overweight/Obese",
        ylab = "Total Cholesterol",
        las = 1)

boxplot(TOTCHOL ~ OVERWEIGHTOBESE_factor, data=fhs,
        xlab = "Total Cholesterol",
        ylab = "Overweight/Obese",
        horizontal = TRUE,
        las = 1)
```



We can look at combinations of two categorical variables crossing two grouping variables on the right hand side of the `~` in the formula statement. For example, if we are interested in presenting boxplots by sex within each overweight/obesity category use `SEX_factor*OVERWEIGHTOBESE_factor`:

```
boxplot(TOTCHOL ~ SEX_factor*OVERWEIGHTOBESE_factor, data=fhs,
        xlab = "Sex:Overweight/Obese",
        ylab = "Total Cholesterol",
        las = 1)
```



Categorical Variables

Categorical variables are typically summarized using frequency tables, cross-tabulations (a.k.a., contingency tables), and barplots.

Numerical Summaries

A **frequency distribution** is summarized in a **frequency table**. Frequencies (counts) and relative frequencies (percentages) are commonly reported together. The relationship between two categorical variables is displayed using a **cross-tabulation.**

Overall

The `table()` function returns the number of observations in each category. We use the `dnn` option to assign a name to the variable displayed in the printed table. The `useNA` option (i.e., `useNA = "ifany"`) will display the `NA` category in the table if missing values are present.

```
table(fhs$CIGPDAYGRP_factor, dnn = "Cigarettes Per Day")
```

```
## Cigarettes Per Day
##     0  1-19 20-39   40+
##  2253   907  1077   165
```

```
table(fhs$CIGPDAYGRP_factor, dnn = "Cigarettes Per Day", useNA = "ifany")  # displays NA category
```

```
## Cigarettes Per Day
##     0  1-19 20-39   40+  <NA>
##  2253   907  1077   165    32
```

The `sort()` function applied to a table displays the table categories in ascending order (categories with lowest count first). The `decreasing = TRUE` option in the `sort()` function sorts in descending order.

```
tab <- table(fhs$CIGPDAYGRP_factor, dnn = "Cigarettes Per Day")
sort(tab)
```

```
## Cigarettes Per Day
##   40+  1-19 20-39     0
##   165   907  1077  2253
```

```
sort(tab, decreasing = TRUE)
```

```
## Cigarettes Per Day
##     0 20-39  1-19   40+
##  2253  1077   907   165
```

The `prop.table()` function returns the proportion of observations in each category. The argument of the `prop.table()` function must be a table object. Here, `table(fhs$CIGPDAYGRP_factor)` is saved as the object `tab` and we apply `prop.table()` to `tab`.

```
tab <- table(fhs$CIGPDAYGRP_factor, dnn = "Cigarettes Per Day")
prop.table(tab)
```

```
## Cigarettes Per Day
##          0       1-19      20-39        40+
## 0.51181281 0.20604271 0.24466152 0.03748296
```

Again, the `useNA = "ifany"` option displays the NA category if missing values are present.

```
prop.table(table(fhs$CIGPDAYGRP_factor, dnn = "Cigarettes Per Day", useNA = "ifany"))  # can nest fu
nctions
```

```
## Cigarettes Per Day
##          0       1-19      20-39        40+       <NA>
## 0.50811908 0.20455571 0.24289581 0.03721245 0.00721696
```

The `cumsum()` function returns the cumulative sum over the categories of a variable. Here, we apply `cumsum()` to the `relfreq` object, which is the table of relative frequencies. The relative frequencies should sum to 1 over all categories.

```
tab <- table(fhs$CIGPDAYGRP_factor)
relfreq <- prop.table(tab)
cumsum(relfreq)
```

```
##         0      1-19     20-39       40+
## 0.5118128 0.7178555 0.9625170 1.0000000
```

Exercise: Combine `prop.table()` and `cumsum()` to find the proportion of non-missing values of `CIGPDAYGRP_factor`.

▶ Answer:

By Group

The `table()` function can be used to create a cross-tabulation by specifying two factor variable arguments. The first variable is the row variable and the second variable is the column variable. Below, we are creating a cross-tabulation of `OVERWEIGHTOBESE_factor` by `SEX_factor`. Specify labels to both table dimensions using a vector in the `dnn` option.

```
table(fhs$SEX_factor, fhs$OVERWEIGHTOBESE_factor, dnn = c("Sex", "Overweight/Obese"))
```

```
##         Overweight/Obese
## Sex        No  Yes
##   Male    715 1224
##   Female 1278 1198
```

The `rowSums()` and `colSums()` functions report the row and column totals, respectively.

```
tab <- table(fhs$SEX_factor, fhs$OVERWEIGHTOBESE_factor, dnn = c("Sex", "Overweight/Obese"))
rowSums(tab)
```

```
##   Male Female
##   1939   2476
```

```
colSums(tab)
```

```
##   No  Yes
## 1993 2422
```

The `prop.table()` function gives **cell proportions**. The cell proportions in a table sum to 1.

```
tab <- table(fhs$SEX_factor, fhs$OVERWEIGHTOBESE_factor, dnn = c("Sex", "Overweight/Obese"))
prop.table(tab)  # cell proportions
```

```
##         Overweight/Obese
## Sex             No       Yes
##   Male   0.1619479 0.2772367
##   Female 0.2894677 0.2713477
```

Often, we would like to look at the proportions within a fixed row or column. Fixing the row and reporting the percentages in each column level gives **row proportions**. For example, out of all male patients, 63.13% are overweight/obese. Out of all female patients, 48.38% are overweight/obese. Row proportions are requested with the `margin = 1` option. Row proportions should sum to 1 in each row.

Likewise, fixing the column and reporting the percentages in each row level gives **column proportions**. In those who are overweight/obese, 50.54% are male and 49.46% are female. Column proportions are requested with the `margin = 2` option. Column proportions should sum to 1 in each column.

