Lecture 5: Augmented vectors and exploratory data analysis

EDUC 263: Managing and Manipulating Data Using R

Ozan Jaquette

1 Introduction

Logistics

Reading to do before next class:

- Work through slides from lecture 5 that we don't get to in class
- 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)

Explanation about `beamer_header.tex` in YAML header:

- We are calling the beamer_header.tex file in the background to customize our slides. Without this LaTeX file, our slides would compile according to the default beamer presentation (PDF).
  - Why would we want to do this?
  - We can customize our slides with the beamer_header.tex LaTeX file to include page numbers, change heading options, or change slide colors (in addition to other things).
- `includes` option in the YAML header customizes the beamer presentation slides
  - Here is a link to a short description of the includes option in the YAML header.

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. Exploratory data analysis (EDA)
   3.1 Tools for EDA
   3.2 Guidelines for EDA
   3.3 Skip patterns in survey data
4. Appendix. 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)
```

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)`

2 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_m
```

2.1 Review data types and structures

Vectors are the primary data structures in R

Two types of vectors:

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

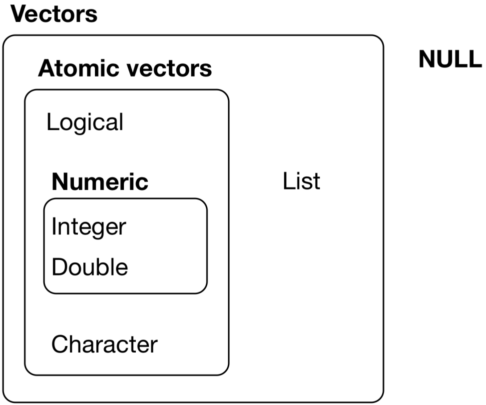

Figure 1: Overview of data structures (Grolemund and Wickham, 2018)

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
```

2.2 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 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"
```

2.3 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 with 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$receive_date[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

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

str(wwlist$ethn_code_fac)
#>  Factor w/ 10 levels "american indian or alaska native",..: 7 10 10 7 10 7 7
```

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

```r
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"

```r
# 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-2 or more"
#>  [8] "other spanish/hispanic"
#>  [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)
```

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 lub

#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
```

2.5 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 (HSLS: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 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
```

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(hsls$s3classes)

hsls %>% select(s3classes) %>% var_label
hsls %>% 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(hsls$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(hsls$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$variable_name,"labels")`
    - e.g., `attr(hsls$s3classes,"labels")`

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

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

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

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

filter specific values

```
hsls %>% filter(s3classes==-8) %>% count() # -8 = unit non-response
hsls %>% 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           22
#> 2 -8         4945
#> 3 -7        16770
#> 4 " 1"        624
#> 5 " 2"        985
#> 6 " 3"        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
```

2.6 Comparing labelled class to factor class

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

	`class==labelled`	`class==factor`
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,s3clglvl) %>% 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
```

3 Exploratory data analysis (EDA)

What is exploratory data analysis (EDA)?

The Towards Data Science website has a nice definition of EDA:

> *"Exploratory Data Analysis refers to the critical process of performing initial investigations on data so as to discover patterns,to spot anomalies,to test hypothesis and to check assumptions with the help of summary statistics"*

This course focuses on "data management":

- investigating and cleaning data for the purpose of creating analysis variables
- Basically, everything that happens **before** you conduct analyses

I think about "exploratory data analysis for data quality"

- Investigating values and patterns of variables from "input data"
- Identifying and cleaning errors or values that need to be changed
- Creating analysis variables
- Checking values of analysis variables agains values of input variables

How we will teach exploratory data analysis

Will teach exploratory data analysis (EDA) in two sub-sections:

1. Introduce "Tools of EDA":
   - Demonstrate code to investigate variables and relatioship between variables
   - I'll focus on the **tidyverse** approach rather than **base R**
   - Most of these tools are just the application of programming skills you have already learned
2. Provide "Guidelines for EDA"
   - Less about coding, more about practices you should follow and mentality necessary to ensure high data quality

3.1 Tools for EDA

Tools of EDA

To do EDA for data quality, must master the following tools:

- **Select, sort, filter, and print** in order to see data patterns, anomolies
  - Select and sort particular values of particular variables
  - Print particular values of particular variables
- **One-way descriptive analyses** (i.e,. focus on one variable)
  - Descriptive analyses for continuous variables
  - Descriptive analyses for discreet/categorical variables
- **Two-way descriptive analyses** (relationship between two variables)
  - Categorical by categorical
  - Categorical by continuous
  - Continuous by continuous

Whenever using any of these tools, **pay close attention to missing values and how they are coded**

- Often, the "input" variables don't code missing values as `NA`
- Especially when working with survey data, missing values coded as a negative number (e.g., `-9`, `-8`, `-4`) with different negative values representing different reasons for data being missing
- sometimes missing values coded as very high positive numbers
- Therefore, important to investigate input vars prior to creating analysis vars

Tools of EDA

First, Let's create a smaller version of the HSLS:09 dataset

