Subsetting

Learning Objectives

- How to subset atomic vectors and lists.
- Chapters 6 and 7 of HOPR
- Chapter 4 from Advanced R
  - These lecture notes are mostly taken straight out of Hadley's book. Many thanks for making my life easier.
- **Subsetting** is extracting elements from an object.
  - Subset because you only want some elements of a vector.
  - Subset so you can assign new elements to that subset.
- Six ways to subset atomic vector.

```
x <- c(8, 1.2, 33, 14)
```

1. **Integer Subsetting**:

- Put integers in brackets and it will extract those elements. R starts counting at 1.

```
x[1]
```

```
[1] 8
```

```
x[c(1, 3)]
```

```
[1]  8 33
```

```r
iset <- c(1, 3)
x[iset]
```

```
[1]  8 33
```

- This can be used for sorting

```r
order(x)
```

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

```r
x[order(x)]
```

```
[1]  1.2  8.0 14.0 33.0
```

- You can use duplicate integers to extract elements more than once.

```r
x[c(2, 2, 2)]
```

```
[1] 1.2 1.2 1.2
```

2. **Negative Integer Subsetting**:

- Putting negative integers in instead will return all elements except the negative elements.

```r
x[-1]
```

```
[1]  1.2 33.0 14.0
```

```r
x[c(-1, -3)]
```

```
[1]  1.2 14.0
```

```r
x[-c(1, 3)]
```

```
[1]  1.2 14.0
```

3. **Logical Vector Subsetting**:

- Wherever there is a `TRUE` will return the element.

```r
x[c(TRUE, FALSE, TRUE, FALSE)]
```

```
[1]  8 33
```

4. **No Subsetting**:

- Empty brackets will return the original object.

```r
x[]
```

```
[1]  8.0  1.2 33.0 14.0
```

5. **Zero Subsetting**:

- Using `0` in a bracket will return a zero-length vector.

```r
x[0]
```

```
numeric(0)
```

6. **Names Subsetting**:

- If a vector has names, then you can subset using those names in quotes.

```r
names(x) <- c("a", "b", "c", "d")
x["a"]
```

```
a 
8 
```

```r
x[c("a", "c")]
```

```
 a  c 
 8 33 
```

```r
x[c("a", "a")]
```

```
a a 
8 8 
```

- If you know what names you want to remove, use `setdiff()`.

```
setdiff(names(x), "a")
```

```
[1] "b" "c" "d"
```

```
x[setdiff(names(x), "a")]
```

```
   b    c    d 
 1.2 33.0 14.0 
```

Exercises:

- **Exercise**: Explain the output of the following

```
y <- 1:9
y[c(TRUE, TRUE, FALSE)]
```

```
[1] 1 2 4 5 7 8
```

```
y[TRUE]
```

```
[1] 1 2 3 4 5 6 7 8 9
```

```
y[FALSE]
```

```
integer(0)
```

- **Exercise**: Explain the output of the following

```
y <- c(1, 2)
y[c(TRUE, TRUE, FALSE, TRUE, TRUE, FALSE)]
```

```
[1]  1  2 NA NA
```

- **Exercise**: Show all the ways to extract the second element of the following vector:

```r
y <- c(af = 3, bd = 6, dd = 2)
```

- Double brackets enforces that you are only extracting one element. This is really good in places where you know that you should only subset one element (like for-loops).

```r
x <- runif(100)
sval <- 0
for (i in seq_along(x)) {
  sval <- sval + x[[i]]
}
```

- Double brackets remove attributes of the vector (even names).

```r
x <- c(a = 1, b = 2)
x[1]
```

```
a
1
```

```r
x[[1]]
```

```
[1] 1
```

List subsetting

- If you subset a list using single brackets, you will get a sublist. You can use integers, negative integers, logicals, and names as before

```r
x <- list(a = 1:3, b = "hello", c = 4:6)
str(x)
```

```
List of 3
 $ a: int [1:3] 1 2 3
 $ b: chr "hello"
 $ c: int [1:3] 4 5 6
```

```r
x[1]
```

```
$a
[1] 1 2 3
```

```
x[c(1, 3)]
```

```
$a
[1] 1 2 3

$c
[1] 4 5 6
```

```
x[-1]
```

```
$b
[1] "hello"

$c
[1] 4 5 6
```

```
x[c(TRUE, FALSE, FALSE)]
```

```
$a
[1] 1 2 3
```

```
x["a"]
```

```
$a
[1] 1 2 3
```

```
x[c("a", "c")]
```

```
$a
[1] 1 2 3

$c
[1] 4 5 6
```