```
tab <- table(fhs$SEX_factor, fhs$OVERWEIGHTOBESE_factor, dnn = c("Sex", "Cigarettes Per Day"))

prop.table(tab, margin = 1)  # row proportions
```

```
##         Cigarettes Per Day
## Sex             No       Yes
##   Male   0.3687468 0.6312532
##   Female 0.5161551 0.4838449
```

```
rowSums(prop.table(tab, margin = 1))
```

```
##   Male Female
##      1      1
```

```
prop.table(tab, margin = 2)  # column proportions
```

```
##         Cigarettes Per Day
## Sex             No       Yes
##   Male   0.3587556 0.5053675
##   Female 0.6412444 0.4946325
```

```
colSums(prop.table(tab, margin = 2))
```

```
##  No Yes
##   1   1
```

Graphical Summaries

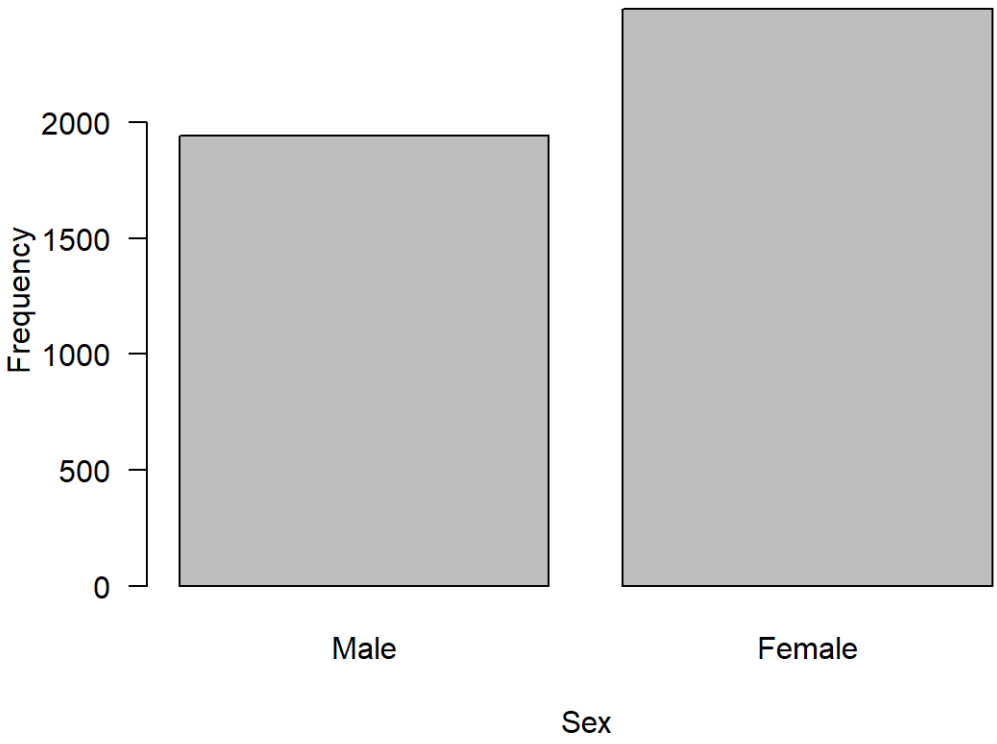
Barplots are used as a visual representation of a frequency distribution.

Overall

Barplots are created using the `barplot()` function. The function is applied to a table object. By default, a vertical barplot is produced. The option `horiz = TRUE` creates a horizontal barplot.

If the table plotted is a frequency table, then counts are plotted on the y-axis.

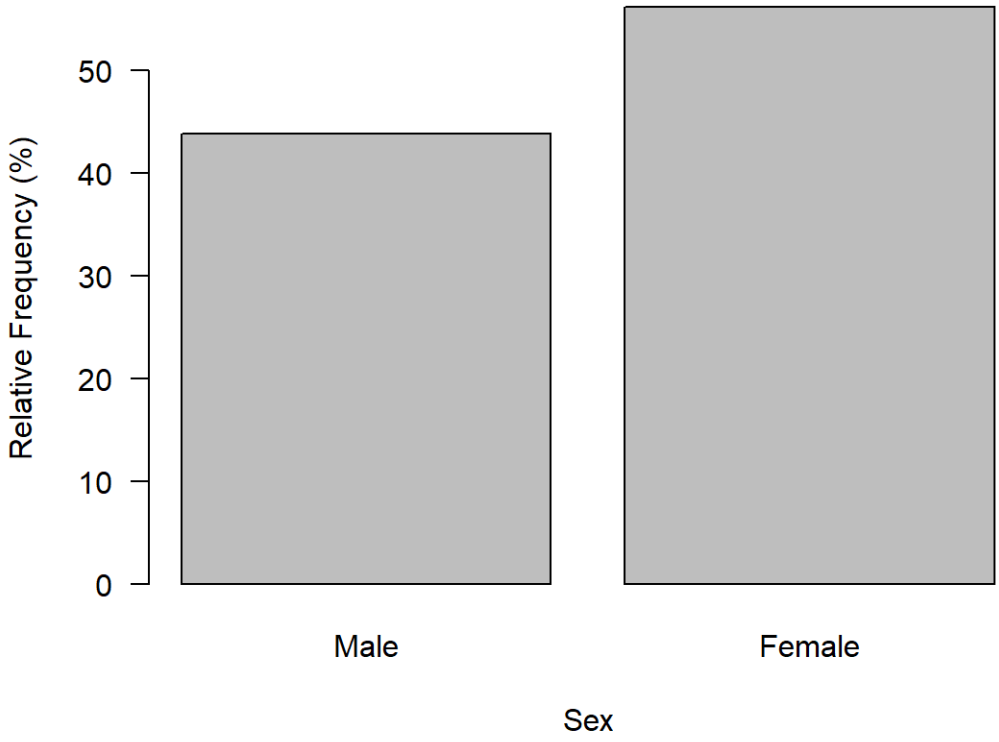
```
tab <- table(fhs$SEX_factor)
barplot(tab,
        main = "",    # title
        xlab = "Sex",
        ylab = "Frequency",
        las = 1)
```



Distribution of FHS Participants by Sex

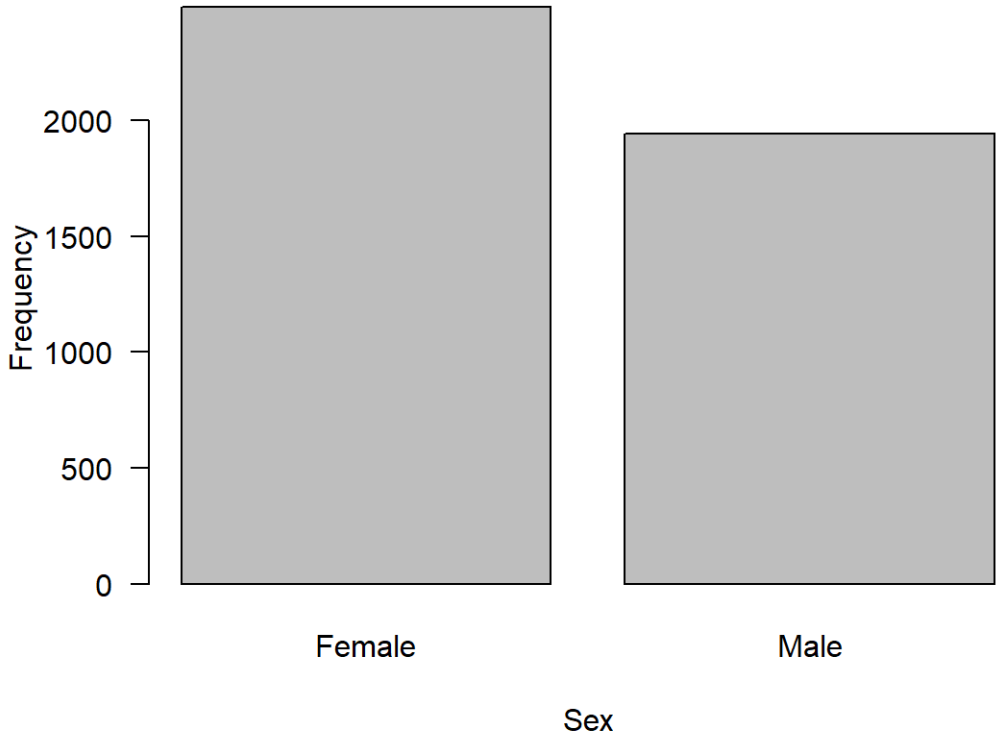
If the table reports proportions, then proportions or relative frequencies are plotted on the y-axis.

