

## Lecture 6: Augmented vectors, Factor + Labelled Variables

# Introduction

## Reading to do before next class:

- ▶ Work through slides from lecture 5 that we didn't get to in class (attaching aggregate measures)
- ▶ GW 15.1 - 15.2 (factors) [this is like 2-3 pages]
- ▶ [OPTIONAL] GW 15.3 - 15.5 (remainder of "factors" chapter)
- ▶ [OPTIONAL] GW 20.6 - 20.7 (attributes and augmented vectors)
- ▶ [OPTIONAL] GW 10 (tibbles)

# What we will do today

## 1. Introduction

## 2. Augmented vectors

2.1 Review data types and structures

2.2 Attributes and augmented vectors

2.3 Object class

2.4 `Class == factor`

2.5 `Class == labelled`

2.6 Comparing labelled class to factor class

## 3. Creating factor variables

## Libraries we will use today

“Load” the package we will use today (output omitted)

► **you must run this code chunk after installing these packages**

```
library(tidyverse)
library(haven)
library(labelled)
library(lubridate)
```

**If package not yet installed**, then must install before you load. Install in “console” rather than .Rmd file

► Generic syntax: `install.packages("package_name")`

► Install “tidyverse”: `install.packages("tidyverse")`

Note: when we load package, name of package is not in quotes; but when we install package, name of package is in quotes:

► `install.packages("tidyverse")`

► `library(tidyverse)`

## Augmented vectors

## Data we will use to introduce augmented vectors

```
rm(list = ls()) # remove all objects

#load("../data/prospect_list/western_washington_college_board_list.RData")
load(url("https://github.com/ozanj/rclass/raw/master/data/prospect_list/wwlist_"))
```

## Review data types and structures



# Vectors are the primary data structures in R

Two types of vectors:

1. **atomic vectors**
2. **lists**

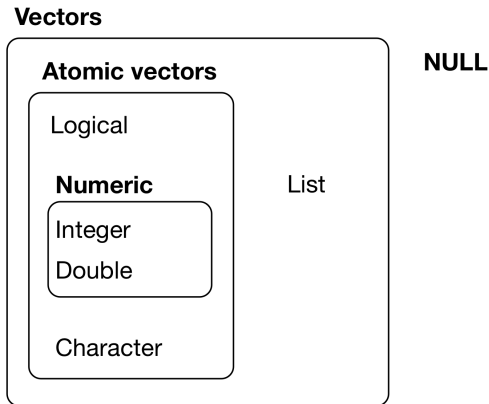


Figure 1: Overview of data structures (Grolemund and Wickham, 2018)

## Review data structures: atomic vectors

An **atomic vector** is a collection of values

- ▶ each value in an atomic vector is an **element**
- ▶ all elements within vector must have same **data type**

```
(a <- c(1,2,3)) # parentheses () assign and print object in one step
#> [1] 1 2 3
length(a)
#> [1] 3
typeof(a)
#> [1] "double"
str(a)
#>  num [1:3] 1 2 3
```

Can assign **names** to vector elements, creating a **named atomic vector**

```
(b <- c(v1=1,v2=2,v3=3))
#> v1 v2 v3
#> 1 2 3
length(b)
#> [1] 3
typeof(b)
#> [1] "double"
str(b)
#> Named num [1:3] 1 2 3
#> - attr(*, "names")= chr [1:3] "v1" "v2" "v3"
```

## Review data structures: lists

- ▶ Like atomic vectors, **lists** are objects that contain **elements**
- ▶ However, **data type** can differ across elements within a list
  - ▶ an element of a list can be another list

```
list_a <- list(1,2,"apple")
```

```
typeof(list_a)
```

```
#> [1] "list"
```

```
length(list_a)
```

```
#> [1] 3
```

```
str(list_a)
```

```
#> List of 3
```

```
#> $ : num 1
```

```
#> $ : num 2
```

```
#> $ : chr "apple"
```

```
list_b <- list(1, c("apple", "orange"), list(1, 2))
```

```
length(list_b)
```

```
#> [1] 3
```

```
str(list_b)
```

```
#> List of 3
```

```
#> $ : num 1
```

```
#> $ : chr [1:2] "apple" "orange"
```

```
#> $ :List of 2
```

```
#> ..$ : num 1
```

```
#> ..$ : num 2
```