```
#hsls %>% var_label()
hsls_small <- hsls %>%
  select(stu_id,x3univ1,x3sqstat,x4univ1,x4sqstat,s3classes,
         s3work,s3focus,s3clgft,s3workft,s3clgid,s3clgcntrl,
         s3clglvl,s3clgsel,s3clgstate,s3proglevel,x4evrappclg,
         x4evratndclg,x4atndclg16fb,x4ps1sector,x4ps1level,
         x4ps1ctrl,x4ps1select,x4refsector,x4reflevel,x4refctrl,
         x4refselect, x2sex,x2race,x2paredu,x2txmtscor,x4x2ses,x4x2sesq5)
```

```
names(hsls_small)
hsls_small %>% var_label()
```

Tools of EDA: select, sort, filter, and print

We've already know `select()`, `arrange()`, `filter()`

Select, sort, and print specific vars

```
#sort and print
hsls_small %>% arrange(desc(stu_id)) %>%
  select(stu_id,x3univ1,x3sqstat,s3classes,s3clglvl)

#investigate variable attributes
hsls_small %>% arrange(desc(stu_id)) %>%
  select(stu_id,x3univ1,x3sqstat,s3classes,s3clglvl) %>% str()

#print observations with value labels rather than variable values
hsls_small %>% arrange(desc(stu_id)) %>%
  select(stu_id,x3univ1,x3sqstat,s3classes,s3clglvl) %>% as_factor()
```

Sometimes helpful to increase the number of observations printed

```
class(hsls_small) #it's a tibble, which is the "tidyverse" version of a data fra
options(tibble.print_min=50)
# execute this in console
hsls_small %>% arrange(desc(stu_id)) %>%
  select(stu_id,x3univ1,x3sqstat,s3classes,s3clglvl)
options(tibble.print_min=10) # set default printing back to 10 lines
```

One-way descriptive stats for continuous vars, Base R approach [SKIP]

```
mean(hsls_small$x2txmtscor)
sd(hsls_small$x2txmtscor)

#Careful: summary stats include value of -8!
min(hsls_small$x2txmtscor)
max(hsls_small$x2txmtscor)
```

Be careful with `NA` values

```
#Create variable replacing -8 with NA
hsls_small_temp <- hsls_small %>%
  mutate(x2txmtscorv2=ifelse(x2txmtscor==-8,NA,x2txmtscor))
hsls_small_temp %>% filter(is.na(x2txmtscorv2)) %>% count(x2txmtscorv2)

mean(hsls_small_temp$x2txmtscorv2)
mean(hsls_small_temp$x2txmtscorv2, na.rm=TRUE)
rm(hsls_small_temp)
```

One-way descriptive stats for continuous vars, Tidyverse approach

Use `summarise_at()`, a variation of `summarise()`, to make descriptive stats

- explain `.args=list(na.rm=TRUE)` on following slides

Task:

- calculate descriptive stats for `x2txmtscor`, math test score

```
#?summarise_at
hsls_small %>% select(x2txmtscor) %>% var_label()
#> $x2txmtscor
#> [1] "X2 Mathematics standardized theta score"
hsls_small %>%
  summarise_at(
    .vars = vars(x2txmtscor),
    .funs = funs(mean, sd, min, max, .args=list(na.rm=TRUE))
  )
#> # A tibble: 1 x 4
#>    mean    sd   min   max
#>   <dbl> <dbl> <dbl> <dbl>
#> 1  44.1  21.8    -8  84.9
```

One-way descriptive stats for continuous vars, Tidyverse approach

Can calculate descriptive stats for more than one variable at a time

Task:

- calculate descriptive stats for `x2txmtscor`, math test score, and `x4x2ses`, socioeconomic index score