```
tab <- table(fhs$SEX_factor)
relfreq <- 100*prop.table(tab)
barplot(relfreq,
        main = "",
        xlab = "Sex",
        ylab = "Relative Frequency (%)",
        las = 1)
```



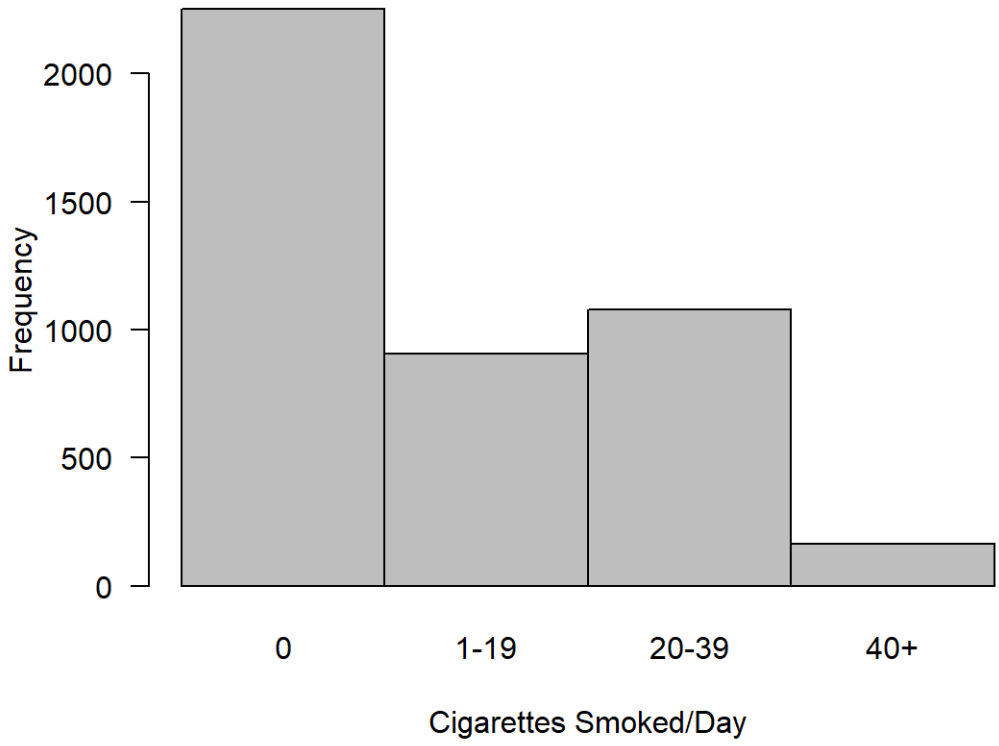
Categories can be plotted in descending order by applying the `sort()` function to the table.

```
tab <- sort(table(fhs$SEX_factor), decreasing = TRUE)
barplot(tab,
        main = "",
        xlab = "Sex",
        ylab = "Frequency",
        las = 1)
```



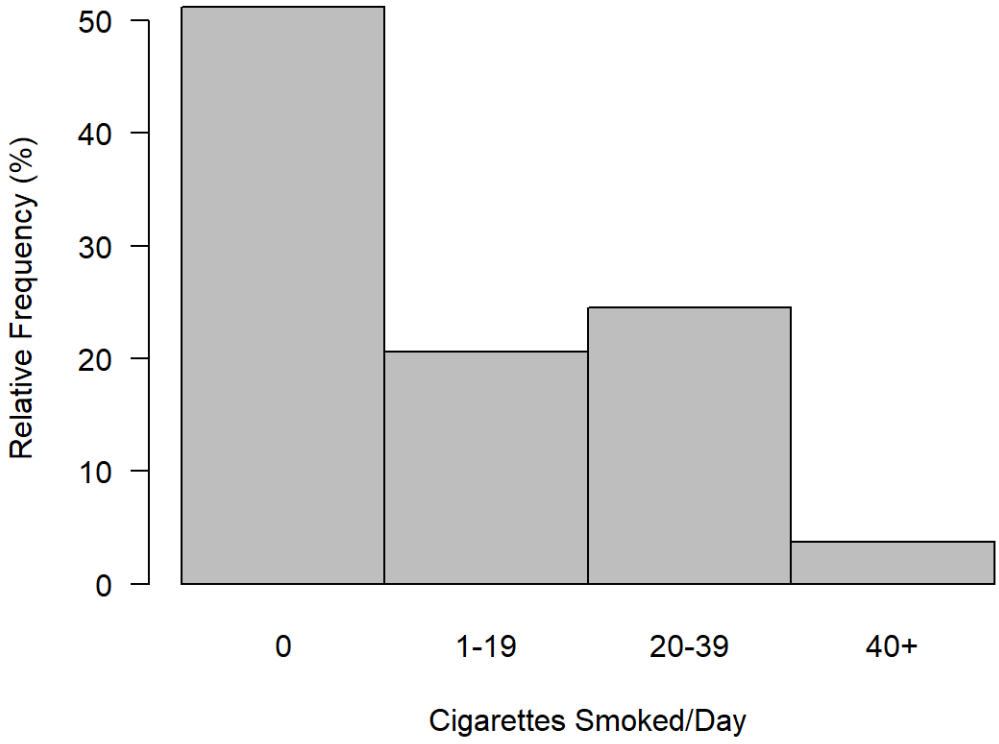
We can use the `barplot()` function to produce what appears to be a histogram for use with ordinal variables by removing the spacing between bars with the `space = 0` option.