## Review data structures: lists

Like atomic vectors, elements within a list can be named, thereby creating a **named list**

```
# not named
str(list_b)
#> List of 3
#> $ : num 1
#> $ : chr [1:2] "apple" "orange"
#> $ :List of 2
#> ..$ : num 1
#> ..$ : num 2

# named
list_c <- list(v1=1, v2=c("apple", "orange"), v3=list(1, 2, 3))
str(list_c)
#> List of 3
#> $ v1: num 1
#> $ v2: chr [1:2] "apple" "orange"
#> $ v3:List of 3
#> ..$ : num 1
#> ..$ : num 2
#> ..$ : num 3
```

## Review data structures: a data frame is a list

A **data frame** is a list with the following characteristics:

- ▶ All the elements must be **vectors** with the same **length**
- ▶ Data frames are **augmented lists** because they have additional **attributes** [described later]

```
#a regular list
```

```
list_d <- list(col_a = c(1,2,3), col_b = c(4,5,6), col_c = c(7,8,9))
```

```
typeof(list_d)
```

```
#> [1] "list"
```

```
str(list_d)
```

```
#> List of 3
```

```
#> $ col_a: num [1:3] 1 2 3
```

```
#> $ col_b: num [1:3] 4 5 6
```

```
#> $ col_c: num [1:3] 7 8 9
```

```
#a data frame
```

```
df_a <- data.frame(col_a = c(1,2,3), col_b = c(4,5,6), col_c = c(7,8,9))
```

```
typeof(df_a)
```

```
#> [1] "list"
```

```
str(df_a)
```

```
#> 'data.frame':    3 obs. of  3 variables:
```

```
#> $ col_a: num  1 2 3
```

```
#> $ col_b: num  4 5 6
```

```
#> $ col_c: num  7 8 9
```

## Attributes and augmented vectors

# Atomic vectors versus augmented vectors

## Atomic vectors [our focus so far]

- ▶ I think of atomic vectors as “just the data”
- ▶ Atomic vectors are the building blocks for augmented vectors

## Augmented vectors

- ▶ **Augmented vectors** are atomic vectors with additional **attributes** attached

## Attributes

- ▶ **Attributes** are additional “metadata” that can be attached to any object (e.g., vector or list)
- ▶ Examples of some important attributes in R:
  - ▶ **Names**: name the elements of a vector (e.g., variable names)
  - ▶ **value labels**: character labels (e.g., “Charter School”) attached to numeric values
  - ▶ **Object class**: How object should be treated by object oriented programming language [discussed below]

## Main takaway:

- ▶ Augmented vectors are atomic vectors (just the data) with additional attributes attached

## Attributes in vectors

Identify attributes in any object using the `attributes()` function

```
#vector with no attributes
```

```
vector1 <- c(1,2,3,4)
```

```
vector1
```

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

```
attributes(vector1)
```

```
#> NULL
```

```
#vector with name attributes
```

```
vector2 <- c(a = 1, b = 2, c = 3, d = 4)
```

```
vector2
```

```
#> a b c d
```

```
#> 1 2 3 4
```

```
attributes(vector2)
```

```
#> $names
```

```
#> [1] "a" "b" "c" "d"
```



## Attributes in lists

```
#no attributes
list1 <- list(c(1,2,3), c(4,5,6))
attributes(list1)
#> NULL

#list with attributes
list2 <- list(col_a = c(1,2,3), col_b = c(4,5,6))
str(list2)
#> List of 2
#> $ col_a: num [1:3] 1 2 3
#> $ col_b: num [1:3] 4 5 6
attributes(list2)
#> $names
#> [1] "col_a" "col_b"

#data frame with attributes
list3 <- data.frame(col_a = c(1,2,3), col_b = c(4,5,6))
str(list3)
#> 'data.frame':    3 obs. of  2 variables:
#> $ col_a: num  1 2 3
#> $ col_b: num  4 5 6
attributes(list3)
#> $names
#> [1] "col_a" "col_b"
#>
#> $class
#> [1] "data.frame"
```

