Summarizing data with R (with Lucy King)

This chapter will introduce you to how to summarize data using R, as well as providing an introduction to a popular set of R tools known as the "Tidyverse."

Before doing anything else we need to load the libraries that we will use in this notebook.

```
library(tidyverse)
```

```
## -- Attaching packages --------------------

## v ggplot2 3.2.1       v purrr   0.2.5
## v tibble  2.0.0       v dplyr   0.8.0.1
## v tidyr   0.8.2       v stringr 1.3.1
## v readr   1.3.1       v forcats 0.3.0

## Warning: package 'ggplot2' was built under R version 3.5.2

## Warning: package 'tibble' was built under R version 3.5.2

## Warning: package 'dplyr' was built under R version 3.5.2

## -- Conflicts ---- tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

```
library(cowplot)
```

```
## Warning: package 'cowplot' was built under R version 3.5.2

##
## Attaching package: 'cowplot'

## The following object is masked from 'package:ggplot2':
##
##     ggsave
```

```
library(knitr)
```

```
## Warning: package 'knitr' was built under R version 3.5.2
```

```
opts_chunk$set(tidy.opts=list(width.cutoff=60), tidy=TRUE)
```

We will use the NHANES dataset for several of our examples, so let's load the library that contains the data.

```
# load the NHANES data library

# first unload it if it's already loaded, to make sure

# we have a clean version

if("NHANES" %in% (.packages())){
  detach('package:NHANES', unload=TRUE)
}
library(NHANES)
```

Introduction to the Tidyverse

In this chapter we will introduce a way of working with data in R that is often referred to as the "Tidyverse." This comprises a set of packages that provide various tools for working with data, as well as a few special ways of using those functions

Making a data frame using tibble()

The tidyverse provides its over version of a data frame, which known as a *tibble*. A tibble is a data frame but with some smart tweaks that make it easier to work with, expecially when using functions from the tidyverse. See here for more information on the function `tibble()`: https://r4ds.had.co.nz/tibbles.html

```
# first create the individual variables
n <- c("russ", "lucy", "jaclyn", "tyler")
x <- c(1, 2, 3, 4)
y <- c(4, 5, 6, 7)
z <- c(7, 8, 9, 10)

# create the data frame
myDataFrame <-
  tibble(
    n, #list each of your columns in the order you want them
    x,
    y,
    z
  )

myDataFrame
```

```
## # A tibble: 4 x 4
##   n          x     y     z
##   <chr>  <dbl> <dbl> <dbl>
## 1 russ       1     4     7
## 2 lucy       2     5     8
## 3 jaclyn     3     6     9
## 4 tyler      4     7    10
```

Take a quick look at the properties of the data frame using `glimpse()`:

```
glimpse(myDataFrame)
```

```
## Observations: 4
## Variables: 4
## $ n <chr> "russ", "lucy", "jaclyn", "tyler"
## $ x <dbl> 1, 2, 3, 4
## $ y <dbl> 4, 5, 6, 7
## $ z <dbl> 7, 8, 9, 10
```

Selecting an element

There are various ways to access the contents within a data frame.

Selecting a row or column by name

```
myDataFrame$x
```

```
## [1] 1 2 3 4
```

The first index refers to the row, the second to the column.

```
myDataFrame[1, 2]
```

```
## # A tibble: 1 x 1
##       x
##   <dbl>
## 1     1
```

```
myDataFrame[2, 3]
```

```
## # A tibble: 1 x 1
##       y
##   <dbl>
## 1     5
```

Selecting a row or column by index

```
myDataFrame[1, ]
```

```
## # A tibble: 1 x 4
##   n         x     y     z
##   <chr> <dbl> <dbl> <dbl>
## 1 russ      1     4     7
```

```
myDataFrame[, 1]
```

```
## # A tibble: 4 x 1
##   n
##   <chr>
## 1 russ
## 2 lucy
## 3 jaclyn
## 4 tyler
```

Select a set of rows

```
myDataFrame %>%
  slice(1:2)
```

```
## # A tibble: 2 x 4
##   n         x     y     z
##   <chr> <dbl> <dbl> <dbl>
## 1 russ      1     4     7
## 2 lucy      2     5     8
```

`slice()` is a function that selects out rows based on their row number.

You will also notice something we haven't discussed before: `%>%`. This is called a "pipe", which is commonly used within the tidyverse; you can read more here. A pipe takes the output from one command and feeds it as input to the next command. In this case, simply writing the name of the data frame (myDataFrame) causes it to be input to the `slice()` command following the pipe. The benefit of pipes will become especially apparent when we want to start stringing together multiple data processing operations into a single command.

In this example, no new variable is created - the output is printed to the screen, just like it would be if you typed the name of the variable. If you wanted to save it to a new variable, you would use the `<-` assignment operator, like this:

```
myDataFrameSlice <- myDataFrame %>%
  slice(1:2)

myDataFrameSlice
```

```
## # A tibble: 2 x 4
##   n         x     y     z
##   <chr> <dbl> <dbl> <dbl>
## 1 russ      1     4     7
## 2 lucy      2     5     8
```

Select a set of rows based on specific value(s)

```
myDataFrame %>%
  filter(n == "russ")
```

```
## # A tibble: 1 x 4
##   n         x     y     z
##   <chr> <dbl> <dbl> <dbl>
## 1 russ      1     4     7
```

`filter()` is a function that retains only those rows that meet your stated criteria. We can also filter for multiple criteria at once — in this example, the `|` symbol indicates "or":

```
myDataFrame %>%
  filter(n == "russ" | n == "lucy")
```

```
## # A tibble: 2 x 4
##   n         x     y     z
##   <chr> <dbl> <dbl> <dbl>
## 1 russ      1     4     7
## 2 lucy      2     5     8
```

Select a set of columns

```
myDataFrame %>%
  select(x:y)
```

```
## # A tibble: 4 x 2
##       x     y
##   <dbl> <dbl>
## 1     1     4
## 2     2     5
## 3     3     6
## 4     4     7
```

`select()` is a function that selects out only those columns you specify using their names

You can also specify a vector of columns to select.

```
myDataFrame %>%
  select(c(x,z))
```

```
## # A tibble: 4 x 2
##       x     z
##   <dbl> <dbl>
## 1     1     7
```

```
## 2     2     8
## 3     3     9
## 4     4    10
```

Adding a row or column

add a named row

```
tiffanyDataFrame <-
  tibble(
    n = "tiffany",
    x = 13,
    y = 14,
    z = 15
  )

myDataFrame %>%
  bind_rows(tiffanyDataFrame)
```

```
## # A tibble: 5 x 4
##   n           x     y     z
##   <chr>   <dbl> <dbl> <dbl>
## 1 russ        1     4     7
## 2 lucy        2     5     8
## 3 jaclyn      3     6     9
## 4 tyler       4     7    10
## 5 tiffany    13    14    15
```

`bind_rows()` is a function that combines the rows from another dataframe to the current dataframe

Creating or modifying variables using `mutate()`

Often we will want to either create a new variable based on an existing variable, or modify the value of an existing variable. Within the tidyverse, we do this using a function called `mutate()`. Let's start with a toy example by creating a data frame containing a single variable.

```
toy_df <- data.frame(x = c(1,2,3,4))
glimpse(toy_df)
```

```
## Observations: 4
## Variables: 1
## $ x <dbl> 1, 2, 3, 4
```

Let's say that we wanted to create a new variable called `y` that would contain the value of x multiplied by 10. We could do this using `mutate()` and then assign the result back to the same data frame:

```
toy_df <- toy_df %>%
  # create a new variable called y that contains x*10
  mutate(y = x*10)
glimpse(toy_df)
```

```
## Observations: 4
## Variables: 2
## $ x <dbl> 1, 2, 3, 4
## $ y <dbl> 10, 20, 30, 40
```

We could also overwrite a variable with a new value:

```
toy_df2 <- toy_df %>%
  # create a new variable called y that contains x*10
  mutate(y = y + 1)
glimpse(toy_df2)
```

```
## Observations: 4
## Variables: 2
## $ x <dbl> 1, 2, 3, 4
## $ y <dbl> 11, 21, 31, 41
```

We will use `mutate()` often so it's an important function to understand.

Here we can use it with our example data frame to create a new variable that is the sum of several other variables.

```
myDataFrame <-
  myDataFrame %>%
  mutate(total = x + y + z)

myDataFrame
```

```
## # A tibble: 4 x 5
##   n          x     y     z total
##   <chr>  <dbl> <dbl> <dbl> <dbl>
## 1 russ       1     4     7    12
## 2 lucy       2     5     8    15
## 3 jaclyn     3     6     9    18
## 4 tyler      4     7    10    21
```

mutate() is a function that creates a new variable in a data frame using the existing variables. In this case, it creates a variable called total that is the sum of the existing variables x, y, and z.

Remove a column using the select() function

Adding a minus sign to the name of a variable within the `select()` command will remove that variable, leaving all of the others.

