Data Frames

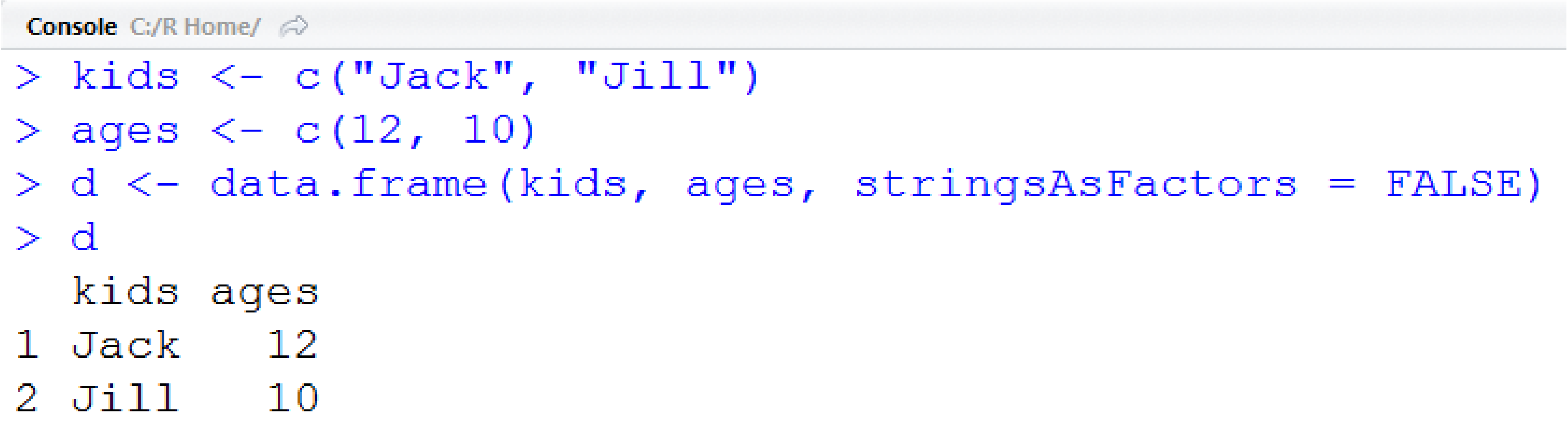
Bok, Jong Soon
javaexpert@nate.com
www.javaexpert.info

Data Frames

- On an intuitive level, a data frame is like a matrix, with a two-dimensional rows-and-columns structure.
- However, it differs from a matrix in that each column may have a different mode.
- For instance, one column may consist of numbers, and another column might have character strings.
- In this sense, just as lists are the heterogeneous analogs of vectors in one dimension, data frames are the heterogeneous analogs of matrices for two-dimensional data.

Creating Data Frames

- **Let's begin.**

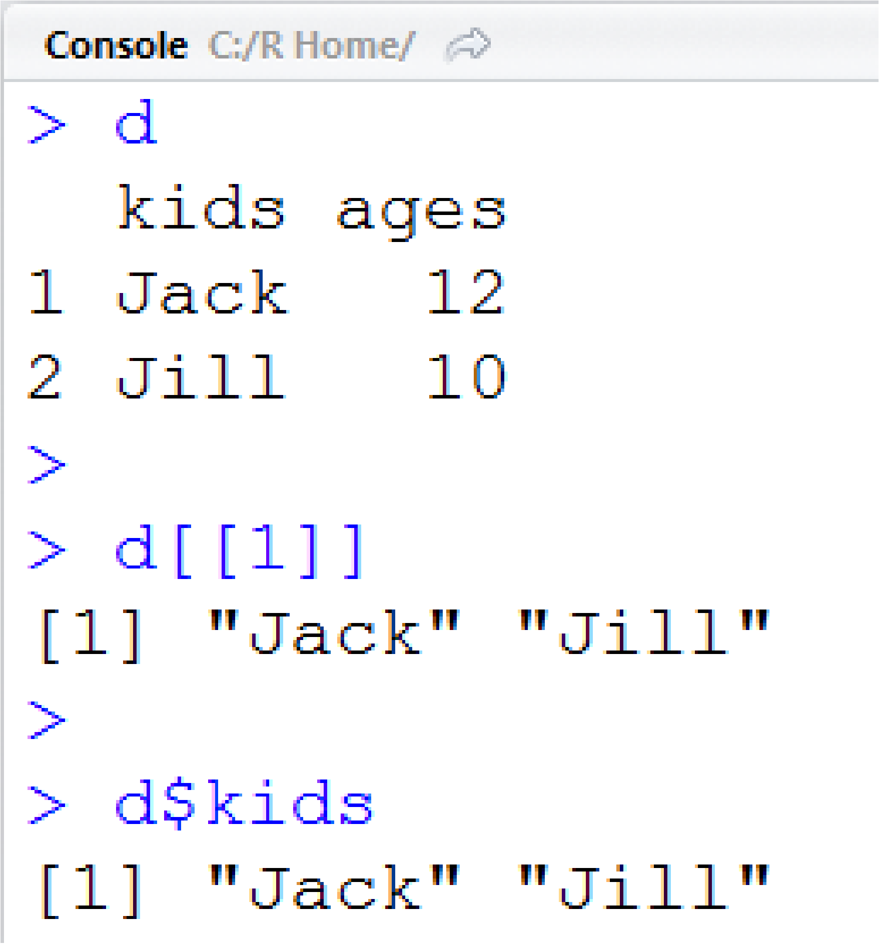


```
> kids <- c("Jack", "Jill")
> ages <- c(12, 10)
> d <- data.frame(kids, ages, stringsAsFactors = FALSE)
> d
  kids ages
1 Jack   12
2 Jill   10
```

Creating Data Frames (Cont.)

- The first two arguments in the call to `data.frame()` are clear.
- Produce a data frame from our two vectors: kids and ages.
- If the named argument `stringsAsFactors` is not specified, then by default, `stringsAsFactors` will be **TRUE**.
- This means that if we create a data frame from a character vector, kids → R will convert that vector to a factor.
- Because our work with character data will typically be with vectors rather than factors, we'll set `stringsAsFactors` to **FALSE**.

Accessing Data Frames

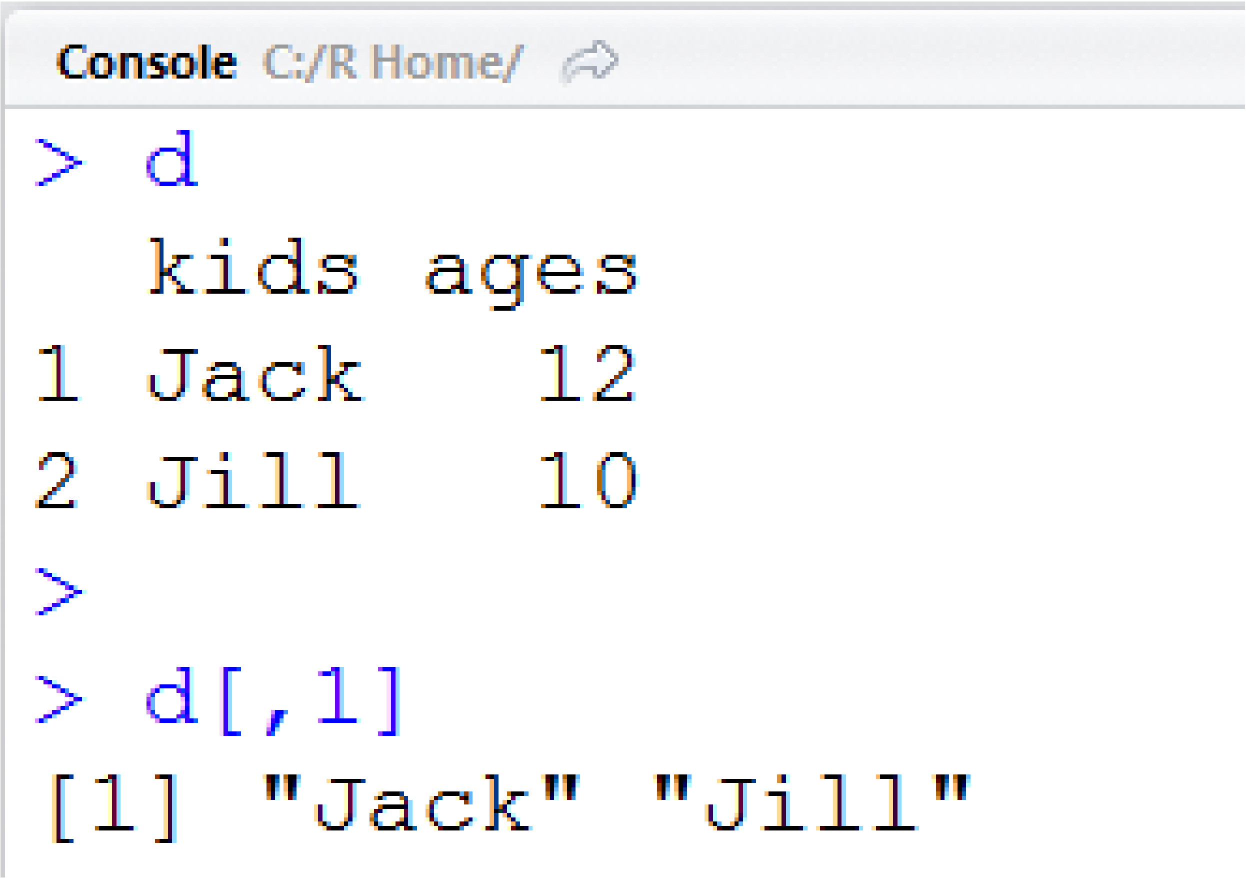


```
> d
  kids ages
1 Jack   12
2 Jill   10
>
> d[[1]]
[1] "Jack" "Jill"
>
> d$kids
[1] "Jack" "Jill"
```

- Since `d` is a list, can access it as such via component index values or component names.

Accessing Data Frames (Cont.)

