Preparing data for analysis and visualization in R

Understanding and changing data types

Jenine Harris
Brown School

Understanding and changing data types

- Objects are interpreted by R as one of several **data types**.
- Create `kStates` as the number of states with legal medical marijuana.

```
# make the kStates object
kStates <- 29
```

- To see what data type `kStates` is, use the `class()` function, like this:

```
# identify data type for kStates object
class(x = kStates)
```

```
## [1] "numeric"
```

- In this code, `class()` is the function, `kStates` is the argument, and `x` is the argument name

Numeric data type

- For the `kStates` object, R prints the data type **numeric** from the `class()` function.
- The numeric data type is the default that R assigns to constants and variables that contain only numbers.
- The numeric data type can be whole numbers or numbers with decimal places, so it is the most appropriate data type for variables measured along a continuum, or **continuous** variables.
    - For example, height and temperature can be both measured along a continuum and would usually be numeric data type in R.

Using class to understand data type

As an example, create a constant that contains the ounces of medical marijuana legally available to purchase per person in Rhode Island, then used `class()` to identify the data type.

```
# assign Rhode Island limit for medical marijuana
# in ounces per person
kOuncesRhode <- 2.5

# identify the data type for ouncesRhode
class(x = kOuncesRhode)
```

```
## [1] "numeric"
```

Integer data type

- The **integer** data type is similar to numeric but only contains whole numbers.
- There are true integers that can only be measured in whole numbers, like the number of cars parked in a lot.
- There are also things that could be numeric but are measured as integers like measuring age as *age in years*.
- When a whole number is assigned to a variable name in R the default type is numeric.

Using as.integer() to assign integer data type

- To change the variable type to integer, use the R function `as.integer()`.
- The `as.integer()` function can also be used to truncate numbers with decimal places.
- Note that **truncation** is not the same as rounding!
    - Truncation cuts off everything after the decimal place.
    - For example, truncating the value 8.9 would leave 8.
    - Rounding goes up or down to the nearest number, so 8.9 would round to 9.

Explore the integer data type

```
# assign the value of 4 to a constant called kTestInteger
# make sure it is an integer
kTestInteger <- as.integer(x = 4)
```

```
# use class() to determine the data type of kTestInteger
class(x = kTestInteger)
```

```
## [1] "integer"
```

```
# use as.integer() to truncate the object kOuncesRhode
as.integer(x = kOuncesRhode)
```

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

```
# multiply the kTestInteger and kOuncesRhode objects
kTestInteger * kOuncesRhode
```

```
## [1] 10
```

Working with integers

```
# multiply kTestInteger and integer kOuncesRhode
kTestInteger * as.integer(x = kOuncesRhode)
```

```
## [1] 8
```

```
# type the object name to see what is currently saved
# in the object
kOuncesRhode
```

```
## [1] 2.5
```

Logical data type

The **logical** data type includes the values of `TRUE` and `FALSE`. The values of `TRUE` and `FALSE` can be assigned to a logical constant, like this:

```
# create the object
kTestLogical <- TRUE

# print the value of the object
kTestLogical
```

```
## [1] TRUE
```

```
# check the object type
class(x = kTestLogical)
```

```
## [1] "logical"
```

Create logical constant

Logical constants can also be created as the result of some expression, such as:

```
# store the result of 6 > 8 in a constant called kSixEight
kSixEight <- 6 > 8

# print kSixEight
kSixEight
```

```
## [1] FALSE
```

```
# determine the data type of kSixEight
class(x = kSixEight)
```

```
## [1] "logical"
```

Because six is not greater than eight, the expression `6 > 8` is `FALSE`, which is assigned to the `kSixEight` object.

Character data type

- The **character** data type includes letters, words, or numbers that cannot logically be included in calculations (e.g., a zip code).
- They are always wrapped in either single or double quotation marks (e.g., `'hello'` or `"world"`).

```
# make character constants
kFirstName <- "Corina"
kLastName <- "Hughes"

# check the data type
class(x = kFirstName)
```

```
## [1] "character"
```

```
# create a zip code constant
# check the data type
kZipCode <- "97405"
class(x = kZipCode)
```

```
## [1] "character"
```

- Zip codes might look like an integer, but it is better to save as a character data type since doing math on a zip code would be trouble!

Factor data type

- The **factor** data type is used for variables that are measured in categories.
    - Variables measured in categories are **categorical**.
- Examples of categorical variables can include variables like:
    - religion
    - marital status
    - age group
- There are two types of categorical variables: **ordinal** and **nominal**.
- Ordinal variables contain categories that have some logical order.
    - For example, categories of age can logically be put in order from younger to older: 18--25, 26--39, 40--59, 60+.
- Nominal variables have categories that have no logical order.
    - Religious affiliation and marital status are examples of nominal variable types because there is no logical order to these characteristics (e.g., Methodist is not inherently greater or less than Catholic).

Check your understanding

Create the `kIllegalNum` as a constant representing the number of states where medical marijuana was illegal as of 2017 (n = 21). Check the data type for the `kIllegalNum` constant.

Answer:

```
# create the object
kIllegalNum <- 21

# check the data type
class(x = kIllegalNum)
```

```
## [1] "numeric"
```