```
tab <- table(fhs$CIGPDAYGRP_factor)
barplot(tab, space = 0,
        main = "",
        xlab = "Cigarettes Smoked/Day",
        ylab = "Frequency",
        las = 1)
```



Again, if the table consists of relative frequencies, the y-axis will display relative frequencies:

```
tab <- table(fhs$CIGPDAYGRP_factor)
relfreq <- 100*prop.table(tab)
barplot(relfreq, space = 0,
        main = "",
        xlab = "Cigarettes Smoked/Day",
        ylab = "Relative Frequency (%)",
        las = 1)
```



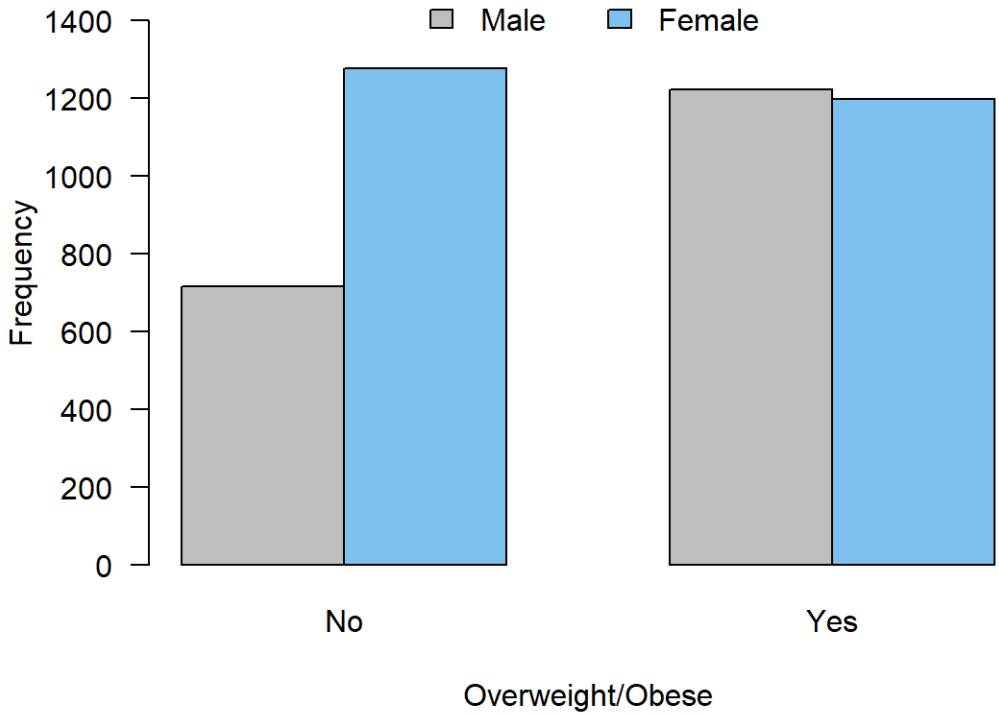
By Group

We can create **grouped barplots** using contingency tables. The column variable (second factor variable listed) will be placed on the x-axis and the row variable will be subgroups within each level of the column variable.

```
tab <- table(fhs$SEX_factor, fhs$OVERWEIGHTOBESE_factor)
tab
```

```
##
##             No  Yes
##   Male    715 1224
##   Female 1278 1198
```

```
barplot(tab,
        main = "",
        xlab = "Overweight/Obese",
        ylab = "Frequency",
        ylim = c(0, 1500),  # increasing range of y-axis to accommodate legend
        beside = TRUE,
        col = c("gray", "skyblue2"),
        las = 1)

legend(x = "top",
       legend = rownames(tab),  # legend text
       fill = c("gray", "skyblue2"),
       bty = "n", # no box surrounding legend
       ncol = 2)  # number of columns in legend
```



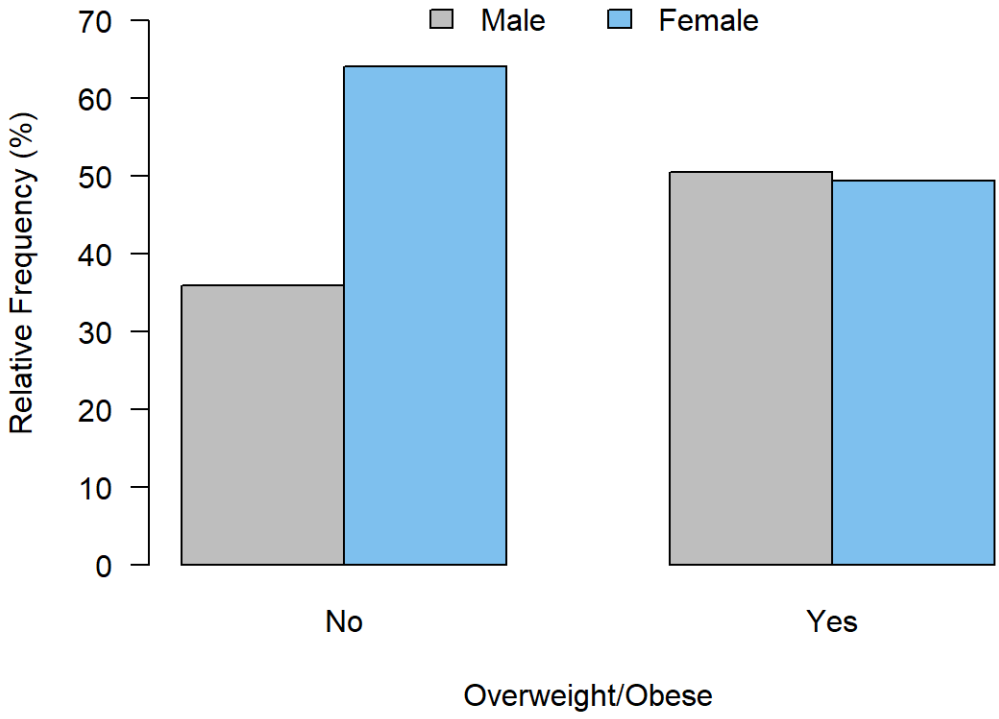
Distribution of Sex and Overweight/Obese Status

In this case, it makes sense to report column percentages (i.e., for a given level of Overweight/Obese, what is the proportion of Males and the proportion of females).