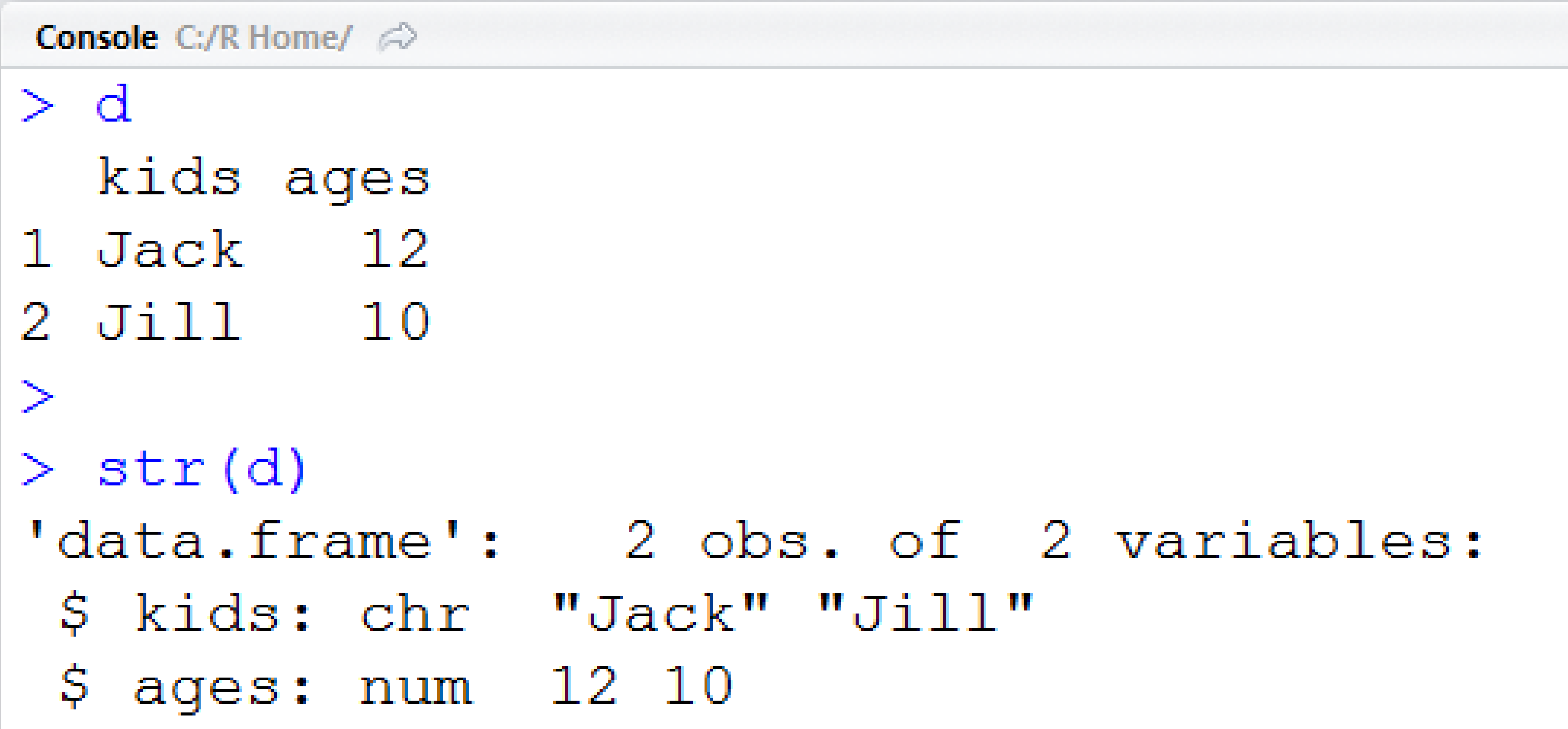
- But Can treat it in a matrix-like fashion as well.
- For example, can view column 1.



```
> d
  kids ages
1 Jack   12
2 Jill   10
>
> d[,1]
[1] "Jack" "Jill"
```

Accessing Data Frames (Cont.)

- This matrix-like quality is also seen when take **d** apart using **str()**.



```
> d
  kids ages
1 Jack   12
2 Jill   10
>
> str(d)
'data.frame':   2 obs. of  2 variables:
 $ kids: chr  "Jack" "Jill"
 $ ages: num  12 10
```

Accessing Data Frames (Cont.)

- Consider three ways to access the first column of our data frame above
  - `d[[1]]`
  - `d[,1]`
  - `d$kids`
- Of these, the third would generally considered to be clearer and, more importantly, safer than the first two.
- This better identifies the column and makes it less likely that you will reference the wrong column.

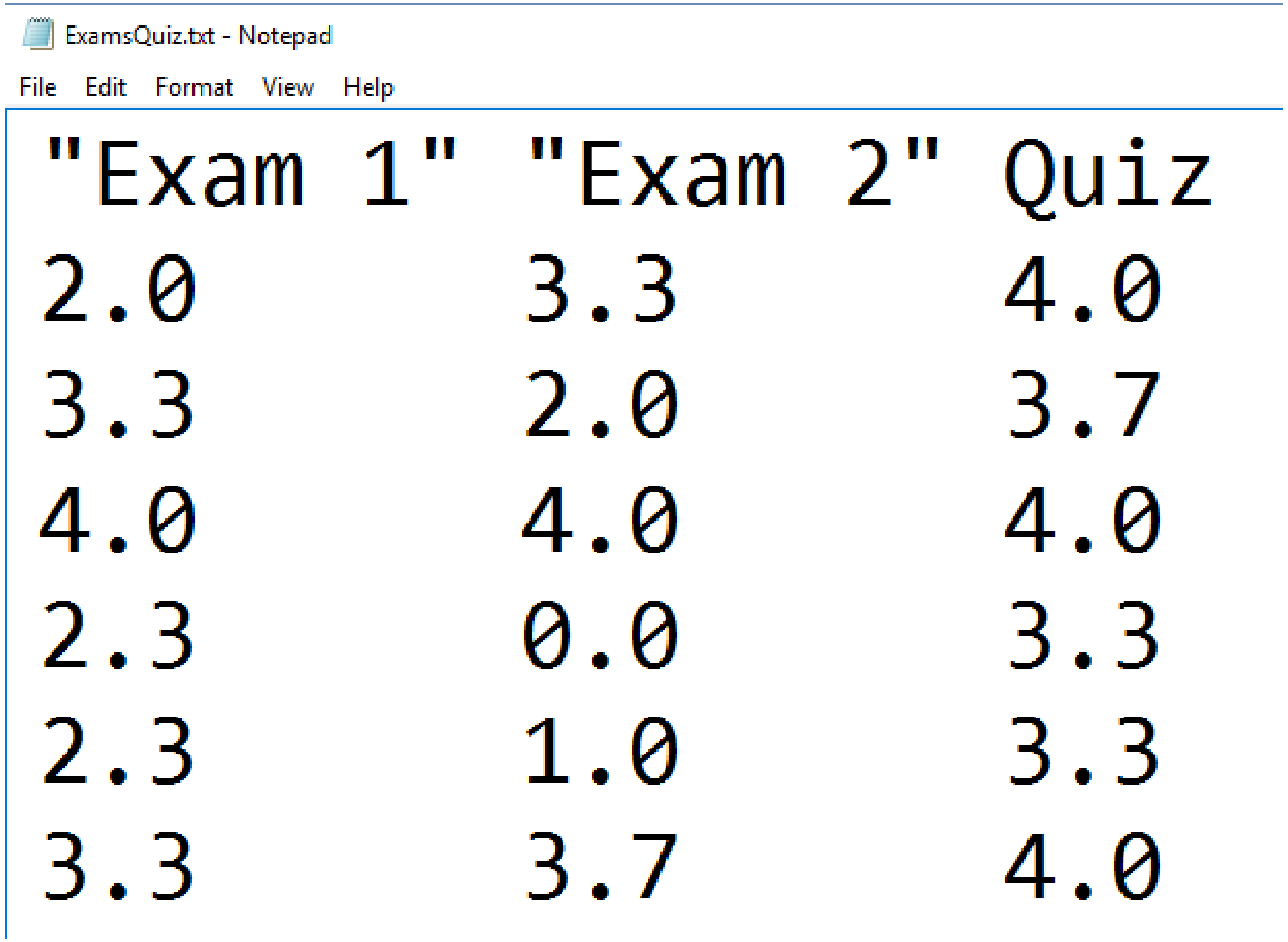
Other Matrix-Like Operations

- Various matrix operations also apply to data frames.
- Most notably and usefully, can do filtering to extract various subdata frames of interest.

Extracting Subdata Frames

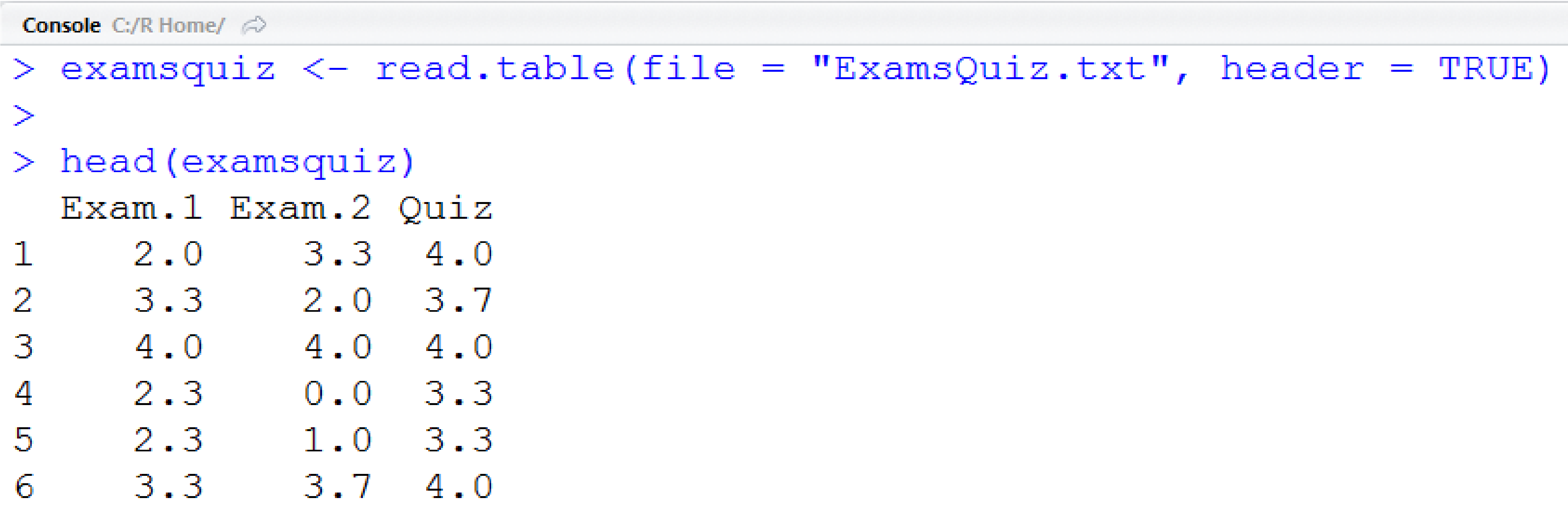
- A data frame can be viewed in row-and-column terms.
- In particular, we can extract subdata frames by rows or columns.