Object class

# Object class

Every object in R has a **class**

- ▶ Object class defines rules for how object can be treated by object oriented programming language (e.g., which functions you can apply to object)
- ▶ class is an **attribute** of an object

Identify the class of an object using the `class()` function

```
(vector2 <- c(a = 1, b = 2, c = 3, d = 4))  
#> a b c d  
#> 1 2 3 4  
class(vector2)  
#> [1] "numeric"
```

When I encounter a new object I often investigate object by applying `typeof()`, `class()`, and `attributes()` functions to that object

```
vector2  
#> a b c d  
#> 1 2 3 4  
typeof(vector2)  
#> [1] "double"  
class(vector2)  
#> [1] "numeric"  
attributes(vector2)  
#> $names  
#> [1] "a" "b" "c" "d"
```

# Object class

Why is **class** important?

- ▶ Specific functions usually work with only particular **classes** of objects
  - ▶ e.g., “date” functions usually only work on objects with a date class
  - ▶ “string” functions usually only work on objects with a character class
  - ▶ Functions that do mathematical computation usually work on objects with a numeric class
- ▶ Note: functions care about object **class**, not object **type**

object with `numeric` class (output omitted)

```
str(wwlist)
```

```
typeof(wwlist$med_inc_zip)
```

```
class(wwlist$med_inc_zip)
```

```
sum(wwlist$med_inc_zip[1:10], na.rm = TRUE) # numeric function
```

```
# load library with date functions
```

```
library(lubridate)
```

```
#Sys.setenv(TZ="America/Los_Angeles") #setting time zone to Los Angeles time
```

```
year(wwlist$med_inc_zip[1:10]) # date function
```

# Object class

Why is **class** important?

- ▶ Specific functions usually work with only particular **classes** of objects
- ▶ Note: functions care about object **class**, not object **type**

Object with `character` class

```
str(wwlist$hs_city)
typeof(wwlist$hs_city)
class(wwlist$hs_city)

tolower(wwlist$hs_city[1:10]) # string function
sum(wwlist$hs_city, na.rm = TRUE) # numeric function
```

Object with a date class

```
typeof(wwlist$receive_date)
class(wwlist$receive_date)

year(wwlist$receive_date[1:10]) # date function
sum(wwlist$receive_date) # numeric function
```

# Class and object oriented programming

Definition of object oriented programming from this [LINK](#)

*“Object-oriented programming (OOP) refers to a type of computer programming in which programmers define not only the data type of a data structure, but also the types of operations (functions) that can be applied to the data structure.”*

Object **class** is fundamental to object oriented programming because:

- ▶ object class determines which functions can be applied to the object
- ▶ object class also determines what those functions do to the object

Many different object classes exist in R

- ▶ we can also create our own classes
- ▶ but in this course we will work with classes that have been created by others

Class == factor

# Factors

**Factors** are an object *class* used to display categorical data (e.g., marital status)

- ▶ A factor is an **augmented vector** built by attaching a “levels” attribute to an (atomic) integer vectors

Usually, we would prefer a categorical variable (e.g., race, school type) to be a factor variable rather than a character variable

- ▶ So far in the course I have made all categorical variables character variables because we had not introduced factors yet

Below, I'll create a factor version of the character variable `ethn_code`

- ▶ (don't worry about understanding this code; I'll explain it later)

```
str(wwlist$ethn_code)
#> chr [1:268396] "other-2 or more" "white" "white" "other-2 or more" "white" .
class(wwlist$ethn_code)
#> [1] "character"
# create factor var; tidyverse approach
wwlist <- wwlist %>% mutate(ethn_code_fac = factor(ethn_code))
#wwlist$ethn_code_fac <- factor(wwlist$ethn_code) # base r approach

str(wwlist$ethn_code_fac)
#> Factor w/ 10 levels "american indian or alaska native",...: 8 10 10 8 10 8 8
```