```
hsls_small %>% select(x2txmtscor,x4x2ses) %>% var_label()
#> $x2txmtscor
#> [1] "X2 Mathematics standardized theta score"
#>
#> $x4x2ses
#> [1] "X4 Revised X2 Socio-economic status composite"

hsls_small %>%
  summarise_at(
    .vars = vars(x2txmtscor,x4x2ses),
    .funs = funs(mean, sd, min, max, .args=list(na.rm=TRUE))
  )
#> # A tibble: 1 x 8
#>   x2txmtscor_mean x4x2ses_mean x2txmtscor_sd x4x2ses_sd x2txmtscor_min
#>             <dbl>        <dbl>         <dbl>      <dbl>          <dbl>
#> 1            44.1       -0.802          21.8       2.63             -8
#> # ... with 3 more variables: x4x2ses_min <dbl>, x2txmtscor_max <dbl>,
#> #   x4x2ses_max <dbl>
```

One-way descriptive stats for continuous vars, Tidyverse approach

"Input vars" in survey data often have negative values for missing/skips

```
hsls_small %>% filter(x2txmtscor<0) %>% count(x2txmtscor)
```

R includes those negative values when calculating stats; you don't want this

- Solution: create version of variable that replaces negative values with `NA`

```
hsls_small %>% mutate(x2txmtscor_na=ifelse(x2txmtscor<0,NA,x2txmtscor)) %>%
  summarise_at(
    .vars = vars(x2txmtscor_na),
    .funs = funs(mean, sd, min, max, .args=list(na.rm=TRUE))
  )
#> # A tibble: 1 x 4
#>    mean    sd   min   max
#>   <dbl> <dbl> <dbl> <dbl>
#> 1  51.5  10.2  22.2  84.9
```

What if you didn't include `.args=list(na.rm=TRUE)`?

```
hsls_small %>% mutate(x2txmtscor_na=ifelse(x2txmtscor<0,NA,x2txmtscor)) %>%
  summarise_at(
    .vars = vars(x2txmtscor_na),
    .funs = funs(mean, sd, min, max))
#> # A tibble: 1 x 4
#>    mean    sd   min   max
#>   <dbl> <dbl> <dbl> <dbl>
#> 1    NA   NaN    NA    NA
```

One-way descriptive stats for continuous vars, Tidyverse approach

How to identify these missing/skip values if you don't have a codebook?

- `count()` combined with `filter()` helpful for finding extreme values of continuous vars, which are often associated with missing or skip

```
#variable x2txmtscor
hsls_small %>% filter(x2txmtscor<0) %>%
  count(x2txmtscor)
#> # A tibble: 1 x 2
#>   x2txmtscor     n
#>        <dbl> <int>
#> 1         -8  2909

#variable s3clglvl
hsls_small %>% select(s3clglvl) %>% var_label()
#> $s3clglvl
#> [1] "S3 Enrolled college IPEDS level"

hsls_small %>% filter(s3clglvl<0) %>%
  count(s3clglvl)
#> # A tibble: 3 x 2
#>   s3clglvl      n
#>   <dbl+lbl> <int>
#> 1 -9          487
#> 2 -8         4945
#> 3 -7         5022
```

One-way descriptive stats student exercise

1. Using the object `hsls`, identify variable type, variable class, and check the variable vakyes and value labels of `x4ps1start`
   - variable `x4ps1start` identifies month and year student first started postsecondary education
   - **Note**: This variable is a bit counterintuitive.
     - e.g., the value `201105` refers to May 2011
2. Get a frequency count of the variable `x4ps1start`
3. Get a frequency count of the variable, but this time only observations that have negative values **hint**: use filter()
4. Create a new version of the variable `x4ps1start_na` that replaces negative values with NAs and use `summarise_at()` to get the min and max value.

One-way descriptive stats student exercise solutions

1. Using the object `hsls`, identify variable type, variable class, and check the variable vakyes and value labels of `x4ps1start`