Extracting Subdata Frames (Cont.)



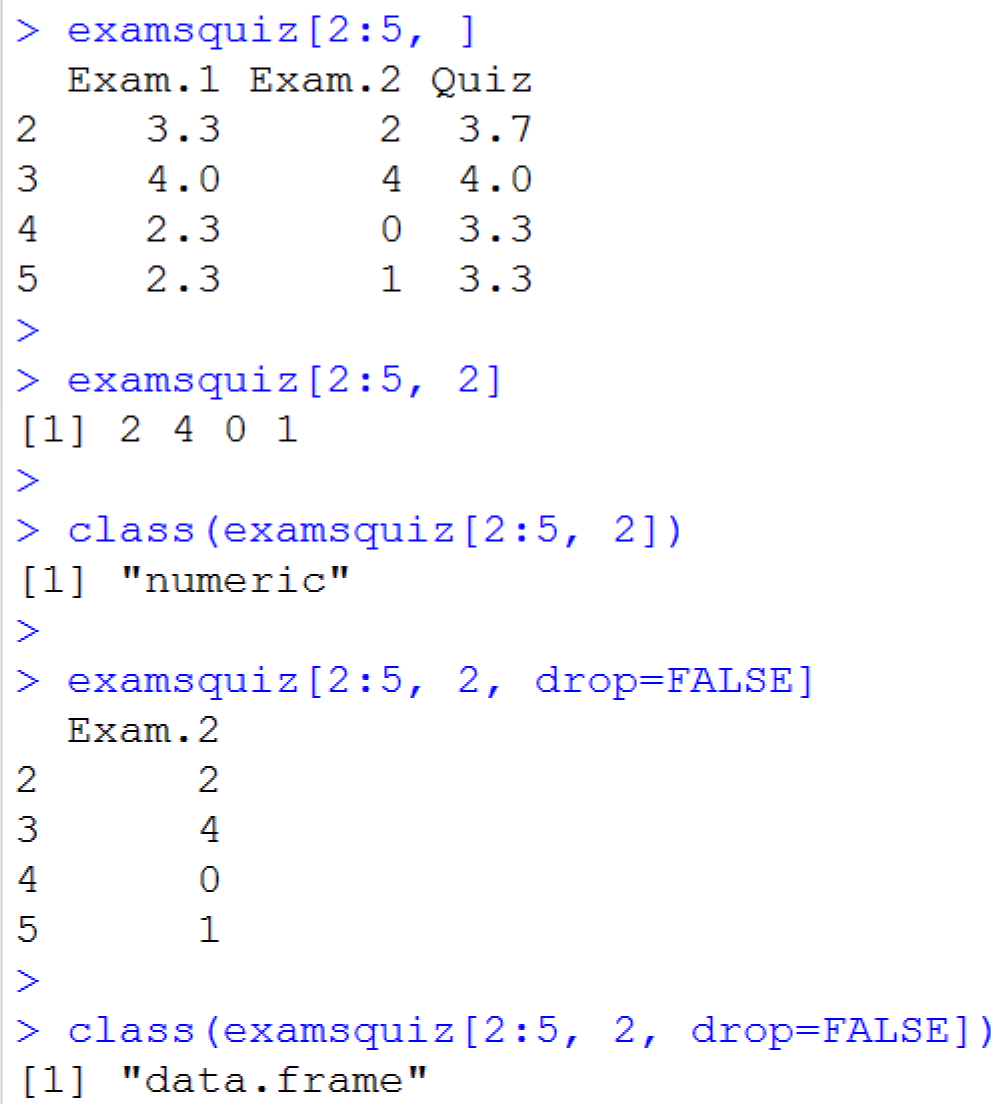
```
"Exam 1" "Exam 2" Quiz
2.0      3.3      4.0
3.3      2.0      3.7
4.0      4.0      4.0
2.3      0.0      3.3
2.3      1.0      3.3
3.3      3.7      4.0
```

Extracting Subdata Frames (Cont.)



```
> examsquiz <- read.table(file = "ExamsQuiz.txt", header = TRUE)
>
> head(examsquiz)
  Exam.1 Exam.2 Quiz
1    2.0    3.3  4.0
2    3.3    2.0  3.7
3    4.0    4.0  4.0
4    2.3    0.0  3.3
5    2.3    1.0  3.3
6    3.3    3.7  4.0
```

Extracting Subdata Frames (Cont.)

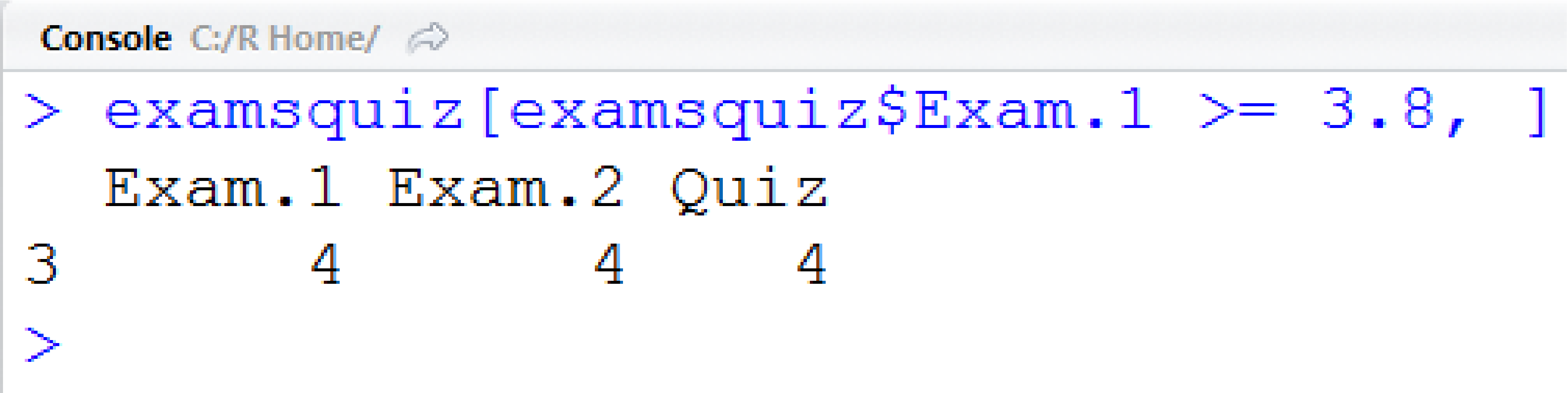


```
> examsquiz[2:5, ]
  Exam.1 Exam.2 Quiz
2    3.3      2  3.7
3    4.0      4  4.0
4    2.3      0  3.3
5    2.3      1  3.3
>
> examsquiz[2:5, 2]
[1] 2 4 0 1
>
> class(examsquiz[2:5, 2])
[1] "numeric"
>
> examsquiz[2:5, 2, drop=FALSE]
  Exam.2
2      2
3      4
4      0
5      1
>
> class(examsquiz[2:5, 2, drop=FALSE])
[1] "data.frame"
```

- Since `examsquiz[2:5,2]` is a vector, R created a vector instead of another data frame.
- By specifying `drop=FALSE`, can keep it as a (one-column) data frame.

Extracting Subdata Frames (Cont.)

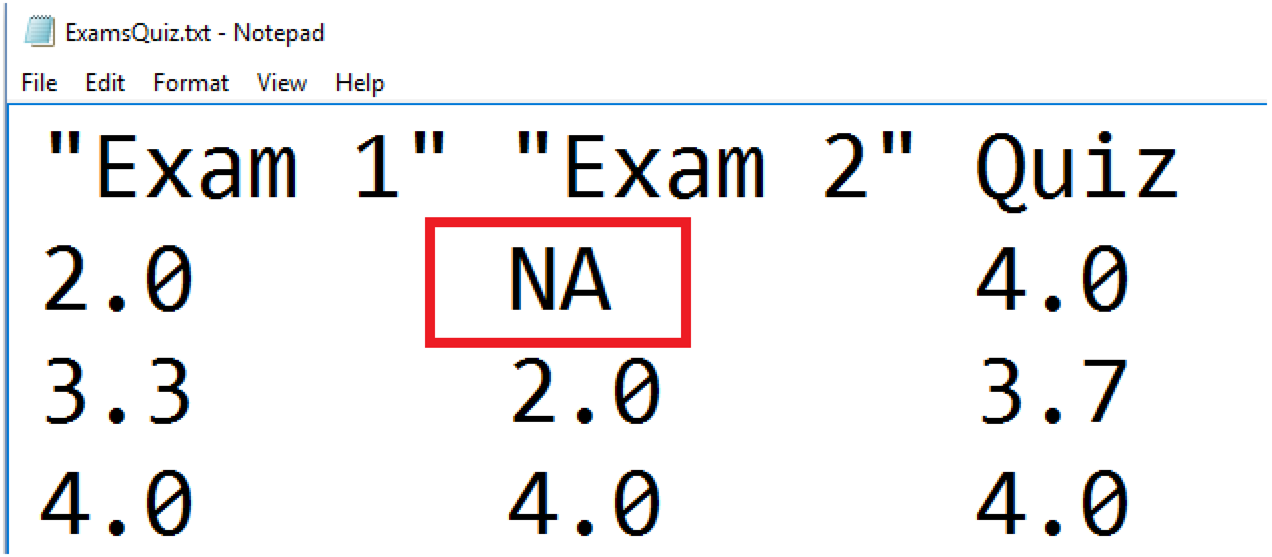
- We can also do filtering. Here’s how to extract the subframe of all students whose first exam score was at least 3.8.



```
> examsquiz[examsquiz$Exam.1 >= 3.8, ]
  Exam.1 Exam.2 Quiz
3      4      4    4
>
```

More on Treatment of NA Values

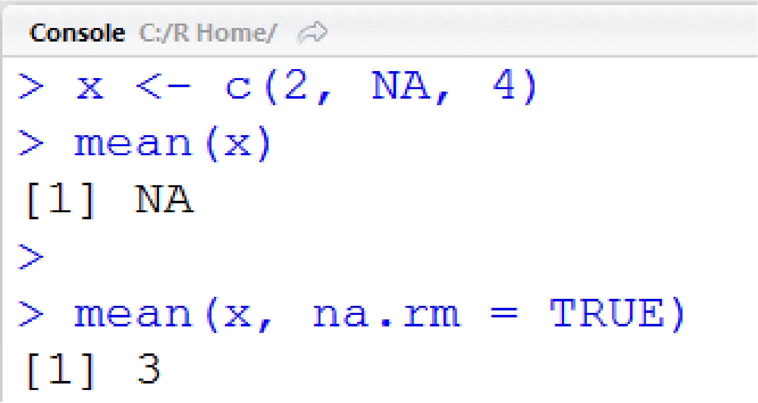
- Suppose the second exam score for the first student had been missing.
- Then we would have typed the following into that line when we were preparing the data file.



```
"Exam 1" "Exam 2" Quiz
2.0        NA      4.0
3.3        2.0     3.7
4.0        4.0     4.0
```

More on Treatment of NA Values (Cont.)