# Factors

A factor is an **augmented vector** built by attaching a “levels” attribute to an (atomic) integer vector

Compare (character) `ethn_code` to (factor) `ethn_code_fac` (output omitted)

```
#character var
typeof(wwlist$ethn_code)
class(wwlist$ethn_code)
str(wwlist$ethn_code)
attributes(wwlist$ethn_code)

#factor var
typeof(wwlist$ethn_code_fac)
class(wwlist$ethn_code_fac)
str(wwlist$ethn_code_fac)
attributes(wwlist$ethn_code_fac)
```

## Main takeaway

- ▶ `ethn_code_fac` has `type=integer` and `class=factor` because the variable has a “levels” attribute
- ▶ Underlying data are integers but levels attribute is used to display the data.

```
wwlist$ethn_code_fac[1:4] # print first few obs of ethn_code_fac
#> [1] other-2 or more white white other-2 or more
#> 10 Levels: american indian or alaska native ...
```

## Working with factor variables

```
attributes(wwlist$ethn_code_fac)
```

Refer to categories of a factor by the values of the **level attribute** rather than the underlying values of the variable

### Task

► count the number of prospects in object `wwlist` who identify as “white”

```
# referring to variable value; this doesn't work
```

```
wwlist %>% filter(ethn_code_fac==10) %>% count
```

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

```
#>       n
```

```
#>   <int>
```

```
#> 1      0
```

```
#referring to value of level attribute; this works
```

```
wwlist %>% filter(ethn_code_fac=="white") %>% count
```

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

```
#>       n
```

```
#>   <int>
```

```
#> 1 159680
```

# Working with factor variables

## Task

- ▶ count the number of prospects in object `wwlist` who identify as “white”

If you want to refer to underlying values, then apply `as.integer()` function to the factor variable

```
attributes(wwlist$ethn_code_fac)
#> $levels
#> [1] "american indian or alaska native"
#> [2] "asian or native hawaiian or other pacific islander"
#> [3] "black or african american"
#> [4] "cuban"
#> [5] "mexican/mexican american"
#> [6] "not reported"
#> [7] "other spanish/hispanic"
#> [8] "other-2 or more"
#> [9] "puerto rican"
#> [10] "white"
#>
#> $class
#> [1] "factor"
wwlist %>% filter(as.integer(ethn_code_fac)==10) %>% count
#> # A tibble: 1 x 1
#>       n
#>   <int>
#> 1 159680
```

## How to identify the variable values associated with factor levels

Let's create a factor version of the character variable `psat_range`

```
wwlist <- wwlist %>% mutate(psat_range_fac = factor(psat_range)) # create factor
```

Run below code in console rather than code chunk to see values associated with each factor

```
wwlist %>% count(psat_range_fac)
attributes(wwlist$psat_range_fac)
levels(wwlist$psat_range_fac) #starts at 1
nlevels(wwlist$psat_range_fac) #7 levels total
levels(wwlist$psat_range_fac)[1:3] #prints levels 1-3
```

Once you know values associated with factor, you can filter based on values

```
wwlist %>% filter(as.integer(psat_range_fac)==4) %>% count()
#> # A tibble: 1 x 1
#>       n
#>   <int>
#> 1  8348
```

Or you can just filter based on value of **factor levels**

```
wwlist %>% filter(psat_range=="1270-1520") %>% count()
#> # A tibble: 1 x 1
#>       n
#>   <int>
#> 1  8348
```

## Creating factor variables from character variables or from integer variables

See Appendix

## Factor student exercise

1. After running the code below, use `typeof`, `class`, `str`, and `attributes` functions to check the new variable `receive_year`
2. Create a factor variable from the input variable `receive_year` and name it `receive_year_fac`
3. Run the same functions (`typeof`, `class`, etc.) from the first question using the new variable you created
4. Get a count of `receive_year_fac`. **hint:** you could also run this in the console to see values associated with each factor