```
myDataFrame <-
  myDataFrame %>%
  dplyr::select(-total)

myDataFrame
```

```
## # A tibble: 4 x 4
##   n          x     y     z
##   <chr>  <dbl> <dbl> <dbl>
## 1 russ       1     4     7
## 2 lucy       2     5     8
## 3 jaclyn     3     6     9
## 4 tyler      4     7    10
```

Tidyverse in action

To see the tidyverse in action, let's clean up the NHANES dataset. Each individual in the NHANES dataset has a unique identifier stored in the variable `ID`. First let's look at the number of rows in the dataset:

```
nrow(NHANES)
```

```
## [1] 10000
```

Now let's see how many unique IDs there are. The `unique()` function returns a vector containing all of the unique values for a particular variable, and the `length()` function returns the length of the resulting vector.

```
length(unique(NHANES$ID))
```

```
## [1] 6779
```

This shows us that while there are 10,000 observations in the data frame, there are only 6779 unique IDs. This means that if we were to use the entire dataset, we would be reusing data from some individuals, which could give us incorrect results. For this reason, we wold like to discard any observations that are duplicated.

Let's create a new variable called `NHANES_unique` that will contain only the distinct observations, with no individuals appearing more than once. The `dplyr` library provides a function called `distinct()` that will do this for us. You may notice that we didn't explicitly load the `dplyr` library above; however, if you look at the messages that appeared when we loaded the `tidyverse` library, you will see that it loaded `dplyr` for us. To create the new data frame with unique observations, we will pipe the NHANES data frame into the `distinct()` function and then save the output to our new variable.

```
NHANES_unique <-
  NHANES %>%
  distinct(ID, .keep_all = TRUE)
```

If we number of rows in the new data frame, it should be the same as the number of unique IDs (6779):

```
nrow(NHANES_unique)
```

```
## [1] 6779
```

In the next example you will see the power of pipes come to life, when we start tying together multiple functions into a single operation (or "pipeline").

Looking at individual variables using pull() and head()

The NHANES data frame contains a large number of variables, but usually we are only interested in a particular variable. We can extract a particular variable from a data frame using the `pull()` function. Let's say that we want to extract the variable `PhysActive`. We could do this by piping the data frame into the pull command, which will result in a list of many thousands of values. Instead of printing out this entire list, we will pipe the result into the `head()` function, which just shows us the first few values contained in a variable. In this case we are not assigning the value back to a variable, so it will simply be printed to the screen.

```
NHANES %>%
  # extract the PhysActive variable
  pull(PhysActive) %>%
  # extract the first 10 values
  head(10)
```

```
##  [1] No   No   No   <NA> No   <NA> <NA> Yes  Yes  Yes
## Levels: No Yes
```

There are two important things to notice here. The first is that there are three different values apparent in the answers: "Yes", "No", and , which means that the value is missing for this person (perhaps they didn't want to answer that question on the survey). When we are working with data we generally need to remove missing values, as we will see below.

The second thing to notice is that R prints out a list of "Levels" of the variable. This is because this variable is defined as a particular kind of variable in R known as a *factor*. You can think of a factor variable as a categorial variable with a specific set of levels. The missing data are not treated as a level, so it can be useful to make the missing values explicit, which can be done using a function called `fct_explicit_na()` in the `forcats` package. Let's add a line to do that:

```
NHANES %>%
  mutate(PhysActive = fct_explicit_na(PhysActive)) %>%
  # extract the PhysActive variable
  pull(PhysActive) %>%
  # extract the first 10 values
  head(10)
```

```
##  [1] No        No        No        (Missing) No        (Missing) (Missing)
##  [8] Yes       Yes       Yes
## Levels: No Yes (Missing)
```

This new line overwrote the old value of `PhysActive` with a version that has been processed by the `fct_explicit_na()` function to convert values to explicitly missing values. Now you can see that Missing values are treated as an explicit level, which will be useful later.

Now we are ready to start summarizing data!

Computing a frequency distribution (Section @ref(frequency-distributions))

We would like to compute a frequency distribution showing how many people report being either active or inactive. The following statement is fairly complex so we will step through it one bit at a time.