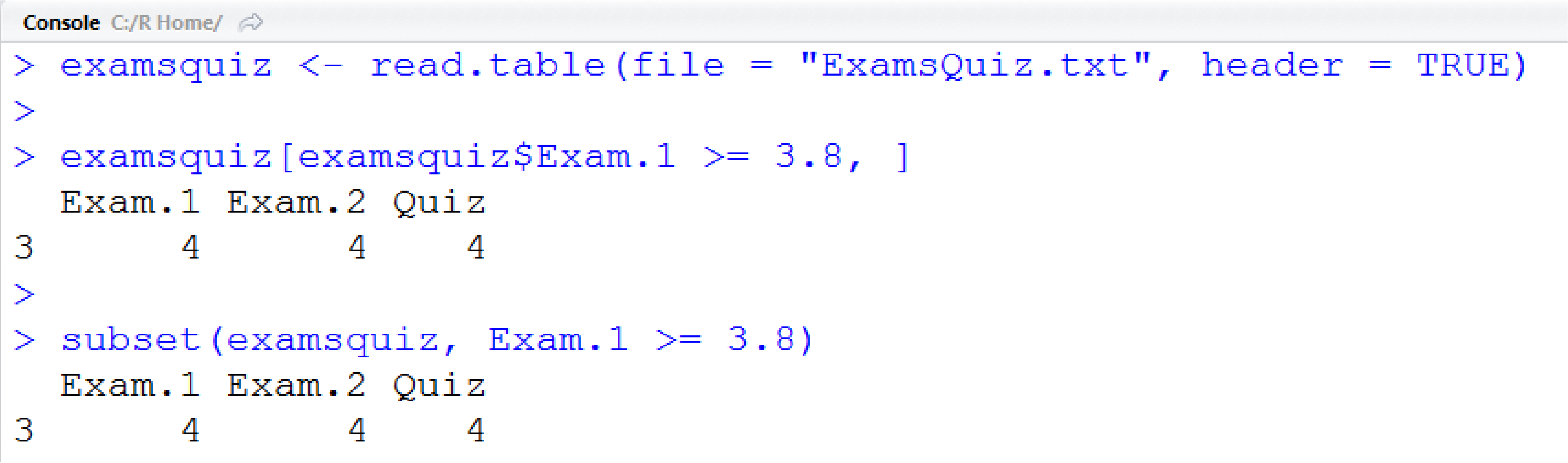
- In any subsequent statistical analyses, R would do its best to cope with the missing data.
- However, in some situations, we need to set the option `na.rm=TRUE`, explicitly telling R to ignore `NA` values.
- For instance, with the missing exam score, calculating the mean score on exam 2 by calling R's `mean()` function would skip that first student in finding the mean.
- Otherwise, R would just report `NA` for the mean.



```
> x <- c(2, NA, 4)
> mean(x)
[1] NA
>
> mean(x, na.rm = TRUE)
[1] 3
```

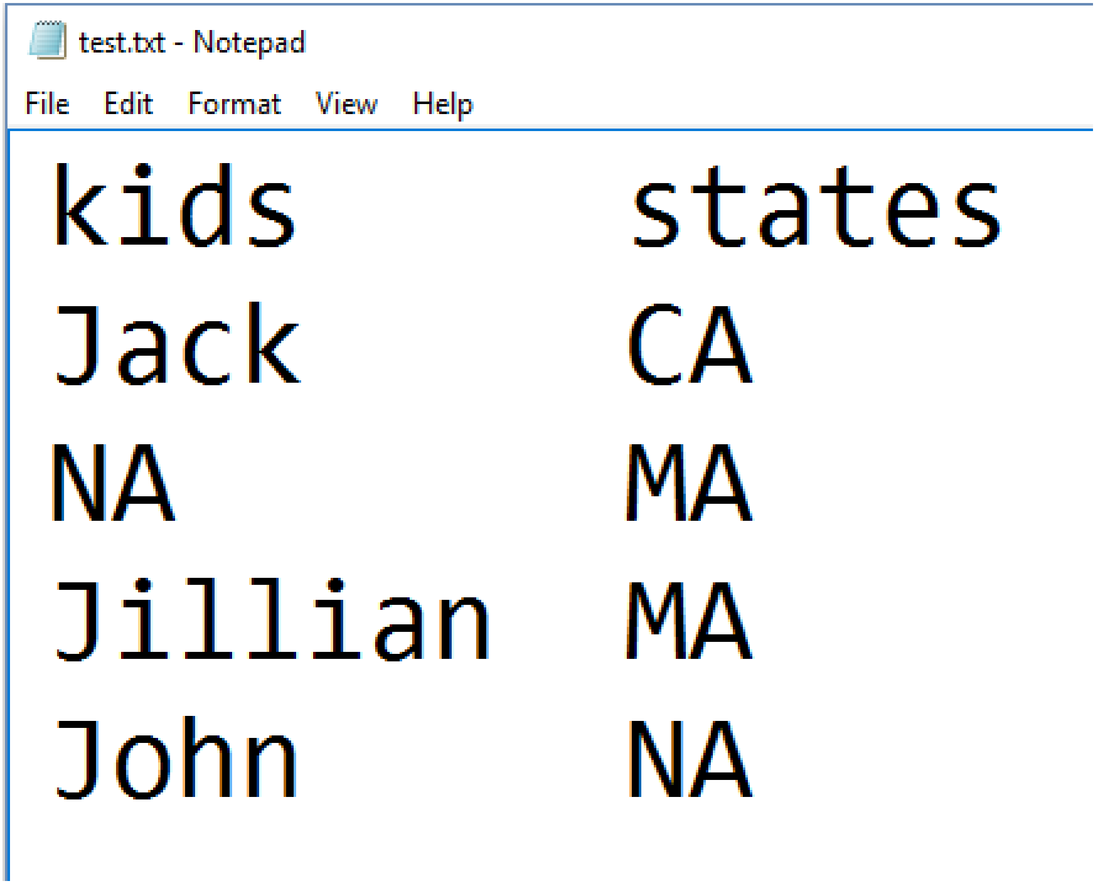
More on Treatment of NA Values (Cont.)

- `subset()` function, which saves you the trouble of specifying `na.rm=TRUE`.
- Can apply it in data frames for row selection.
- The column names are taken in the context of the given data frame.



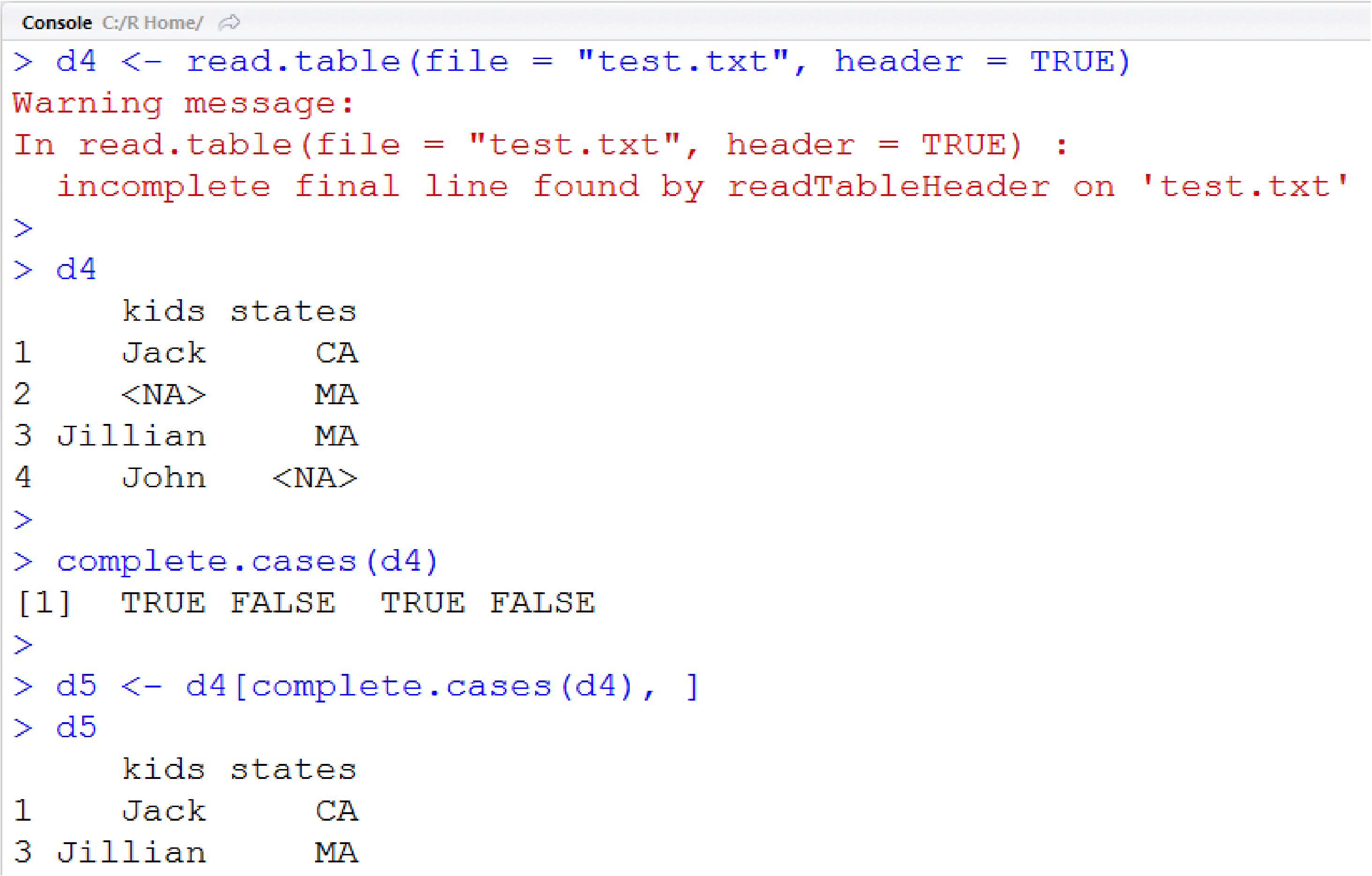
```
> examsquiz <- read.table(file = "ExamsQuiz.txt", header = TRUE)
>
> examsquiz[examsquiz$Exam.1 >= 3.8, ]
  Exam.1 Exam.2 Quiz
3      4      4    4
>
> subset(examsquiz, Exam.1 >= 3.8)
  Exam.1 Exam.2 Quiz
3      4      4    4
```

More on Treatment of NA Values (Cont.)



```
kids      states
Jack      CA
NA        MA
Jillian   MA
John      NA
```

More on Treatment of NA Values (Cont.)