Run this code to create a year variable from the input variable "receive\_date"

```
#wwlist %>% glimpse()
```

```
library(lubridate) #load library if you haven't already
```

```
wwlist <- wwlist %>%
```

```
  mutate(receive_year = year(receive_date)) #creating year variable with the lubridate
```

```
#Check variable
```

```
wwlist %>%
```

```
  count(receive_year)
```

```
wwlist %>%
```

```
  group_by(receive_year) %>%
```

```
  count(receive_date)
```

# Factor student exercise solutions

1. Use `typeof`, `class`, `str`, and `attributes` functions to check the new variable `receive_year`

```
typeof(wwlist$receive_year)
#> [1] "double"
class(wwlist$receive_year)
#> [1] "numeric"
str(wwlist$receive_year)
#> num [1:268396] 2016 2016 2016 2016 2016 ...
attributes(wwlist$receive_year)
#> NULL
```

## Factor student exercise solutions

2. Now create a factor variable from the input variable `receive_year` and name it `receive_year_fac`

```
# create factor var; tidyverse approach  
wwlist <- wwlist %>%  
  mutate(receive_year_fac = factor(receive_year))
```



## Factor student exercise solutions

3. Run the same functions ( `typeof` , `class` , etc.) from the first question using the new variable you created

```
typeof(wwlist$receive_year_fac)
#> [1] "integer"
class(wwlist$receive_year_fac)
#> [1] "factor"
str(wwlist$receive_year_fac)
#> Factor w/ 3 levels "2016","2017",...: 1 1 1 1 1 1 1 1 1 1 ...
attributes(wwlist$receive_year_fac)
#> $levels
#> [1] "2016" "2017" "2018"
#>
#> $class
#> [1] "factor"
```

## Factor student exercise solutions

4. Get a count of `receive_year_fac`. **hint:** you could also run this in the console to see values associated with each factor

```
wwlist %>%  
  count(receive_year_fac)  
#> # A tibble: 3 x 2  
#>   receive_year_fac      n  
#>   <fct>           <int>  
#> 1 2016            89637  
#> 2 2017            89816  
#> 3 2018            88943
```

Class == labelled

## Data we will use to introduce labelled class

High school longitudinal surveys from National Center for Education Statistics (NCES)

- ▶ Follow U.S. students from high school through college, labor market

We will be working with [High School Longitudinal Study of 2009 \(HSL:09\)](#)

- ▶ Follows 9th graders from 2009
- ▶ Data collection waves
  - Base Year (2009)
  - First Follow-up (2012)
  - 2013 Update (2013)
  - High School Transcripts (2013-2014)
  - Second Follow-up (2016)

## haven package

`haven`, which is part of **tidyverse**, “enables R to read and write various data formats” from the following statistical packages:

- ▶ SAS
- ▶ SPSS
- ▶ Stata

When using `haven` to read data, resulting R objects have these characteristics:

- ▶ Are **tibbles**, a particular type of data frame we discuss in future weeks
- ▶ Transform variables with “value labels” into the `labelled()` class [our focus today]
  - ▶ `labelled` is an object **class** created by folks who created `haven` package
  - ▶ `labelled` is an object class, just like `factor` is an object class
  - ▶ `labelled` and `factor` classes are both viable alternatives for categorical variables
  - ▶ Helpful description of `labelled` class [HERE](#)
- ▶ Dates and times converted to R date/time classes
- ▶ Character vectors not converted to factors

## haven package

Use `read_dta()` function from `haven` to import Stata dataset into R

```
hsls <- read_dta(file="https://github.com/ozanj/rclass/raw/master/data/hsls/hsls.dta")
```

Let's examine the data [you **must** run this code chunk]

```
names(hsls)
names(hsls) <- tolower(names(hsls)) # convert names to lowercase
names(hsls)

str(hsls) # ugh

str(hsls$s3classes)
attributes(hsls$s3classes)
typeof(hsls$s3classes)
class(hsls$s3classes)
```

## labelled package

Purpose of the `labelled` package is to work with data imported from SPSS/Stata/SAS using the `haven` package.