```
typeof(hsls$x4ps1start)
#> [1] "double"
class(hsls$x4ps1start)
#> [1] "labelled"

hsls %>% select(x4ps1start) %>% var_label()
#> $x4ps1start
#> [1] "X4 Month and year of enrollment at first postsecondary institution"

hsls %>% select(x4ps1start) %>% val_labels()
#> $x4ps1start
#>                                       Missing
#>                                            -9
#>                             Unit non-response
#>                                            -8
#>                        Item legitimate skip/NA
#>                                            -7
#>                      Component not applicable
#>                                            -6
#> Item not administered: abbreviated interview
#>                                            -4
#>                         Carry through missing
#>                                            -3
#>                                    Don't know
```

One-way descriptive stats student exercise solutions

2. Get a frequency count of the variable `x4ps1start`

```
hsls %>%
  count(x4ps1start)
#> # A tibble: 9 x 2
#>   x4ps1start     n
#>   <dbl+lbl>  <int>
#> 1 "    -9"     107
#> 2 "    -8"    6168
#> 3 "    -7"    4281
#> 4 201100        57
#> 5 201200       206
#> 6 201300     10800
#> 7 201400      1295
#> 8 201500       471
#> 9 201600       118
```

One-way descriptive stats student exercise solutions

3. Get a frequency count of the variable, but this time only observations that have negative values **hint**: use filter()

```
hsls %>%
  filter(x4ps1start<0) %>%
  count(x4ps1start)
#> # A tibble: 3 x 2
#>   x4ps1start     n
#>   <dbl+lbl>  <int>
#> 1 -9           107
#> 2 -8          6168
#> 3 -7          4281
```

One-way descriptive stats student exercise solutions

4. Create a new version `x4ps1start_na` of the variable `x4ps1start` that replaces negative values with NAs and use `summarise_at()` to get the min and max value.

```
hsls %>% mutate(x4ps1start_na=ifelse(x4ps1start<0,NA,x4ps1start)) %>%
  summarise_at(
    .vars = vars(x4ps1start_na),
    .funs = funs(min, max, .args=list(na.rm=TRUE))
  )
#> # A tibble: 1 x 2
#>      min    max
#>    <dbl>  <dbl>
#> 1 201100 201600
```

One-way descriptive stats for discrete/categorical vars, Tidyverse approach

Use `count()` to investigate values of discreet or categorical variables

For variables where `class==labelled`

```
class(hsls_small$s3classes)
#show counts of variable values
hsls_small %>% count(s3classes)
#show counts of value labels
hsls_small %>% count(s3classes) %>% as_factor()
```

- I like `count()` because the default setting is to show `NA` values too!

```
hsls_small %>% mutate(s3classes_na=ifelse(s3classes<0,NA,s3classes)) %>%
  count(s3classes_na)
```

Simultaneously show both values and value labels on count tables for `class==labelled`

- requires some concepts/functions we haven't introduced

```
x <- hsls_small %>% count(s3classes)
y <- hsls_small %>% count(s3classes) %>% as_factor()
bind_cols(x[,1], y)
```

One-way descriptive stats for factor vars [OPTIONAL/SKIP]

For variables where `class==factor`

- Note: data frame object `hsls_f` created in previous section

```
#use variable from the hsls data frame where vars are factors
typeof(hsls_f$s3classes)
class(hsls_f$s3classes)
attributes(hsls_f$s3classes)

#show frequency table
hsls_f %>% count(s3classes)

#Create VAR that converts different types of missing to NA and then create frequ

#note: within ifelse() used levels(s3classes)[s3classes]) rather than s3classes
hsls_f %>% mutate(s3classes_f=ifelse(s3classes %in% c("Missing","Unit non-respon
  count(s3classes_f)
```

Relationship between variables, categorical by categorical

Two-way frequency table, called "cross tabulation", important for data quality

- When you create categorical analysis var from single categorical "input" var
  - Two-way tables show us whether we did this correctly
- Two-way tables helpful for understanding skip patterns in surveys

key to syntax

- `group_by(var1) %>% count(var2)`
- play around with which variable is `var1` and which variable is `var2`

Task:

- Create a two-way table between `s3classes` and `s3clglvl`