```
tab <- table(fhs$SEX_factor, fhs$OVERWEIGHTOBESE_factor)
relfreq <- 100*prop.table(tab, margin = 2)
relfreq
```

```
##
##                No      Yes
##   Male   35.87556 50.53675
##   Female 64.12444 49.46325
```

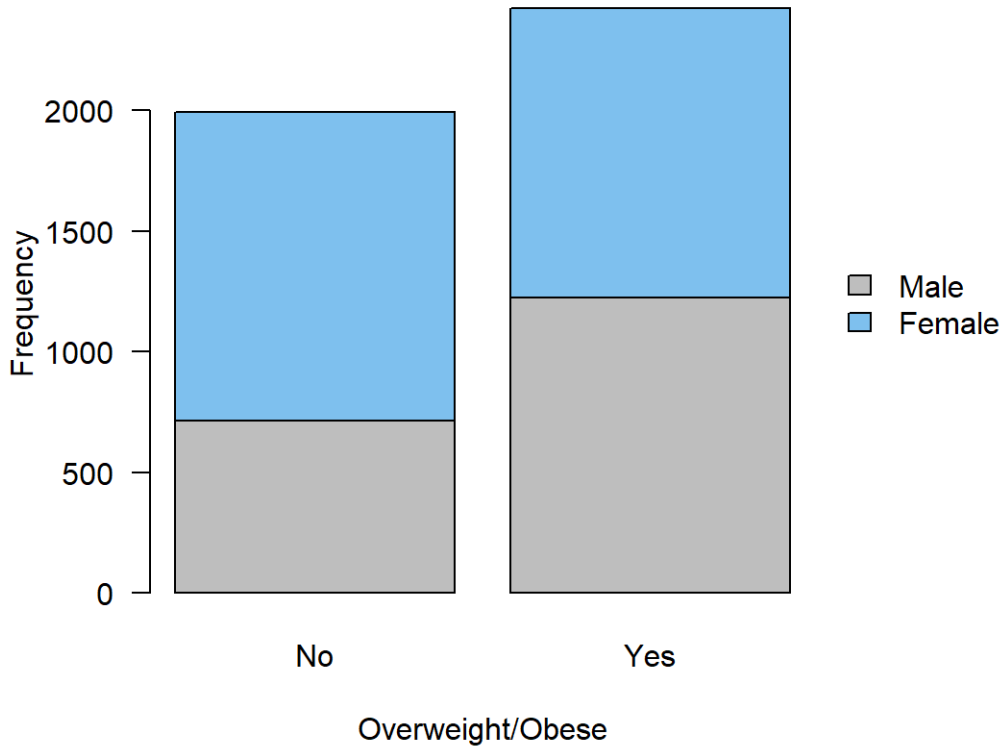
```
barplot(relfreq,
        main = "",
        xlab = "Overweight/Obese",
        ylab = "Relative Frequency (%)",
        ylim = c(0, 75),
        beside = TRUE,
        col = c("gray", "skyblue2"),
        las = 1)

legend(x = "top",
       legend = rownames(tab),
       fill = c("gray", "skyblue2"),
       bty = "n",
       ncol = 2)
```



Distribution of Sex and Overweight/Obese Status

We can similarly create **stacked barplots** by removing the `beside = TRUE` option:

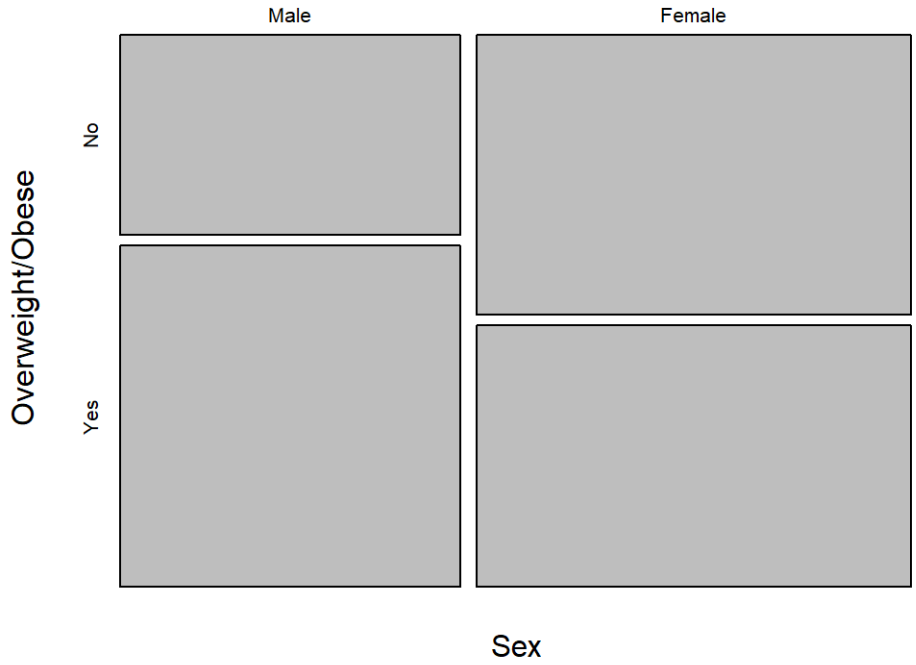
```
tab <- table(fhs$SEX_factor, fhs$OVERWEIGHTOBESE_factor)

par(mar = c(5,4,4,8))  # changing the plot margins (increasing right margin to accommodate legend in margin)

barplot(tab,
        main = "",
        xlab = "Overweight/Obese",
        ylab = "Frequency",
        col = c("gray", "skyblue2"),
        las = 1)

legend(x = "right",
       legend = rownames(tab),
       fill = c("gray", "skyblue2"),
       bty = "n",
       xpd = TRUE,  # placement of legend in margin
       inset=c(-0.3, 0))  # adjustment to inset may be necessary for proper legend placement
```



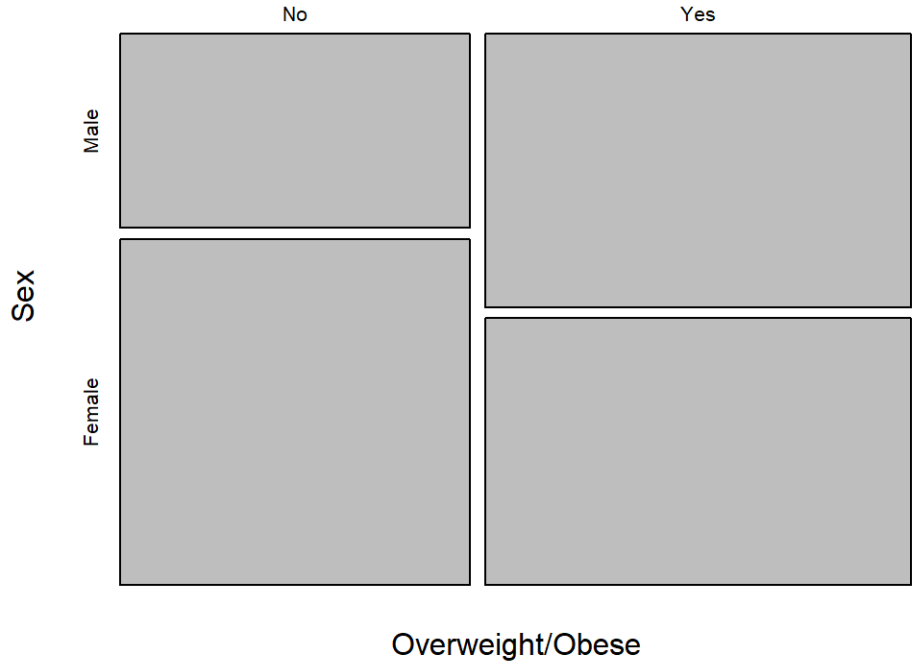
Distribution of Sex and Overweight/Obese Status

Mosaic plots are useful for visualizing the data from the contingency table. The first variable (`SEX_factor`) is plotted along the x-axis and the second variable (`OVERWEIGHTOBESE_factor`) is plotted along the y-axis.