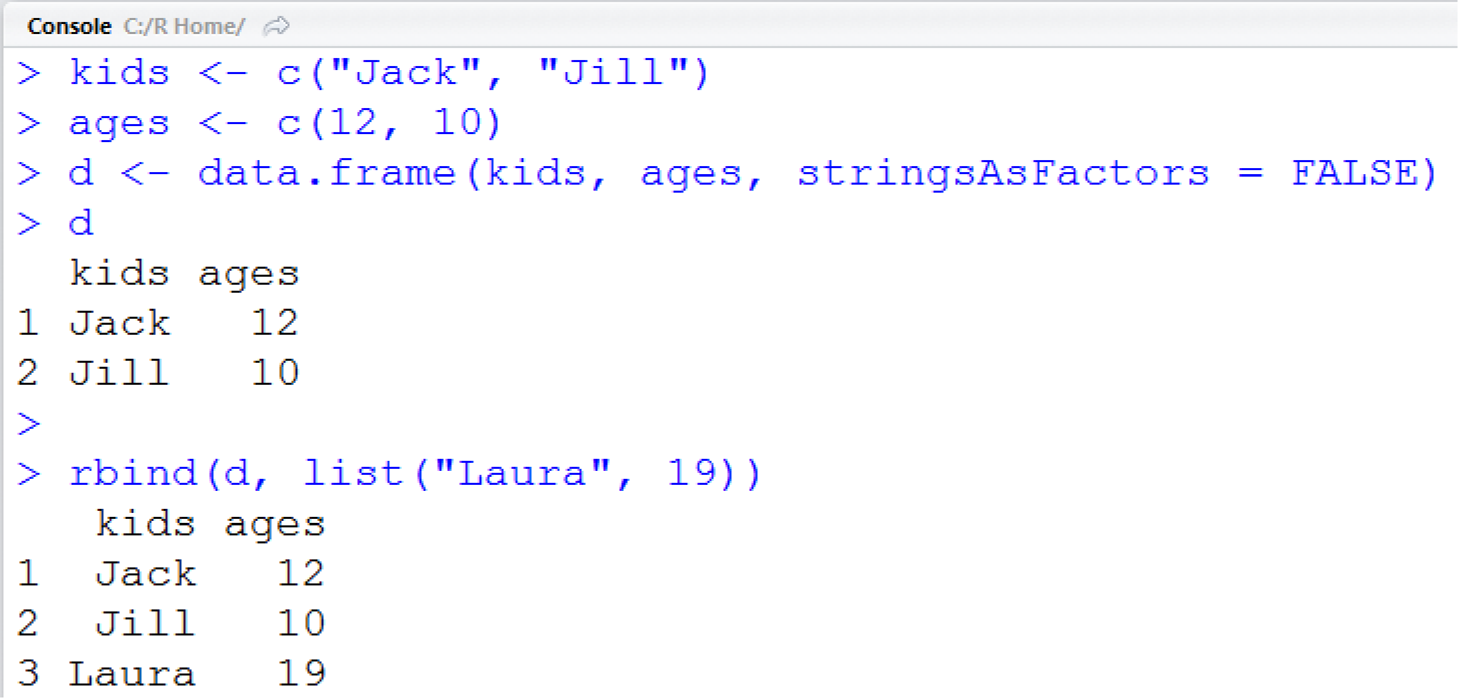


```
> d4 <- read.table(file = "test.txt", header = TRUE)
Warning message:
In read.table(file = "test.txt", header = TRUE) :
  incomplete final line found by readTableHeader on 'test.txt'
>
> d4
     kids states
1    Jack     CA
2    <NA>     MA
3 Jillian     MA
4    John   <NA>
>
> complete.cases(d4)
[1]  TRUE FALSE  TRUE FALSE
>
> d5 <- d4[complete.cases(d4), ]
> d5
     kids states
1    Jack     CA
3 Jillian     MA
```

- In some cases, we may wish to rid our data frame of any observation that has at least one **NA** value.
- A handy function for this purpose is `complete.cases()`.

Using the rbind() and cbind() Functions and Alternatives

- Can use **cbind()** to add a new column that has the same length as the existing columns.
- In using **rbind()** to add a row, the added row is typically in the form of another data frame or list.

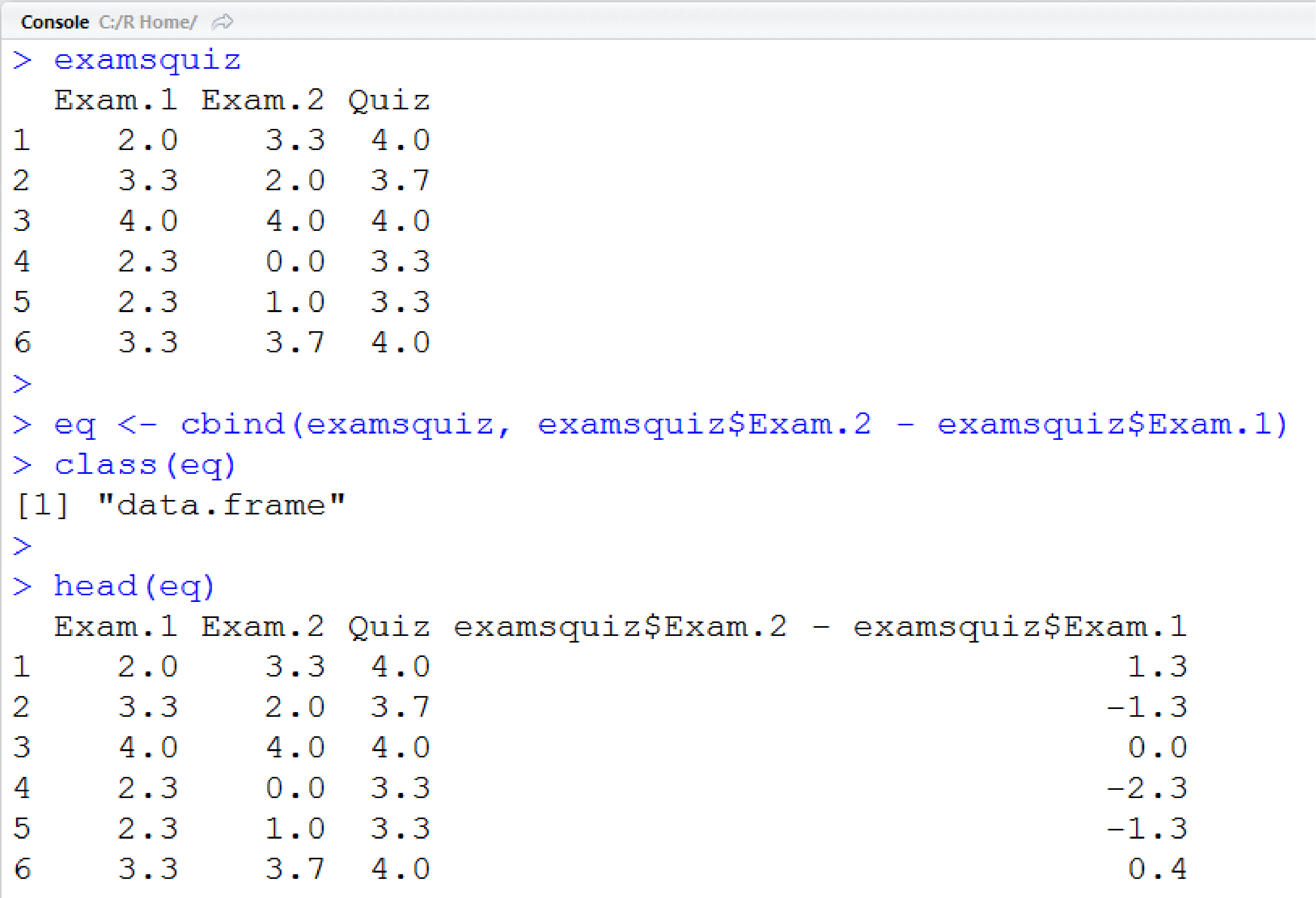


```
> kids <- c("Jack", "Jill")
> ages <- c(12, 10)
> d <- data.frame(kids, ages, stringsAsFactors = FALSE)
> d
  kids ages
1 Jack   12
2 Jill   10
>
> rbind(d, list("Laura", 19))
   kids ages
1  Jack   12
2  Jill   10
3 Laura   19
```

Using the rbind() and cbind() Functions and Alternatives (Cont.)

- Can also create new columns from old ones.
- For instance, can add a variable that is the difference between exams 1 and 2.