```
PhysActive_table <- NHANES_unique %>%
  # convert the implicit missing values to explicit
  mutate(PhysActive = fct_explicit_na(PhysActive)) %>%
  # select the variable of interest
  dplyr::select(PhysActive) %>%
  # group by values of the variable
  group_by(PhysActive) %>%
  # count the values
  summarize(AbsoluteFrequency = n())

# kable() prints out the table in a prettier way.
kable(PhysActive_table)
```

PhysActive	AbsoluteFrequency
No	2473
Yes	2972
(Missing)	1334

The first step should be familiar from the previous section (we are adding the `head()` function here to show us the first few rows of the data frame):

```
NHANES_unique %>%
  mutate(PhysActive = fct_explicit_na(PhysActive)) %>%
  head(10) %>%
```

```r
  glimpse()
```

```
## Observations: 10
## Variables: 76
## $ ID               <int> 51624, 51625, 51630, 51638, 51646, 51647, 51654, 5...
## $ SurveyYr         <fct> 2009_10, 2009_10, 2009_10, 2009_10, 2009_10, 2009_...
## $ Gender           <fct> male, male, female, male, male, female, male, male...
## $ Age              <int> 34, 4, 49, 9, 8, 45, 66, 58, 54, 10
## $ AgeDecade        <fct>  30-39,  0-9,  40-49,  0-9,  0-9,  40-49,  60-69, ...
## $ AgeMonths        <int> 409, 49, 596, 115, 101, 541, 795, 707, 654, 123
## $ Race1            <fct> White, Other, White, White, White, White, White, W...
## $ Race3            <fct> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA
## $ Education        <fct> High School, NA, Some College, NA, NA, College Gra...
## $ MaritalStatus    <fct> Married, NA, LivePartner, NA, NA, Married, Married...
## $ HHIncome         <fct> 25000-34999, 20000-24999, 35000-44999, 75000-99999...
## $ HHIncomeMid      <int> 30000, 22500, 40000, 87500, 60000, 87500, 30000, 1...
## $ Poverty          <dbl> 1.36, 1.07, 1.91, 1.84, 2.33, 5.00, 2.20, 5.00, 2....
## $ HomeRooms        <int> 6, 9, 5, 6, 7, 6, 5, 10, 6, 10
## $ HomeOwn          <fct> Own, Own, Rent, Rent, Own, Own, Own, Rent, Rent, Own
## $ Work             <fct> NotWorking, NA, NotWorking, NA, NA, Working, NotWo...
## $ Weight           <dbl> 87.4, 17.0, 86.7, 29.8, 35.2, 75.7, 68.0, 78.4, 74...
## $ Length           <dbl> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA
## $ HeadCirc         <dbl> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA
## $ Height           <dbl> 164.7, 105.4, 168.4, 133.1, 130.6, 166.7, 169.5, 1...
## $ BMI              <dbl> 32.22, 15.30, 30.57, 16.82, 20.64, 27.24, 23.67, 2...
## $ BMICatUnder20yrs <fct> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA
## $ BMI_WHO          <fct> 30.0_plus, 12.0_18.5, 30.0_plus, 12.0_18.5, 18.5_t...
## $ Pulse            <int> 70, NA, 86, 82, 72, 62, 60, 62, 76, 80
## $ BPSysAve         <int> 113, NA, 112, 86, 107, 118, 111, 104, 134, 104
## $ BPDiaAve         <int> 85, NA, 75, 47, 37, 64, 63, 74, 85, 68
## $ BPSys1           <int> 114, NA, 118, 84, 114, 106, 124, 108, 136, 102
## $ BPDia1           <int> 88, NA, 82, 50, 46, 62, 64, 76, 86, 66
## $ BPSys2           <int> 114, NA, 108, 84, 108, 118, 108, 104, 132, 102
## $ BPDia2           <int> 88, NA, 74, 50, 36, 68, 62, 72, 88, 66
## $ BPSys3           <int> 112, NA, 116, 88, 106, 118, 114, 104, 136, 106
## $ BPDia3           <int> 82, NA, 76, 44, 38, 60, 64, 76, 82, 70
## $ Testosterone     <dbl> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA
## $ DirectChol       <dbl> 1.29, NA, 1.16, 1.34, 1.55, 2.12, 0.67, 0.96, 1.16...
## $ TotChol          <dbl> 3.49, NA, 6.70, 4.86, 4.09, 5.82, 4.99, 4.24, 6.41...
## $ UrineVol1        <int> 352, NA, 77, 123, 238, 106, 113, 163, 215, 7
## $ UrineFlow1       <dbl> NA, NA, 0.094, 1.538, 1.322, 1.116, 0.489, NA, 0.9...
## $ UrineVol2        <int> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA
## $ UrineFlow2       <dbl> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA
## $ Diabetes         <fct> No, No, No, No, No, No, No, No, No, No
## $ DiabetesAge      <int> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA
## $ HealthGen        <fct> Good, NA, Good, NA, NA, Vgood, Vgood, Vgood, Fair, NA
## $ DaysPhysHlthBad  <int> 0, NA, 0, NA, NA, 0, 10, 0, 4, NA
## $ DaysMentHlthBad  <int> 15, NA, 10, NA, NA, 3, 0, 0, 0, NA
## $ LittleInterest   <fct> Most, NA, Several, NA, NA, None, None, None, None, NA
## $ Depressed        <fct> Several, NA, Several, NA, NA, None, None, None, No...
## $ nPregnancies     <int> NA, NA, 2, NA, NA, 1, NA, NA, NA, NA
## $ nBabies          <int> NA, NA, 2, NA, NA, NA, NA, NA, NA, NA
## $ Age1stBaby       <int> NA, NA, 27, NA, NA, NA, NA, NA, NA, NA
## $ SleepHrsNight    <int> 4, NA, 8, NA, NA, 8, 7, 5, 4, NA
```