```
hsls_small %>% select(s3classes,s3clglvl) %>% var_label()

hsls_small %>% group_by(s3classes) %>% count(s3clglvl) # show values
hsls_small %>% group_by(s3classes) %>% count(s3clglvl) %>% as_factor() # show va
```

Relationship between variables, categorical by categorical

Two-way frequency table, also called "cross tabulation"

What if one of the variables has `NAs` ?

- Table created by `group_by()` and `count()` shows `NAs` !

Task:

- Create a version of `s3classes` called `s3classes_na` that changes negative values to `NA`
- Create a two-way table between `s3classes_na` and `s3clglvl`

```
hsls_small %>%
  mutate(s3classes_na=ifelse(s3classes<0,NA,s3classes)) %>%
  group_by(s3classes_na) %>% count(s3clglvl)

#example where we create some NA obs in the second variable
hsls_small %>%
  mutate(s3classes_na=ifelse(s3classes<0,NA,s3classes),
         s3clglvl_na=ifelse(s3clglvl==-7,NA,s3clglvl)) %>%
  group_by(s3classes_na) %>% count(s3clglvl_na)
```

Relationship between variables, categorical by categorical [SKIP]

Tables above are pretty ugly

Use the `spread()` function from `tidyr` package to create table with one variable as columns and the other variable as rows

- The variable you place in `spread()` will be columns
- We learn `spread()` function next week

```
hsls_small %>% group_by(s3classes) %>% count(s3clglvl) %>%
  spread(s3classes, n)

hsls_small %>% group_by(s3classes) %>% count(s3clglvl) %>%
  as_factor() %>%  spread(s3classes, n)
hsls_small %>% group_by(s3classes) %>% count(s3clglvl) %>%
  as_factor() %>%  spread(s3clglvl, n)
```

Relationship between variables, categorical by continuous

Investigating relationship between multiple variables is a little tougher when at least one of the variables is continuous

Conditional mean (like regression with continuous Y and one categorical X):

- Shows average values of continous variables within groups
- Groups are defined by your categorical variable(s)

key to syntax

- `group_by(categorical_var) %>% summarise_at(.vars = vars(continuous_var)`

Task

- Calculate mean math score, `x2txmtscor`, for each value of parental education, `x2paredu`

```
#first, investigate parental education
hsls_small %>% count(x2paredu)
hsls_small %>% count(x2paredu) %>% as_factor

# using dplyr to get average math score by parental education level
hsls_small %>% group_by(x2paredu) %>%
    summarise_at(.vars = vars(x2txmtscor),
                 .funs = funs(mean, .args = list(na.rm = TRUE))) %>%
    as_factor()
```

Relationship between variables, categorical by continuous

Task

- Calculate mean math score, `x2txmtscor`, for each value of `x2paredu`

For checking data quality, helpful to calculate other stats besides mean

```
hsls_small %>% group_by(x2paredu) %>%
    summarise_at(.vars = vars(x2txmtscor),
                 .funs = funs(mean, min, max, .args = list(na.rm = TRUE))) %>%
    as_factor()
```

Always Investigate presence of missing/skip values

```
hsls_small %>% filter(x2paredu<0) %>% count(x2paredu)
hsls_small %>% filter(x2txmtscor<0) %>% count(x2txmtscor)
```

Replace `-8` with `NA` and re-calculate conditional stats

```
hsls_small %>%
  mutate(x2paredu_na=ifelse(x2paredu<0,NA,x2paredu),
         x2txmtscor_na=ifelse(x2txmtscor<0,NA,x2txmtscor)) %>%
  group_by(x2paredu_na) %>%
  summarise_at(.vars = vars(x2txmtscor_na),
               .funs = funs(mean, min, max, .args = list(na.rm = TRUE))) %>%
  as_factor()
```

Student exercise

Can use same approach to calculate conditional mean by multiple `group_by()` variables

- Just add additional variables within `group_by()`

1. Calculate mean math test score (`x2txmtscor`), for each combination of parental education (`x2paredu`) and sex (`x2sex`).

Student exercise solution

1. Calculate mean math test score (`x2txmtscor`), for each combination of parental education (`x2paredu`) and sex (`x2sex`)