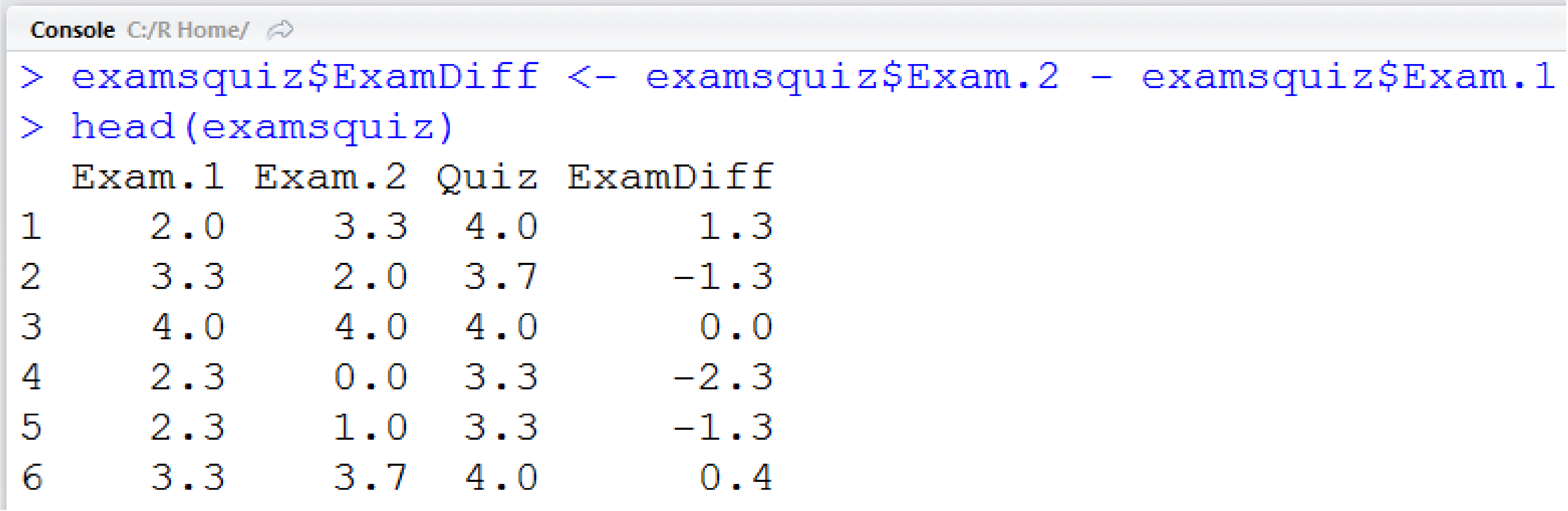
Using the rbind() and cbind() Functions and Alternatives (Cont.)



```
> examsquiz
  Exam.1 Exam.2 Quiz
1    2.0    3.3  4.0
2    3.3    2.0  3.7
3    4.0    4.0  4.0
4    2.3    0.0  3.3
5    2.3    1.0  3.3
6    3.3    3.7  4.0
>
> eq <- cbind(examsquiz, examsquiz$Exam.2 - examsquiz$Exam.1)
> class(eq)
[1] "data.frame"
>
> head(eq)
  Exam.1 Exam.2 Quiz examsquiz$Exam.2 - examsquiz$Exam.1
1    2.0    3.3  4.0                                 1.3
2    3.3    2.0  3.7                                -1.3
3    4.0    4.0  4.0                                 0.0
4    2.3    0.0  3.3                                -2.3
5    2.3    1.0  3.3                                -1.3
6    3.3    3.7  4.0                                 0.4
```

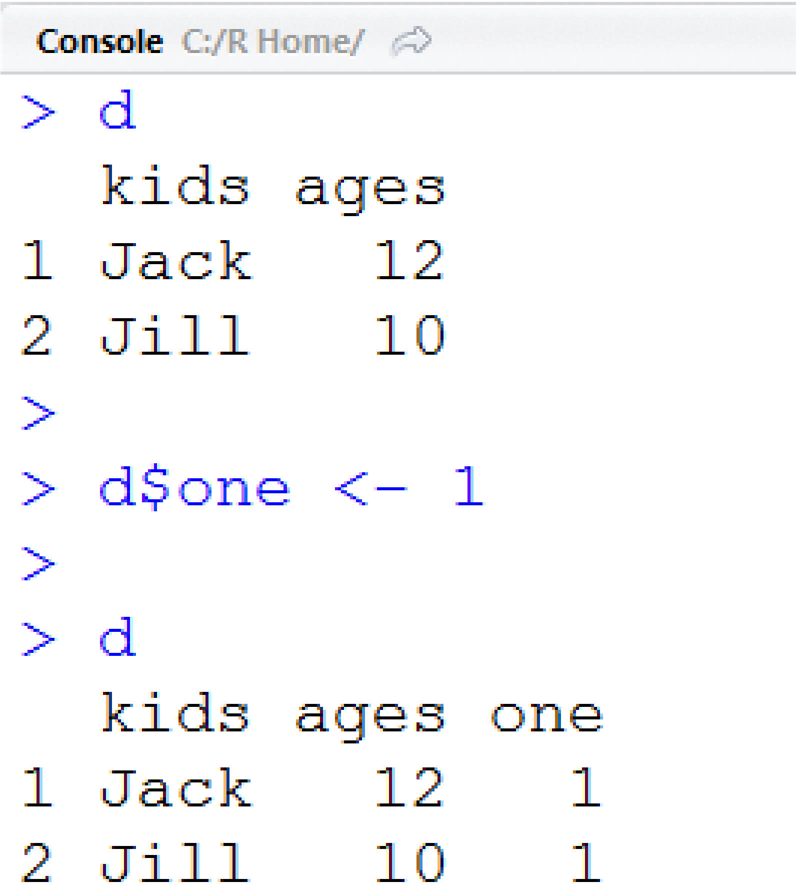
Using the rbind() and cbind() Functions and Alternatives (Cont.)

- The new name is rather unwieldy.
- It’s long, and it has embedded blanks.



```
> examsquiz$ExamDiff <- examsquiz$Exam.2 - examsquiz$Exam.1
> head(examsquiz)
  Exam.1 Exam.2 Quiz ExamDiff
1    2.0    3.3  4.0      1.3
2    3.3    2.0  3.7     -1.3
3    4.0    4.0  4.0      0.0
4    2.3    0.0  3.3     -2.3
5    2.3    1.0  3.3     -1.3
6    3.3    3.7  4.0      0.4
```

Using the rbind() and cbind() Functions and Alternatives (Cont.)

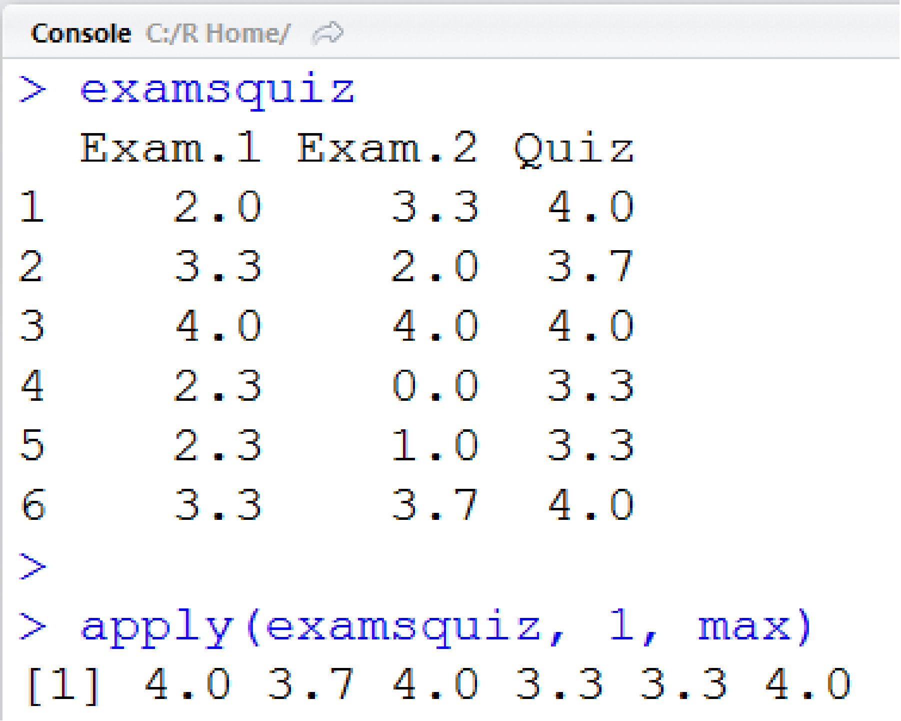


```
> d
  kids ages
1 Jack   12
2 Jill   10
>
> d$one <- 1
>
> d
  kids ages one
1 Jack   12   1
2 Jill   10   1
```

- Can even exploit recycling to add a column that is of a different length than those in the data frame.

Applying apply()

- You can use apply() on data frames, if the columns are all of the same type.
- For instance, we can find the maximum grade for each student.



```
> examsquiz
  Exam.1 Exam.2 Quiz
1    2.0    3.3  4.0
2    3.3    2.0  3.7
3    4.0    4.0  4.0
4    2.3    0.0  3.3
5    2.3    1.0  3.3
6    3.3    3.7  4.0
>
> apply(examsquiz, 1, max)
[1] 4.0 3.7 4.0 3.3 3.3 4.0
```

Merging Data Frames

- In the relational database world, one of the most important operations is that of a *join.*
- Join two tables can be combined according to the values
  of a common variable.
- In R, two data frames can be similarly combined using the merge() function.

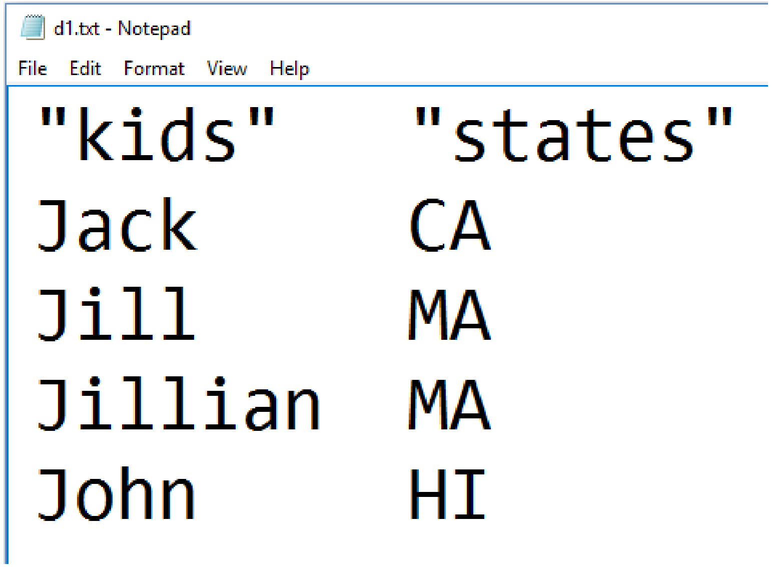
Merging Data Frames (Cont.)