- Using double brackets extracts out a single element.

```
x[[1]]
```

```
[1] 1 2 3
```

```
x[["a"]]
```

```
[1] 1 2 3
```

- A shorthand for using names inside double brackets is to use dollar signs.

```
x$a
```

```
[1] 1 2 3
```

- **Exericse**: Why does this not work. Suggest a correction.

```
var <- "a"
x$var
```

```
NULL
```

Data Frame Subsetting

- Data frame subsetting behaves both like lists and like matrices.

```
df <- data.frame(a = 1:3,
                 b = c("a", "b", "c"),
                 c = 4:6)
```

- It behaves like a list for `$`, `[[`, and `[` if you only provide one index. The columns are the elements of the list.

```
df$a
```

```
[1] 1 2 3
```

```r
df[1]
```

```
  a
1 1
2 2
3 3
```

```r
df[[1]]
```

```
[1] 1 2 3
```

```r
df[c(1, 3)]
```

```
  a c
1 1 4
2 2 5
3 3 6
```

- It behaves like a matrix if you provide two indices.

```r
df[1:2, 2]
```

```
[1] "a" "b"
```

- You can keep the data frame structure by using `drop = FALSE`.

```r
df[1:2, 2, drop = FALSE]
```

```
  b
1 a
2 b
```

- It is common to filter by rows by using the matrix indexing.

```r
df[df$a < 3, ]
```

```
  a b c
1 1 a 4
2 2 b 5
```

Hadley's Advanced R Exercises

1. Fix each of the following common data frame subsetting errors:

```r
mtcars[mtcars$cyl = 4, ]
mtcars[-1:4, ]
mtcars[mtcars$cyl <= 5]
mtcars[mtcars$cyl == 4 | 6, ]
```

2. Why does the following code yield five missing values? (Hint: why is it different from `x[NA_real_]`?)

```r
x <- 1:5
x[NA]
```

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

3. What does `upper.tri()` return? How does subsetting a matrix with it work?

```r
x <- outer(1:5, 1:5, FUN = "*")
x[upper.tri(x)]
```

```
 [1]  2  3  6  4  8 12  5 10 15 20
```

4. Why does `mtcars[1:20]` return an error? How does it differ from the similar `mtcars[1:20, ]`?

5. An `lm` object is a list-like object. Given a linear model, e.g., `mod <- lm(mpg ~ wt, data = mtcars)`, extract the residual degrees of freedom. Then extract the R squared from the model summary (`summary(mod)`).

Subassignment

- All subsetting operators can be used to assign subsets of a vector new values. This is called **subassignment**.

```r
x <- 1:5
x[[2]] <- 200
x
```

```
[1]   1 200   3   4   5
```

```r
x[c(1, 3)] <- 0
x
```

```
[1]   0 200   0   4   5
```

```r
x[x == 0] <- NA_real_
x
```

```
[1]  NA 200  NA   4   5
```

```r
y <- list(a = 1:3,
          b = "hello",
          c = 4:6)
y$a <- "no way"
y
```

```
$a
[1] "no way"

$b
[1] "hello"

$c
[1] 4 5 6
```

- Remove a list element with `NULL`.

```r
y[[1]] <- NULL
y
```

```
$b
[1] "hello"

$c
[1] 4 5 6
```

```r
    y$b <- NULL
    y
```

```
$c
[1] 4 5 6
```

Exercises

These are just meant to buff up your Base R skills. Consider the data from the `{Sleuth3}` package that contains information on sex and salary at a bank. Try to use just base R methods.

```r
library(Sleuth3)
data("case0102")
sal <- case0102
```

1. What is the salary of the person in the 51st row? Use two different subsetting strategies to get this.

2. What is the mean salary of Male's?

3. How many Females are in the data?

4. How many Females make over $6000?