- ▶ In particular, `labelled` package creates functions to work with objects that have `labelled` class
- ▶ From package documentation: "purpose of the `labelled` package is to provide functions to manipulate *metadata* as variable labels, value labels and defined missing values using the `labelled` class and the `label` attribute introduced in `haven` package.
- ▶ More info on the `labelled` package: [LINK](#)

Functions in `labelled` package

- ▶ [Full list](#)
- ▶ A couple relevant functions
  - ▶ `val_labels` : get or set variable *value labels*
  - ▶ `var_label` : get or set a *variable label*

```
attributes(hs1s$s3classes)
```

```
hs1s %>% select(s3classes) %>% var_label()  
hs1s %>% select(s3classes) %>% val_labels()
```

# Core concepts for understanding labelled class [SKIP]

**atomic vectors (and lists)** the underlying data

- ▶ data structures: vector or list
- ▶ data type: numeric (integer or double); character; logical

```
typeof(hs1s$s3classes)
```

```
#> [1] "double"
```

**augmented vectors** are atomic vectors with **attributes** attached

**attributes** are “metadata” attached to an object. Examples

- ▶ **names**: names of elements of a vector or list (e.g., variable names)
- ▶ **levels**: display output associated with values of a factor variable
- ▶ **class**: e.g., factor, labelled

```
attributes(hs1s$s3classes)
```

**class** is an object oriented programming concept. The **class** of an object determines which functions can be applied to the object and what those functions do

- ▶ e.g., can't apply `sum()` to an object where `class=character`



## What is labelled class?

- ▶ `labelled` is an object class created by the `haven` package for importing variables from SAS/SPSS/Stata that have **value labels**
- ▶ **value labels** [in Stata] are labels attached to specific values of a variable:
  - ▶ e.g., variable value `1` attached to value label "married", `2` = "single", `3` = "divorced"
- ▶ Variables in an R data frame with `class==labelled` :
  - ▶ data type can be numeric(double) or character
  - ▶ To see value labels associated with each value:
    - ▶ `attr(data_frame_name$variable_name,"labels")`
    - ▶ e.g., `attr(hs1s$s3classes,"labels")`

Let's investigate the attributes of `hs1s$s3classes`

```
typeof(hs1s$s3classes)
class(hs1s$s3classes)
str(hs1s$s3classes)
attributes(hs1s$s3classes)
```

use `attr(object_name,"attribute_name")` to refer to each attribute

```
attr(hs1s$s3classes,"label")
attr(hs1s$s3classes,"labels")
attr(hs1s$s3classes,"class")
attr(hs1s$s3classes,"format.stata")
```

## Working with labelled class data

Show variable labels ( `var_label` ); and show value labels ( `val_labels` )

```
hsls %>% select(s3classes,s3clglvl) %>% var_label #show variable label
hsls %>% select(s3classes,s3clglvl) %>% val_labels #show value labels
```

Create frequency tables with labelled class variables using `count()`

► Default setting is to show variable **values** not **value labels**

```
hsls %>% count(s3classes)
#investigate the object created
hsls_freq_temp <- hsls %>% count(s3classes)
hsls_freq_temp
rm(hsls_freq_temp)
```

To make frequency table show **value labels** add `%>% as_factor()` to pipe

► `as_factor()` is function from `haven` that converts an object to a factor

```
hsls %>% count(s3classes) %>% as_factor()
#investigate the object created
hsls_freq_temp <- hsls %>% count(s3classes) %>% as_factor()
hsls_freq_temp
rm(hsls_freq_temp)
```

## Working with labelled class data

To isolate values of labelled class variables in `filter()` function:

- ▶ refer to variable **value**, not the **value label**

### Task

- ▶ how many observations in var `s3classes` associated with “Unit non-response”
- ▶ how many observations in var `s3classes` associated with “Yes”

General steps to follow:

1. investigate object
2. use filter to isolate desired observations

Investigate object

```
class(hs1s$s3classes)
hs1s %>% select(s3classes,s3clglvl) %>% var_label #show variable label
hs1s %>% select(s3classes,s3clglvl) %>% val_labels #show value label
hs1s %>% count(s3classes) #freq table, values
hs1s %>% count(s3classes) %>% as_factor() # freq table, value labels
```

filter specific values

```
hs1s %>% filter(s3classes==8) %>% count() # 8 = unit non-response
hs1s %>% filter(s3classes==1) %>% count() # 1 = yes
```