```
#hsls_small %>% count(x2sex)

hsls_small %>%
  group_by(x2paredu,x2sex) %>%
  summarise_at(.vars = vars(x2txmtscor),
               .funs = funs(mean, .args = list(na.rm = TRUE))) %>%
  as_factor()
```

3.2 Guidelines for EDA

Guidelines for "EDA for data quality"

Assme that your goal in "EDA for data quality" is to investigate "input" data sources and create "analysis variables"

- Usually, your analysis dataset will incorporate multiple sources of input data, including data you collect (primary data) and/or data collected by others (secondary data)

While this is not a linear process, these are the broad steps I follow

1. Understand how input data sources were created
   - e.g., when working with survey data, have survey questionnaire and codebooks on hand
2. For each input data source, identify the "unit of analysis" and which combination of variables uniquely identify observations
3. Investigate patterns in input variables
4. Create analysis variable from input variable(s)
5. Verify that analysis variable is created correctly through descriptive statistics that compare values of input variable(s) against values of the analysis variable

Always be aware of missing values

- They will not always be coded as `NA` in input variables

“Unit of analysis” and which variables uniquely identify observations

“Unit of analysis” refers to “what does each observation represent” in an input data source

- If each obs represents a student, you have “student level data”
- If each obs represents a student-course, you have “student-course level data”
- If each obs represents a school, you have “school-level data”
- If each obs represents a school-year, you have “school-year level data”

How to identify unit of analysis

- data documentation
- investigating the data set

We will go over syntax for identifying unit of analysis in subsequent weeks

Rules for variable creation

Rules I follow for variable creation

1. Never modify "input variable"; instead create new variable based on input variable(s)
   - Always keep input variables used to create new variables
2. Investigate input variable(s) and relationship between input variables
3. Developing a plan for creation of analysis variable
   - e.g., for each possible value of input variables, what should value of analysis variable be?
4. Write code to create analysis variable
5. Run descriptive checks to verify new variables are constructed correctly
   - Can "comment out" these checks, but don't delete them
6. Document new variables with notes and labels

Rules for variable creation

Task:

- Create analysis for variable ses qunitile called `sesq5` based on `x4x2sesq5` that converts negative values to `NAs`

```
#investigate input variable
hsls_small %>% select(x4x2sesq5) %>% var_label()
hsls_small %>% select(x4x2sesq5) %>% val_labels()
hsls_small %>% select(x4x2sesq5) %>% count(x4x2sesq5)
hsls_small %>% select(x4x2sesq5) %>% count(x4x2sesq5) %>% as_factor()

#create analysis variable
hsls_small_temp <- hsls_small %>%
  mutate(sesq5=ifelse(x4x2sesq5==-8,NA,x4x2sesq5)) # approach 1
hsls_small_temp <- hsls_small %>%
  mutate(sesq5=ifelse(x4x2sesq5<0,NA,x4x2sesq5)) # approach 2

#verify
hsls_small_temp %>% group_by(x4x2sesq5) %>% count(sesq5)
```

Overview of problem set due next week

Assignment:

- create GPA from postsecondary transcript student-course level data

Data source: National Longitudinal Study of 1972 (NLS72)

- Follows 12th graders from 1972
  - Base year: 1972
  - Follow-up surveys in: 1973, 1974, 1976, 1979, 1986
  - Postsecondary transcripts collected in 1984

Why use such an old survey for this assignment?

- NLS72 predates data privacy agreements; transcript data publicly available

What we do to make assignment more manageable

- we give you code for investigation of variables
- we give you some hints/guidelines
- but you are responsible for developing plan to create GPA vars and for executing plan (rather than us giving you step-by-step quations)

Why this assignment?

1. Give you more practice investigating data, cleaning data, creating variables that require processing across rows
2. Sometimes social justice is creating student GPA from course-level data

3.3 Skip patterns in survey data

What are skip patterns

Pretty easy to create an analysis variable based on a single input variable

Harder to create analysis variables based on multiple input variables

- When working with survey data, even seemingly simple analysis variables require multiple input variables due to "skip patterns"

What are "skip patterns"?

- Response on a particular survey item determines whether respondent answers some set of subsequent questions
- What are some examples of this?

Key to working with skip patterns

- Have the survey questionnaire on hand
- Sometimes it appears that analysis variable requires only one input variable, but really depends on several input variables because of skip patterns
  - Don't just blindly turn "missing" and "skips" from survey data to `NAs` in your analysis variable
  - Rather, trace why these "missing" and "skips" appear and decide how they should be coded in your analysis variable

Creating analysis variables in the presence of skip patterns

Task: Create a measure of "level" of postsecondary institution attended in 2013 from HSLS:09 survey data

- "level" is highest award-level of the postsecondary institution
  - e.g., if highest award is associate's degree (a two-year degree), then 'level==2'
- The measure, `pselev2013`, should have following [non-missing] values:
  1. Not attending postsecondary education institution
  2. Attending a 2-year or less-than-2-year institution
  3. Attending 4-year or greater-than-4year institution

Background info:

- In "2013 Update" of HSLS:09, students asked about college attendance
  - Variables from student responses to "2013 Update" have prefix `s3`
- Survey questionnaire for 2013 update can be found HERE
- The "online codebook" website HERE has info about specific variables
- Measure has 3 input variables [usually must figure this out yourself]:
  1. `x3sqstat`: "X3 Student questionnaire status"
  2. `s3classes`: "S3 B01A Taking postsecondary classes as of Nov 1 2013"
  3. `s3clglvl`: "S3 Enrolled college IPEDS level"