- The simplest form is as follows:

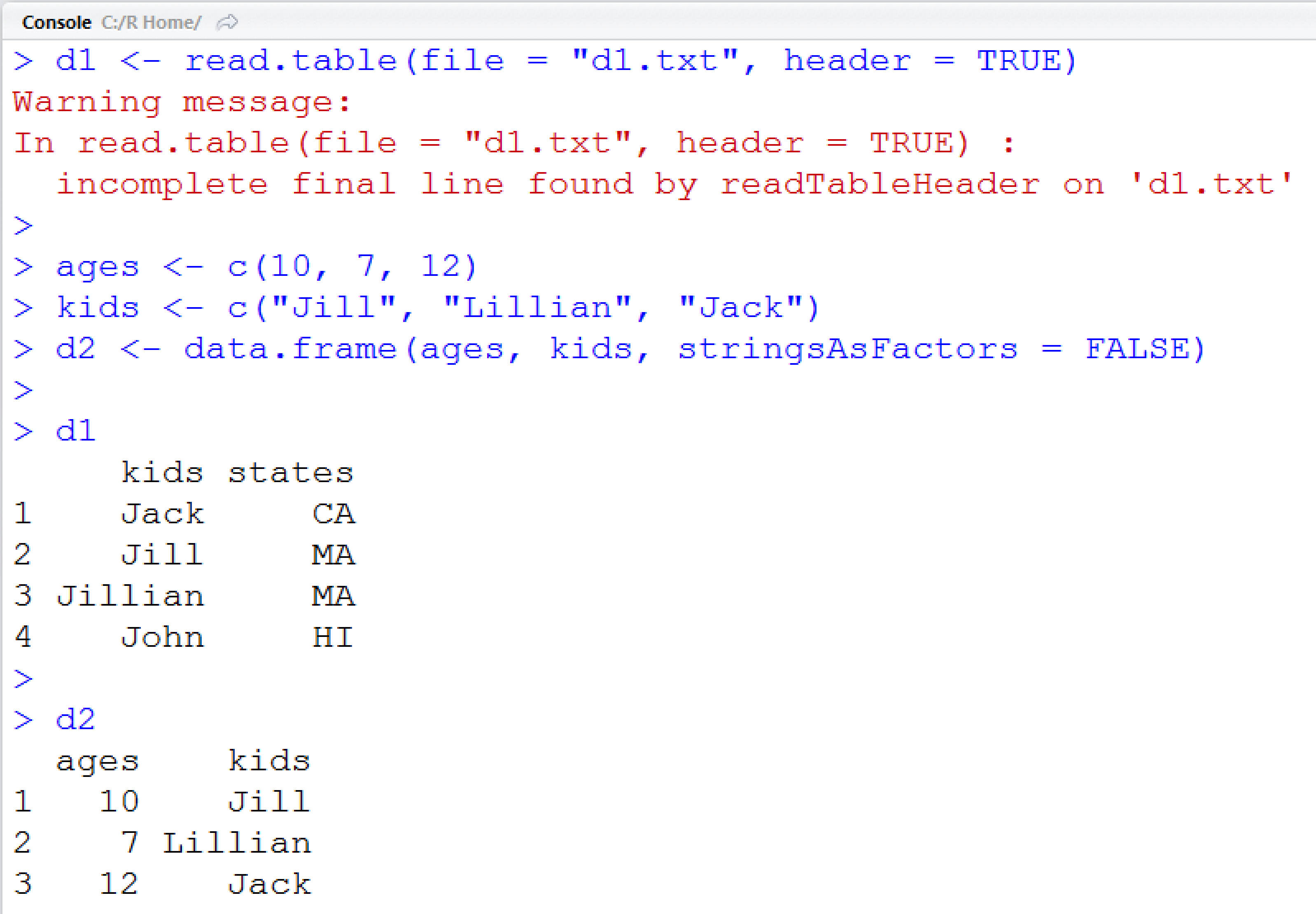
```
merge(x,y)
```

- This merges data frames x and y.
- It assumes that the two data frames have one or more columns with names in common.

Merging Data Frames (Cont.)



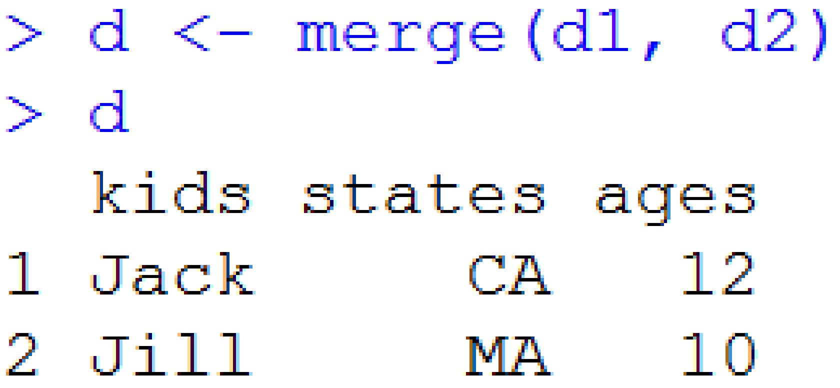
```
"kids"    "states"
Jack      CA
Jill      MA
Jillian   MA
John      HI
```



```
> d1 <- read.table(file = "d1.txt", header = TRUE)
Warning message:
In read.table(file = "d1.txt", header = TRUE) :
  incomplete final line found by readTableHeader on 'd1.txt'
>
> ages <- c(10, 7, 12)
> kids <- c("Jill", "Lillian", "Jack")
> d2 <- data.frame(ages, kids, stringsAsFactors = FALSE)
>
> d1
     kids states
1    Jack     CA
2    Jill     MA
3 Jillian     MA
4    John     HI
>
> d2
  ages    kids
1   10    Jill
2    7 Lillian
3   12    Jack
```

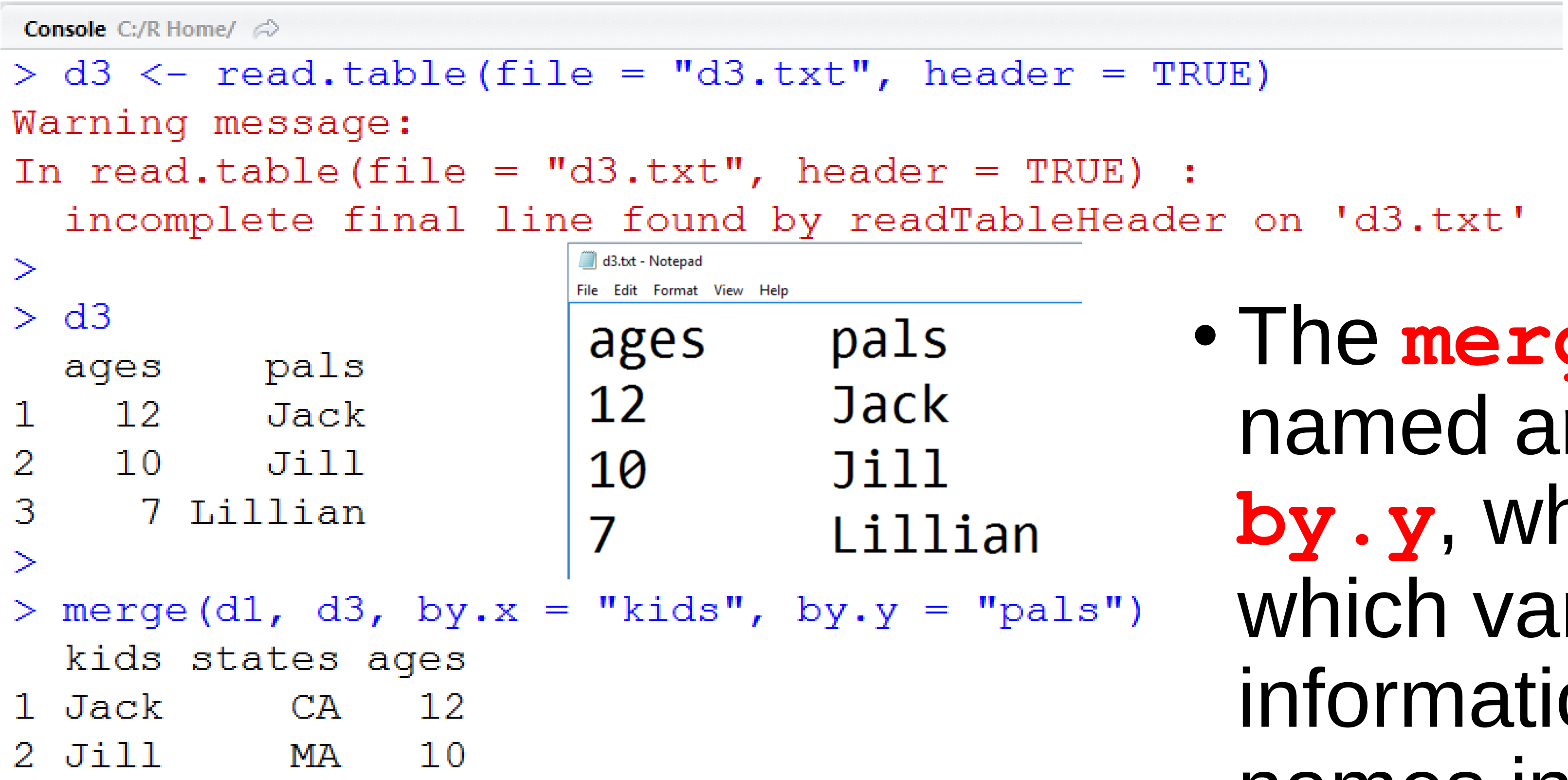
Merging Data Frames (Cont.)

- The two data frames have the variable kids in common.
- R found the rows in which this variable had the same value of kids in both data frames(the ones for Jack and Jill).
- It then created a data frame with corresponding rows and with columns taken from data frames (kids, states, and ages).



```
> d <- merge(d1, d2)
> d
  kids states ages
1 Jack     CA   12
2 Jill     MA   10
```

Merging Data Frames (Cont.)