```
## $ SleepTrouble     <fct> Yes, NA, Yes, NA, NA, No, No, No, Yes, NA
## $ PhysActive       <fct> No, (Missing), No, (Missing), (Missing), Yes, Yes,...
## $ PhysActiveDays   <int> NA, NA, NA, NA, NA, 5, 7, 5, 1, NA
## $ TVHrsDay         <fct> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA
## $ CompHrsDay       <fct> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA
## $ TVHrsDayChild    <int> NA, 4, NA, 5, 1, NA, NA, NA, NA, 4
## $ CompHrsDayChild  <int> NA, 1, NA, 0, 6, NA, NA, NA, NA, 3
## $ Alcohol12PlusYr  <fct> Yes, NA, Yes, NA, NA, Yes, Yes, Yes, Yes, NA
## $ AlcoholDay       <int> NA, NA, 2, NA, NA, 3, 1, 2, 6, NA
## $ AlcoholYear      <int> 0, NA, 20, NA, NA, 52, 100, 104, 364, NA
## $ SmokeNow         <fct> No, NA, Yes, NA, NA, NA, No, NA, NA, NA
## $ Smoke100         <fct> Yes, NA, Yes, NA, NA, No, Yes, No, No, NA
## $ Smoke100n        <fct> Smoker, NA, Smoker, NA, NA, Non-Smoker, Smoker, No...
## $ SmokeAge         <int> 18, NA, 38, NA, NA, NA, 13, NA, NA, NA
## $ Marijuana        <fct> Yes, NA, Yes, NA, NA, Yes, NA, Yes, Yes, NA
## $ AgeFirstMarij    <int> 17, NA, 18, NA, NA, 13, NA, 19, 15, NA
## $ RegularMarij     <fct> No, NA, No, NA, NA, No, NA, Yes, Yes, NA
## $ AgeRegMarij      <int> NA, NA, NA, NA, NA, NA, NA, 20, 15, NA
## $ HardDrugs        <fct> Yes, NA, Yes, NA, NA, No, No, Yes, Yes, NA
## $ SexEver          <fct> Yes, NA, Yes, NA, NA, Yes, Yes, Yes, Yes, NA
## $ SexAge           <int> 16, NA, 12, NA, NA, 13, 17, 22, 12, NA
## $ SexNumPartnLife  <int> 8, NA, 10, NA, NA, 20, 15, 7, 100, NA
## $ SexNumPartYear   <int> 1, NA, 1, NA, NA, 0, NA, 1, 1, NA
## $ SameSex          <fct> No, NA, Yes, NA, NA, Yes, No, No, No, NA
## $ SexOrientation   <fct> Heterosexual, NA, Heterosexual, NA, NA, Bisexual, ...
## $ PregnantNow      <fct> NA, NA, NA, NA, NA, NA, NA, NA, NA, NA
```

You can see that this data frame contains all of the original variables. Since we are only interested in the `PhysActive` variable, let's extract that one and get rid of the rest. We can do this using the `select()` command from the `dplyr` package. Because there is also another select command available in R, we need to explicitly refer to the one from the `dplyr` package, which we do by including the package name followed by two colons: `dplyr::select()`.