```
hsls_small %>% select(x3sqstat,s3classes,s3clglvl) %>% var_label()
```

You won't have time to complete this task, but develop a plan for the task and get as far as you can

Creating analysis variables in the presence of skip patterns

Step 1a: Investigate each input variable separately

```
#variable labels
hsls_small %>% select(x3sqstat,s3classes,s3clglvl) %>% var_label()

hsls_small %>% count(x3sqstat)
hsls_small %>% count(x3sqstat) %>% as_factor()

hsls_small %>% count(s3classes)
hsls_small %>% count(s3classes) %>% as_factor()

hsls_small %>% count(s3clglvl)
hsls_small %>% count(s3clglvl) %>% as_factor()
```

Creating analysis variables in the presence of skip patterns

Step 1b: Investigate relationship between input variables

```
#x3sqstate and s3classes
hsls_small %>% group_by(x3sqstat) %>% count(s3classes)
hsls_small %>% group_by(x3sqstat) %>% count(s3classes) %>% as_factor()

hsls_small %>% filter(x3sqstat==8) %>% count(s3classes)
hsls_small %>% filter(x3sqstat==8) %>% count(s3classes==-8)
hsls_small %>% filter(x3sqstat !=8) %>% count(s3classes)

#x3sqstate, s3classes and s3clglvl
hsls_small %>% group_by(s3classes) %>% count(s3clglvl)
hsls_small %>% group_by(s3classes) %>% count(s3clglvl) %>% as_factor()

#add filter for whether student did not respond to X3 questionnaire
hsls_small %>% filter(x3sqstat==8) %>% group_by(s3classes) %>% count(s3clglvl)
hsls_small %>% filter(x3sqstat !=8) %>% group_by(s3classes) %>% count(s3clglvl)

#continued on the next page
```

Creating analysis variables in the presence of skip patterns

Step 1b: Investigate relationship between input variables continued...

```
#add filter for s3classes is "missing" [-9]
hsls_small %>% filter(x3sqstat !=8,s3classes==-9) %>% group_by(s3classes) %>%
  count(s3clglvl)
#> # A tibble: 1 x 3
#> # Groups:   s3classes [1]
#>   s3classes s3clglvl      n
#>   <dbl+lbl> <dbl+lbl> <int>
#> 1 -9        -9           59
hsls_small %>% filter(x3sqstat !=8,s3classes!=-9) %>% group_by(s3classes) %>%
  count(s3clglvl)
#> # A tibble: 6 x 3
#> # Groups:   s3classes [3]
#>   s3classes s3clglvl      n
#>   <dbl+lbl> <dbl+lbl> <int>
#> 1 1         -9          428
#> 2 1         " 1"       8894
#> 3 1         " 2"       3929
#> 4 1         " 3"        226
#> 5 2         -7         3401
#> 6 3         -7         1621

#add filter for s3classes equal to "no" or "don't know"
hsls_small %>% filter(x3sqstat !=8,s3classes!=-9, s3classes %in% c(2,3)) %>%
  group_by(s3classes) %>% count(s3clglvl)
#> # A tibble: 2 x 3
```

4 Appendix. 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","O
str(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-sixt
str(wwlist$receive_year_fac)
str(wwlist$receive_year)

#Check variable
wwlist %>%
  count(receive_year_fac)

wwlist %>%
  count(receive_year)
```