## Labelled student exercise

1. Get variable and value labels of `s3hs`
2. Get a count of the variable showing the values and the value labels. **hint** use `factor()`
3. Filter if value is associated with “Missing”
4. Filter if value is associated with “Missing” or “Unit non-response”

## Labelled student exercise solutions

### 1. Get variable and value labels of s3hs

```
hsls %>%  
  select(s3hs) %>%  
  var_label()  
#> $s3hs  
#> [1] "S3 B01F Attending high school or homeschool as of Nov 1 2013"
```

```
hsls %>%  
  select(s3hs) %>%  
  val_labels()  
#> $s3hs  
#> Missing  
#> -9  
#> Unit non-response  
#> -8  
#> Item legitimate skip/NA  
#> -7  
#> Component not applicable  
#> -6  
#> Item not administered: abbreviated interview  
#> -4  
#> Yes  
#> 1  
#> No  
#> 2  
#> Don't know  
#> 3
```

## Labelled student exercise solutions

2. Get a count of the variable `s3hs` showing the value labels. **hint** use `factor()`

```
hsls %>%  
  count(s3hs)  
#> # A tibble: 6 x 2  
#>           s3hs      n  
#>           <dbl+lbl> <int>  
#> 1 -9 [Missing]      22  
#> 2 -8 [Unit non-response] 4945  
#> 3 -7 [Item legitimate skip/NA] 16770  
#> 4  1 [Yes]          624  
#> 5  2 [No]           985  
#> 6  3 [Don't know]    157
```

```
hsls %>%  
  count(s3hs) %>%  
  as_factor()  
#> # A tibble: 6 x 2  
#>   s3hs      n  
#>   <fct>    <int>  
#> 1 Missing      22  
#> 2 Unit non-response 4945  
#> 3 Item legitimate skip/NA 16770  
#> 4 Yes          624  
#> 5 No           985  
#> 6 Don't know    157
```

## Labelled student exercise solutions

### 3. Filter if value is associated with “Missing”

```
hsls %>%  
  filter(s3hs == -9) %>%  
  count()  
#> # A tibble: 1 x 1  
#>       n  
#>   <int>  
#> 1    22
```

## Labelled student exercise solutions

### 4. Filter if value is associated with “Missing” or “Unit non-response”

```
hsls %>%  
  filter(s3hs== -9 | s3hs== -8) %>%  
  count()  
#> # A tibble: 1 x 1  
#>       n  
#>   <int>  
#> 1  4967
```



Comparing labelled class to factor class

## Comparing `class==labelled` to `class==factor`

	<code>class==labelled</code>	<code>class==factor</code>
data type	numeric or character	integer
name of value label attribute	labels	levels
refer to data using	variable values	levels attribute

## Converting `class==labelled` to `class==factor`

The `as_factor()` function from `haven` package converts variables with `class==labelled` to `class==factor`

- ▶ Can be used for descriptive statistics

```
hsls %>% select(s3classes) %>% count(s3classes) %>% as_factor()
```

- ▶ Can create object with some or all `labelled` vars converted to `factor`

```
hsls_f <- as_factor(hsls, only_labelled = TRUE)
```

Let's examine this object

```
glimpse(hsls_f)
hsls_f %>% select(s3classes, s3clglvl1) %>% str()
typeof(hsls_f$s3classes)
class(hsls_f$s3classes)
attributes(hsls_f$s3classes)

hsls_f %>% select(s3classes) %>% var_label()
hsls_f %>% select(s3classes) %>% val_labels()
```

## Working with `class==factor` data

Showing values associated with factor levels

```
hsls_f %>% count(s3classes)
#> # A tibble: 5 x 2
#>   s3classes      n
#>   <fct>      <int>
#> 1 Missing      59
#> 2 Unit non-response 4945
#> 3 Yes        13477
#> 4 No         3401
#> 5 Don't know   1621
```