```
NHANES_unique %>%
  # convert the implicit missing values to explicit
  mutate(PhysActive = fct_explicit_na(PhysActive)) %>%
  # select the variable of interest
  dplyr::select(PhysActive) %>%
  head(10)
```

```
## # A tibble: 10 x 1
##    PhysActive
##    <fct>
##  1 No
##  2 (Missing)
##  3 No
##  4 (Missing)
##  5 (Missing)
##  6 Yes
##  7 Yes
##  8 Yes
##  9 Yes
## 10 (Missing)
```

The next function, `group_by()` tells R that we are going to want to analyze the data separate according to the different levels of the `PhysActive` variable:

```r
NHANES_unique %>%
  # convert the implicit missing values to explicit
  mutate(PhysActive = fct_explicit_na(PhysActive)) %>%
  # select the variable of interest
  dplyr::select(PhysActive) %>%
  group_by(PhysActive) %>%
  head(10)
```

```
## # A tibble: 10 x 1
## # Groups:   PhysActive [3]
##    PhysActive
##    <fct>
##  1 No
##  2 (Missing)
##  3 No
##  4 (Missing)
##  5 (Missing)
##  6 Yes
##  7 Yes
##  8 Yes
##  9 Yes
## 10 (Missing)
```

The final command tells R to create a new data frame by summarizing the data that we are passing in (which in this case is the PhysActive variable, grouped by its different levels). We tell the `summarize()` function to create a new variable (called `AbsoluteFrequency`) will contain a count of the number of observations for each group, which is generated by the `n()` function.

```r
NHANES_unique %>%
  # convert the implicit missing values to explicit
  mutate(PhysActive = fct_explicit_na(PhysActive)) %>%
  # select the variable of interest
  dplyr::select(PhysActive) %>%
  group_by(PhysActive) %>%
  summarize(AbsoluteFrequency = n())
```

```
## # A tibble: 3 x 2
##   PhysActive AbsoluteFrequency
##   <fct>                  <int>
## 1 No                      2473
## 2 Yes                     2972
## 3 (Missing)               1334
```

Now let's say we want to add another column with percentage of observations in each group. We compute the percentage by dividing the absolute frequency for each group by the total number. We can use the table that we already generated, and add a new variable, again using `mutate()`:

```r
PhysActive_table <- PhysActive_table %>%
  mutate(
    Percentage = AbsoluteFrequency / sum(AbsoluteFrequency) * 100
  )

kable(PhysActive_table, digits=2)
```

PhysActive	AbsoluteFrequency	Percentage
No	2473	36.48

PhysActive	AbsoluteFrequency	Percentage
Yes	2972	43.84
(Missing)	1334	19.68

Computing a cumulative distribution (Section @ref(cumulative-distributions))

Let's compute a cumulative distribution for the `SleepHrsNight` variable in NHANES. This looks very similar to what we saw in the previous section.

```
# create summary table for relative frequency of different
# values of SleepHrsNight

SleepHrsNight_cumulative <-
  NHANES_unique %>%
  # drop NA values for SleepHrsNight variable
  drop_na(SleepHrsNight) %>%
  # remove other variables
  dplyr::select(SleepHrsNight) %>%
  # group by values
  group_by(SleepHrsNight) %>%
  # create summary table
  summarize(AbsoluteFrequency = n()) %>%
  # create relative and cumulative frequencies
  mutate(
    RelativeFrequency = AbsoluteFrequency / sum(AbsoluteFrequency),
    CumulativeDensity = cumsum(RelativeFrequency)
  )

kable(SleepHrsNight_cumulative)
```

SleepHrsNight	AbsoluteFrequency	RelativeFrequency	CumulativeDensity
2	9	0.0017875	0.0017875
3	49	0.0097319	0.0115194
4	200	0.0397219	0.0512413
5	406	0.0806356	0.1318769
6	1172	0.2327706	0.3646475
7	1394	0.2768620	0.6415094
8	1405	0.2790467	0.9205561
9	271	0.0538232	0.9743793
10	97	0.0192651	0.9936445
11	15	0.0029791	0.9966236
12	17	0.0033764	1.0000000

Exercises