```
tab <- table(fhs$SEX_factor, fhs$OVERWEIGHTOBESE_factor)
mosaicplot(tab,
           main = "",
           xlab = "Sex",
           ylab = "Overweight/Obese")
```




```
mosaicplot(table(fhs$OVERWEIGHTOBESE_factor, fhs$SEX_factor),  # nest table within mosaicplot() func
tion
           main = "",
           xlab = "Overweight/Obese",
           ylab = "Sex")
```



(Bonus) If there's time during Lab: `tableby()`

Quantitative Variables

The `tableby()` [1] function in the `arsenal` package can be used to create customized tables that can accommodate both quantitative and categorical variables. To create a simple table stratified by levels of a grouping variable (e.g., overweight/obese status), use a formula statement to specify the grouping variable (usually a factor) followed by `~` and the variables that you want summarized, each separated by `+`. For example, `tableby(grouping_variable ~ var1 + var2 + var3 + ...)`. *Numeric variables* are summarized using default summary statistics such as mean, standard deviation and range; *character and factor variables* are summarized using default summary statistics such as count and percentage. Be sure to use the `results = "asis"` chunk option to display the formatted table in your output document.

First, we create the `tableby()` object (named `tab1`, below), then we run `summary(tab1)` to produce the table.

```
# Default tableby table summarizing some quantitative variables by group
tab1 <- tableby(OVERWEIGHTOBESE_factor ~ AGE + BMI + TOTCHOL + SYSBP+ DIABP + HEARTRTE + GLUCOSE + C
IGPDAY,
                test = FALSE, data = fhs)
summary(tab1, term.name = TRUE)  # term.name displays the factor label in the header
```

OVERWEIGHTOBESE_factor	No (N=1993)	Yes (N=2422)	Total (N=4415)
AGE			
Mean (SD)	48.632 (8.613)	50.977 (8.565)	49.918 (8.665)

OVERWEIGHTOBESE_factor	No (N=1993)	Yes (N=2422)	Total (N=4415)
Range	33.000 - 69.000	32.000 - 70.000	32.000 - 70.000
BMI			
Mean (SD)	22.485 (1.821)	28.612 (3.315)	25.846 (4.102)
Range	15.540 - 24.990	25.010 - 56.800	15.540 - 56.800
TOTCHOL			
N-Miss	19	32	51
Mean (SD)	230.795 (44.287)	242.192 (44.309)	237.037 (44.656)
Range	129.000 - 696.000	113.000 - 600.000	113.000 - 696.000
SYSBP			
Mean (SD)	126.591 (20.355)	138.079 (22.606)	132.893 (22.360)
Range	83.500 - 244.000	83.500 - 295.000	83.500 - 295.000
DIABP			
Mean (SD)	79.142 (10.871)	86.347 (12.004)	83.094 (12.051)
Range	50.000 - 136.000	48.000 - 142.500	48.000 - 142.500
HEARTRTE			
N-Miss	0	1	1
Mean (SD)	75.168 (12.063)	76.485 (12.133)	75.890 (12.118)
Range	45.000 - 143.000	44.000 - 140.000	44.000 - 143.000
GLUCOSE			
N-Miss	184	208	392
Mean (SD)	80.423 (21.779)	83.573 (25.926)	82.157 (24.197)
Range	40.000 - 394.000	40.000 - 394.000	40.000 - 394.000
CIGPDAY			
N-Miss	14	18	32
Mean (SD)	10.023 (11.474)	8.122 (12.247)	8.980 (11.940)
Range	0.000 - 60.000	0.000 - 70.000	0.000 - 70.000

Note: To create a table for the overall sample (i.e., not stratified by a grouping variable), omit the `grouping_variable` from the `tableby()` function (i.e., `tableby( ~ var1 + var2 + var3 + ...)`).

By default, reported statistics for quantitative variables (`numeric.stats`) are `Nmiss` , `meansd` , and `range` , but the `tableby()` table can be customized. Request different statistics in the `numeric.stats` option in the `tableby.control()` function:

```
numeric.stats = c(...)    stats.labels = list(...)
```

`numeric.stats = c(...)`	`stats.labels = list(...)`
`N`	N = “Sample Size”
`Nmiss`	Nmiss = “Missing” (only shows “Missing” row when missing values are present)
`Nmiss2`	Nmiss2 = “Missing” (always shows “Missing” row)
`meansd`	meansd = “Mean (SD)”
`median`	median = “Median”
`medianrange`	medianrange = “Median (Range)”
`medianq1q3`	medianq1q3 = “Median (Q1, Q3)”
`q1q3`	q1q3 = “Q1, Q3”
`iqr`	iqr = “IQR”
`mean`	mean = “Mean”
`sd`	sd = “SD”
`var`	var = “Var”
`max`	max = “Max”
`min`	min = “Min”
`meanCI`	meanCI = “Mean (CI)”
`sum`	sum = “Sum”
`range`	range = “Range”
`Npct`	Npct = “N (Pct)”; displays number (%) of non-missing observations in each category
`countpct`	countpct = “N (%)” (column percent); displays each value of the quantitative variable

It is also possible to apply variable labels to make the output clearer to the reader (e.g., instead of displaying `TOTCHOL`, change the variable label of `TOTCHOL` to display as `Total Cholesterol` in the table). This is done using the object `my_labels` created below.