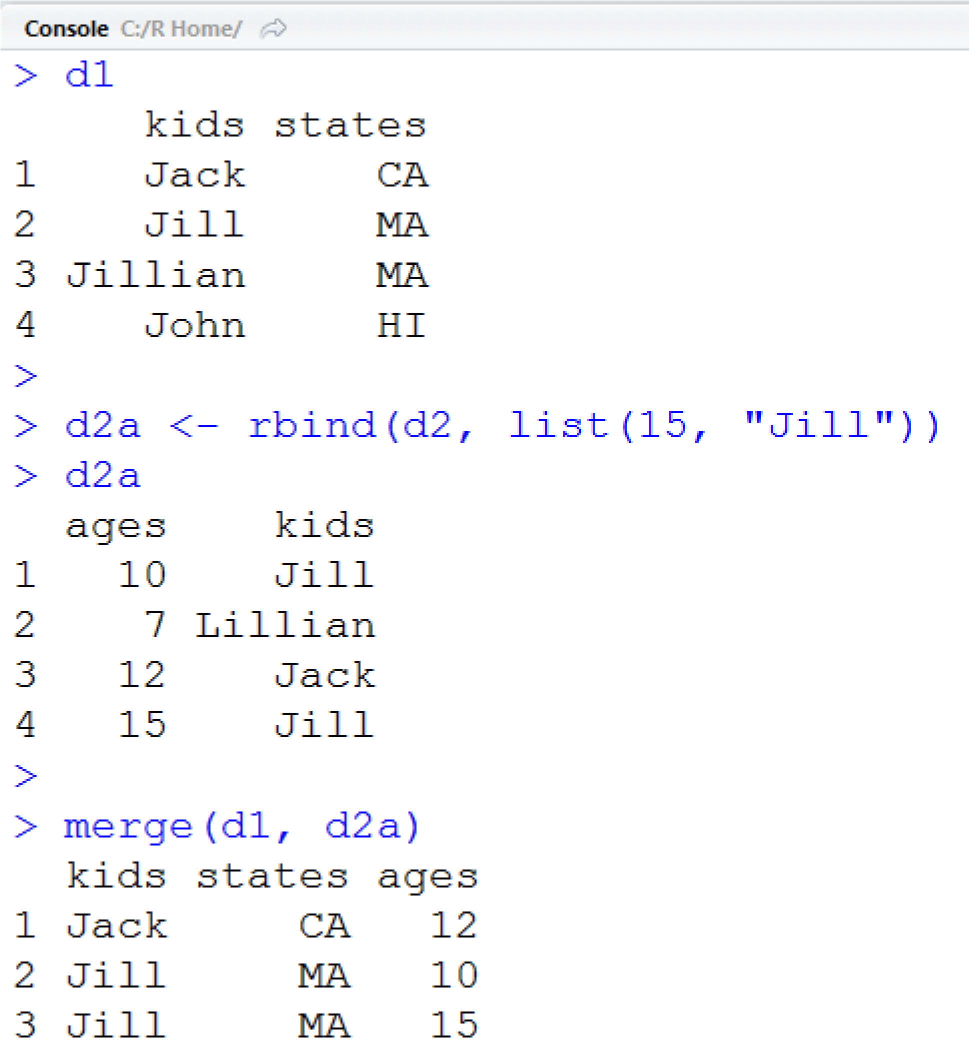


```
> d3 <- read.table(file = "d3.txt", header = TRUE)
Warning message:
In read.table(file = "d3.txt", header = TRUE) :
  incomplete final line found by readTableHeader on 'd3.txt'
>
> d3
  ages    pals
1   12    Jack
2   10    Jill
3    7 Lillian
>
> merge(d1, d3, by.x = "kids", by.y = "pals")
  kids states ages
1 Jack     CA   12
2 Jill     MA   10
```

```
ages      pals
12        Jack
10        Jill
7         Lillian
```

- The **merge()** function has named arguments **by.x** and **by.y**, which handle cases in which variables have similar information but different names in the two data frames

Merging Data Frames (Cont.)



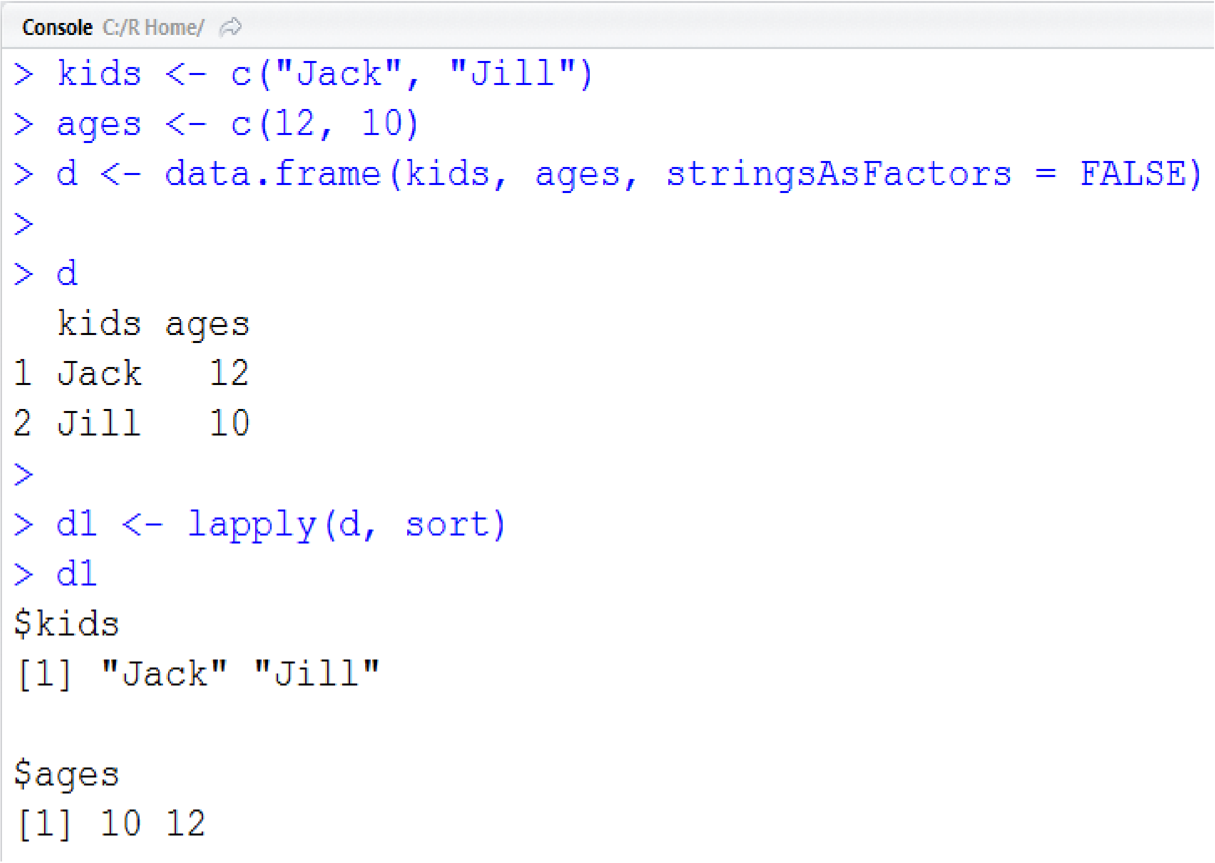
```
> d1
     kids states
1    Jack     CA
2    Jill     MA
3 Jillian     MA
4    John     HI
>
> d2a <- rbind(d2, list(15, "Jill"))
> d2a
  ages    kids
1   10    Jill
2    7 Lillian
3   12    Jack
4   15    Jill
>
> merge(d1, d2a)
  kids states ages
1 Jack     CA   12
2 Jill     MA   10
3 Jill     MA   15
```

- Duplicate matches will appear in full in the result, possibly in undesirable ways.
- There are *two* Jills in d2a.
- There is a Jill in d1 who lives in MA and another Jill with unknown residence.
- In previous example, `merge(d1,d2)`, there was only one Jill, who was presumed to be the same person in both data frames.
- But here, in the call `merge(d1,d2a)` , it may have been the case that only one of the `Jills` was a MS resident.

Applying Functions to Data Frames

- As with lists, you can use the `lapply` and `sapply` functions with data frames.

Using lapply() and sapply() on Data Frames



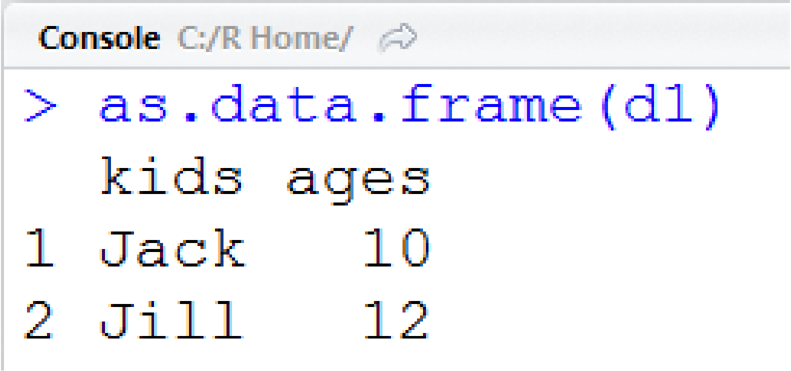
```
> kids <- c("Jack", "Jill")
> ages <- c(12, 10)
> d <- data.frame(kids, ages, stringsAsFactors = FALSE)
>
> d
  kids ages
1 Jack   12
2 Jill   10
>
> d1 <- lapply(d, sort)
> d1
$kids
[1] "Jack" "Jill"

$ages
[1] 10 12
```

- Data frames are special cases of lists, with the list components consisting of the data frame’s columns.
- Thus, if you call `lapply()` on a data frame with a specified function `f()`, then `f()` will be called on each of the frame’s columns, with the return values placed in a list.

Using lapply() and sapply() on Data Frames (Cont.)

- Note that dl is just a list, not a data frame. We could coerce it to a data frame, like this.



```
> as.data.frame(dl)
  kids ages
1 Jack   10
2 Jill   12
```

- But this would make no sense, as the correspondence between names and ages has been lost.
- `Jack`, for instance, is now listed as `10` years old instead of `12`.