In code, refer `level` attribute not variable value

```
hsls_f %>% filter(s3classes=="Yes") %>% count(s3classes)
#> # A tibble: 1 x 2
#>   s3classes      n
#>   <fct>      <int>
#> 1 Yes        13477
```

## Creating factor variables

## Create factors [from string variables]

To create a factor variable from string variable

1. create a character vector containing underlying data
2. create a vector containing valid levels
3. Attach levels to the data using the `factor()` function

```
#underlying data: months my fam is born
x1 <- c("Jan", "Aug", "Apr", "Mar")
#create vector with valid levels
month_levels <- c("Jan", "Feb", "Mar", "Apr", "May", "Jun",
  "Jul", "Aug", "Sep", "Oct", "Nov", "Dec")
#attach levels to data
x2 <- factor(x1, levels = month_levels)
```

Note how attributes differ

```
str(x1)
#> chr [1:4] "Jan" "Aug" "Apr" "Mar"
str(x2)
#> Factor w/ 12 levels "Jan","Feb","Mar",...: 1 8 4 3
```

Sorting differs

```
sort(x1)
#> [1] "Apr" "Aug" "Jan" "Mar"
sort(x2)
#> [1] Jan Mar Apr Aug
#> Levels: Jan Feb Mar Apr May Jun Jul Aug Sep Oct Nov Dec
```

## Create factors [from string variables]

Let's create a character version of variable `hs_state` and then turn it into a factor

```
#wwlist %>%
# count(hs_state)
#Subset obs to West Coast states
wwlist_temp <- wwlist %>%
  filter(hs_state %in% c("CA", "OR", "WA"))

#Create character version of high school state for West Coast states only
wwlist_temp$hs_state_char <- as.character(wwlist_temp$hs_state)

#investigate character variable
str(wwlist_temp$hs_state_char)
table(wwlist_temp$hs_state_char)

#create new variable that assigns levels
wwlist_temp$hs_state_fac <- factor(wwlist_temp$hs_state_char, levels = c("CA", "OR", "WA"))
str(wwlist_temp$hs_state_fac)
attributes(wwlist_temp$hs_state_fac)

#wwlist_temp %>%
# count(hs_state_fac)
rm(wwlist_temp)
```

## Create factors [from string variables]

How the `levels` argument works when underlying data is character

- ▶ Matches value of underlying data to value of the level attribute
- ▶ Converts underlying data to integer, with level attribute attached

See chapter 15 of Wickham for more on factors (e.g., modifying factor order, modifying factor levels)



## Creating factors [from integer vectors]

Factors are just integer vectors with level attributes attached to them. So, to create a factor:

1. create a vector for the underlying data
2. create a vector that has level attributes
3. Attach levels to the data using the `factor()` function

```
a1 <- c(1,1,1,0,1,1,0) #a vector of data
a2 <- c("zero","one") #a vector of labels

#attach labels to values
a3 <- factor(a1, labels = a2)
a3
#> [1] one one one zero one one zero
#> Levels: zero one
str(a3)
#> Factor w/ 2 levels "zero","one": 2 2 2 1 2 2 1
```

Note: By default, `factor()` function attached “zero” to the lowest value of vector `a1` because “zero” was the first element of vector `a2`

## Creating factors [from integer vectors]

Let's turn an integer variable into a factor variable in the `wwlist` data frame

Create integer version of `receive_year`

```
#typeof(wwlist_temp$receive_year)
wwlist$receive_year_int <- as.integer(wwlist$receive_year)
str(wwlist$receive_year_int)
#> int [1:268396] 2016 2016 2016 2016 2016 2016 2016 2016 2016 2016 ...
typeof(wwlist$receive_year_int)
#> [1] "integer"
```

Assign levels to values of integer variable

```
wwlist$receive_year_fac <- factor(wwlist$receive_year_int,
  labels=c("Twenty-sixteen", "Twenty-seventeen", "Twenty-eighteen"))
str(wwlist$receive_year_fac)
str(wwlist$receive_year)

#Check variable
wwlist %>%
  count(receive_year_fac)

wwlist %>%
  count(receive_year)
```