```
# Requesting specific statistics, specifying stat labels
my_controls <- tableby.control(
  test = F,             # do not display p-value comparing columns
  total = T,            # display Total column
  numeric.stats = c("Nmiss2", "meansd", "medianq1q3", "medianrange", "range"),    # requested stats
  cat.stats = c("Nmiss", "countpct"),
  stats.labels = list(
    Nmiss2 = "Missing",
    meansd = "Mean (SD)",
    medianq1q3 = "Median (Q1, Q3)",
    medianrange = "Median (Range)",
    range = "Min - Max",
    Nmiss = "Missing",
    countpct = "N (%)"
  ),
  digits = 1
)

# Labeling variables
my_labels <- list(
  AGE = "Age (years)",
  BMI = "Body Mass Index",
  TOTCHOL = "Total Cholesterol",
  SYSBP = "Systolic Blood Pressure",
  DIABP = "Diastolic Blood Pressure",
  OVERWEIGHTOBESE_factor = "Overweight or Obese"
)

# tableby(grouping_variable ~ variables to be summarized, separated by +)
table_one <- tableby(OVERWEIGHTOBESE_factor ~ AGE + BMI + TOTCHOL + SYSBP + DIABP,
  data = fhs,
  control = my_controls
)

# Produces table, adds title
summary(table_one,
  labelTranslations = my_labels,
  title = "Summary Statistics of FHS Data: Quantitative Variables",
  term.name = TRUE
)
```

Summary Statistics of FHS Data: Quantitative Variables

Overweight or Obese	**No (N=1993)**	**Yes (N=2422)**	**Total (N=4415)**
Age (years)			
Missing	0	0	0
Mean (SD)	48.6 (8.6)	51.0 (8.6)	49.9 (8.7)
Median (Q1, Q3)	47.0 (41.0, 55.0)	51.0 (44.0, 58.0)	49.0 (42.5, 57.0)
Median (Range)	47.0 (33.0, 69.0)	51.0 (32.0, 70.0)	49.0 (32.0, 70.0)
Min - Max	33.0 - 69.0	32.0 - 70.0	32.0 - 70.0

Overweight or Obese	No (N=1993)	Yes (N=2422)	Total (N=4415)
Body Mass Index			
Missing	0	0	0
Mean (SD)	22.5 (1.8)	28.6 (3.3)	25.8 (4.1)
Median (Q1, Q3)	22.9 (21.3, 24.0)	27.8 (26.3, 29.8)	25.4 (23.1, 28.1)
Median (Range)	22.9 (15.5, 25.0)	27.8 (25.0, 56.8)	25.4 (15.5, 56.8)
Min - Max	15.5 - 25.0	25.0 - 56.8	15.5 - 56.8
Total Cholesterol			
Missing	19	32	51
Mean (SD)	230.8 (44.3)	242.2 (44.3)	237.0 (44.7)
Median (Q1, Q3)	226.0 (200.0, 258.0)	239.0 (212.0, 269.8)	234.0 (206.0, 264.0)
Median (Range)	226.0 (129.0, 696.0)	239.0 (113.0, 600.0)	234.0 (113.0, 696.0)
Min - Max	129.0 - 696.0	113.0 - 600.0	113.0 - 696.0
Systolic Blood Pressure			
Missing	0	0	0
Mean (SD)	126.6 (20.4)	138.1 (22.6)	132.9 (22.4)
Median (Q1, Q3)	123.0 (112.5, 135.0)	134.0 (122.0, 150.0)	129.0 (117.5, 144.0)
Median (Range)	123.0 (83.5, 244.0)	134.0 (83.5, 295.0)	129.0 (83.5, 295.0)
Min - Max	83.5 - 244.0	83.5 - 295.0	83.5 - 295.0
Diastolic Blood Pressure			
Missing	0	0	0
Mean (SD)	79.1 (10.9)	86.3 (12.0)	83.1 (12.1)
Median (Q1, Q3)	78.0 (72.0, 85.0)	85.0 (79.0, 93.0)	82.0 (75.0, 90.0)
Median (Range)	78.0 (50.0, 136.0)	85.0 (48.0, 142.5)	82.0 (48.0, 142.5)
Min - Max	50.0 - 136.0	48.0 - 142.5	48.0 - 142.5

Note that one would not normally display all three of the following: Median (Q1, Q3), Median (Range), and Min - Max.

Categorical Variables

The `tableby()` function in the `arsenal` package can be used to create tables that can accommodate both quantitative and categorical variables. As we did above when considering only quantitative variables, use a formula statement to specify the grouping variable followed by `~` and the character or factor variables that we want summarized, each separated by `+`. For example, `grouping_variable ~ var1_factor + var2_factor + var3_factor`. Again, be sure to use the `results = "asis"` chunk option to display the formatted table in your output document.

```
# Default tableby table
tab1 <- tableby(OVERWEIGHTOBESE_factor ~ SEX_factor + CIGPDAYGRP_factor,
                test = FALSE, data = fhs)
summary(tab1, term.name = TRUE)  # term.name displays the factor label in the header
```

OVERWEIGHTOBESE_factor	No (N=1993)	Yes (N=2422)	Total (N=4415)
SEX_factor			
Male	715 (35.9%)	1224 (50.5%)	1939 (43.9%)
Female	1278 (64.1%)	1198 (49.5%)	2476 (56.1%)
CIGPDAYGRP_factor			
N-Miss	14	18	32
0	842 (42.5%)	1399 (58.2%)	2241 (51.1%)
1-19	513 (25.9%)	392 (16.3%)	905 (20.6%)
20-39	564 (28.5%)	508 (21.1%)	1072 (24.5%)
40+	60 (3.0%)	105 (4.4%)	165 (3.8%)

The `includeNA()` function can be applied to categorical variables to include NAs in the counts and percents.

```
# Default tableby table with NA category n and % displayed
tab1 <- tableby(OVERWEIGHTOBESE_factor ~ includeNA(SEX_factor) + includeNA(CIGPDAYGRP_factor),
                test = FALSE, data = fhs)
summary(tab1, term.name = TRUE)
```

OVERWEIGHTOBESE_factor	No (N=1993)	Yes (N=2422)	Total (N=4415)
includeNA(SEX_factor)			
Male	715 (35.9%)	1224 (50.5%)	1939 (43.9%)
Female	1278 (64.1%)	1198 (49.5%)	2476 (56.1%)
(Missing)	0 (0.0%)	0 (0.0%)	0 (0.0%)
includeNA(CIGPDAYGRP_factor)			
0	842 (42.2%)	1399 (57.8%)	2241 (50.8%)
1-19	513 (25.7%)	392 (16.2%)	905 (20.5%)
20-39	564 (28.3%)	508 (21.0%)	1072 (24.3%)
40+	60 (3.0%)	105 (4.3%)	165 (3.7%)
(Missing)	14 (0.7%)	18 (0.7%)	32 (0.7%)

By default, reported statistics for categorical variables (`cat.stats`) are `Nmiss` , and `countpct` , but the `tableby()` table can be customized. Request different statistics in the `cat.stats` option in the `tableby.control()` function:

```
cat.stats = c(...)        stats.labels = list(...)
```

cat.stats = c(...)	stats.labels = list(...)
Nmiss	Nmiss = “Missing” (only shows “Missing” row when missing values are present)
Nmiss2	Nmiss2 = “Missing” (always shows “Missing” row)
count	count = “N”
countN	countN = “N/Total”
countpct	countpct = “N (%)” (column percent)
countrowpct	countrowpct = “N (%)” (row percent)
countcellpct	countcellpct = “N (%)” (cell percent)

```
# Requesting specific statistics, specifying stat labels
my_controls <- tableby.control(
  test = F,
  total = T,
  numeric.stats = c("Nmiss2", "meansd", "medianq1q3", "medianrange", "range"),
  cat.stats = c("Nmiss", "countpct"),     # requested stats
  stats.labels = list(
    Nmiss2 = "Missing",
    meansd = "Mean (SD)",
    medianq1q3 = "Median (Q1, Q3)",
    medianrange = "Median (Range)",
    range = "Min - Max",
    Nmiss = "Missing",
    countpct = "N (%)"
    ),
  digits = 1
)

# Labeling variables
my_labels <- list(
  SEX_factor = "Sex",
  CIGPDAYGRP_factor = "Cigarettes per Day",
  OVERWEIGHTOBESE_factor = "Overweight or Obese"
)

# tableby(grouping_variable ~ variables to be summarized, separated by +)
table_one <- tableby(OVERWEIGHTOBESE_factor ~ SEX_factor + CIGPDAYGRP_factor,
  data = fhs,
  control = my_controls
)

# Produces table, adds title
summary(table_one,
  labelTranslations = my_labels,
  title = "Summary Statistics of FHS Data: Categorical Variables",
  term.name = TRUE
)
```

Summary Statistics of FHS Data: Categorical Variables

Overweight or Obese	No (N=1993)	Yes (N=2422)	Total (N=4415)
Sex			
Male	715 (35.9%)	1224 (50.5%)	1939 (43.9%)
Female	1278 (64.1%)	1198 (49.5%)	2476 (56.1%)
Cigarettes per Day			
Missing	14	18	32
0	842 (42.5%)	1399 (58.2%)	2241 (51.1%)
1-19	513 (25.9%)	392 (16.3%)	905 (20.6%)
20-39	564 (28.5%)	508 (21.1%)	1072 (24.5%)
40+	60 (3.0%)	105 (4.4%)	165 (3.8%)

All Variables

The nice thing about the `tableby()` function is that we can include both quantitative and categorical variables in the same table. *Numeric variables* are summarized by the `numeric.stats` and *character and factor variables* are summarized by the `cat.stats` requested in `tableby.control()`.

```
# Requesting specific statistics, specifying stat labels
my_controls <- tableby.control(
  test = F,
  total = T,
  numeric.stats = c("Nmiss2", "meansd", "medianrange"),  # requested stats
  cat.stats = c("Nmiss", "countpct"),                    # requested stats
  stats.labels = list(
    Nmiss2 = "Missing",
    meansd = "Mean (SD)",
    medianrange = "Median (Range)",
    Nmiss = "Missing",
    countpct = "N (%)"
  ),
  digits = 1
)

# Labeling variables
my_labels <- list(
  AGE = "Age (years)",
  BMI = "Body Mass Index",
  TOTCHOL = "Total Cholesterol",
  SYSBP = "Systolic Blood Pressure",
  DIABP = "Diastolic Blood Pressure",
  SEX_factor = "Sex",
  CIGPDAYGRP_factor = "Cigarettes per Day",
  OVERWEIGHTOBESE_factor = "Overweight or Obese"
)

# tableby(grouping_variable ~ variables to be summarized, separated by +)
table_one <- tableby(OVERWEIGHTOBESE_factor ~ AGE + SEX_factor + BMI + CIGPDAYGRP_factor + TOTCHOL +
SYSBP + DIABP,
  data = fhs,
  control = my_controls
)

# Produces table, adds title
summary(table_one,
  labelTranslations = my_labels,
  title = "Summary Statistics of FHS Data: All Variables",
  term.name = TRUE
)
```

Summary Statistics of FHS Data: All Variables

Overweight or Obese	No (N=1993)	Yes (N=2422)	Total (N=4415)
Age (years)			
Missing	0	0	0
Mean (SD)	48.6 (8.6)	51.0 (8.6)	49.9 (8.7)
Median (Range)	47.0 (33.0, 69.0)	51.0 (32.0, 70.0)	49.0 (32.0, 70.0)
Sex			

Overweight or Obese	No (N=1993)	Yes (N=2422)	Total (N=4415)
Male	715 (35.9%)	1224 (50.5%)	1939 (43.9%)
Female	1278 (64.1%)	1198 (49.5%)	2476 (56.1%)
Body Mass Index			
Missing	0	0	0
Mean (SD)	22.5 (1.8)	28.6 (3.3)	25.8 (4.1)
Median (Range)	22.9 (15.5, 25.0)	27.8 (25.0, 56.8)	25.4 (15.5, 56.8)
Cigarettes per Day			
Missing	14	18	32
0	842 (42.5%)	1399 (58.2%)	2241 (51.1%)
1-19	513 (25.9%)	392 (16.3%)	905 (20.6%)
20-39	564 (28.5%)	508 (21.1%)	1072 (24.5%)
40+	60 (3.0%)	105 (4.4%)	165 (3.8%)
Total Cholesterol			
Missing	19	32	51
Mean (SD)	230.8 (44.3)	242.2 (44.3)	237.0 (44.7)
Median (Range)	226.0 (129.0, 696.0)	239.0 (113.0, 600.0)	234.0 (113.0, 696.0)
Systolic Blood Pressure			
Missing	0	0	0
Mean (SD)	126.6 (20.4)	138.1 (22.6)	132.9 (22.4)
Median (Range)	123.0 (83.5, 244.0)	134.0 (83.5, 295.0)	129.0 (83.5, 295.0)
Diastolic Blood Pressure			
Missing	0	0	0
Mean (SD)	79.1 (10.9)	86.3 (12.0)	83.1 (12.1)
Median (Range)	78.0 (50.0, 136.0)	85.0 (48.0, 142.5)	82.0 (48.0, 142.5)

1. https://www.rdocumentation.org/packages/arsenal/versions/3.5.0/topics/tableby (https://www.rdocumentation.org/packages/arsenal/versions/3.5.0/topics/tableby), https://cran.r-project.org/web/packages/arsenal/vignettes/tableby.html (https://cran.r-project.org/web/packages/arsenal/vignettes/tableby.html)↩