R Objects

Dim

- You can transform an atomic vector into an *n*-dimensional array by giving it a dimensions attribute with `dim`.

Console Terminal

R 4.4.1 · ~/

```
> die
[1] 1 2 3 4 5 6
> dim(die) <- c(2, 3)
> die
     [,1] [,2] [,3]
[1,]    1    3    5
[2,]    2    4    6
>
```

R Objects

Dim

- You can transform an atomic vector into an *n*-dimensional array by giving it a dimensions attribute with `dim`.
- To do this, set the `dim` attribute to a numeric vector of length *n*.

Console Terminal

R 4.4.1 · ~/

```
> die
[1] 1 2 3 4 5 6
> dim(die) <- c(2, 3)
> die
     [,1] [,2] [,3]
[1,]    1    3    5
[2,]    2    4    6
>
```

R Objects

Dim

- You can transform an atomic vector into an *n*-dimensional array by giving it a dimensions attribute with `dim`.
- To do this, set the `dim` attribute to a numeric vector of length *n*.
- R will reorganize the elements of the vector into *n* dimensions.

Console Terminal

R 4.4.1 · ~/

```
> die
[1] 1 2 3 4 5 6
> dim(die) <- c(2, 3)
> die
     [,1] [,2] [,3]
[1,]    1    3    5
[2,]    2    4    6
>
```

R Objects

Dim

- You can transform an atomic vector into an *n*-dimensional array by giving it a dimensions attribute with `dim`.
- To do this, set the `dim` attribute to a numeric vector of length *n*.
- R will reorganize the elements of the vector into *n* dimensions.
- Each dimension will have as many rows (or columns, etc.) as the *nth* value of the `dim` vector.

Console Terminal

R 4.4.1 · ~/

```
> die
[1] 1 2 3 4 5 6
> dim(die) <- c(2, 3)
> die
     [,1] [,2] [,3]
[1,]    1    3    5
[2,]    2    4    6
>
```

R Objects

Matrices

Console Terminal

R 4.4.1 · ~/

```
> m <- matrix(die, nrow = 2)
> m
     [,1] [,2] [,3]
[1,]    1    3    5
[2,]    2    4    6
>
```

R Objects

Matrices

- Matrices store values in a two-dimensional array, just like a matrix from linear algebra.

Console Terminal

R 4.4.1 · ~/

```
> m <- matrix(die, nrow = 2)
> m
     [,1] [,2] [,3]
[1,]    1    3    5
[2,]    2    4    6
>
```

R Objects

Matrices

- Matrices store values in a two-dimensional array, just like a matrix from linear algebra.
- To create one, first give `matrix` an atomic vector to reorganize into a matrix.

Console Terminal

R 4.4.1 · ~/

```
> m <- matrix(die, nrow = 2)
> m
     [,1] [,2] [,3]
[1,]    1    3    5
[2,]    2    4    6
>
```

R Objects

Matrices

- Matrices store values in a two-dimensional array, just like a matrix from linear algebra.
- To create one, first give `matrix` an atomic vector to reorganize into a matrix.
- Then, define how many rows should be in the matrix by setting the `nrow` argument to a number. `matrix` will organize your vector of values into a matrix with the specified number of rows.

Console Terminal

R 4.4.1 · ~/

```
> m <- matrix(die, nrow = 2)
> m
     [,1] [,2] [,3]
[1,]    1    3    5
[2,]    2    4    6
>
```

R Objects

Matrices

- Matrices store values in a two-dimensional array, just like a matrix from linear algebra.
- To create one, first give `matrix` an atomic vector to reorganize into a matrix.
- Then, define how many rows should be in the matrix by setting the `nrow` argument to a number. `matrix` will organize your vector of values into a matrix with the specified number of rows.
- Alternatively, you can set the `ncol` argument, which tells R how many columns to include in the matrix.

Console Terminal

R 4.4.1 · ~/

```
> m <- matrix(die, nrow = 2)
> m
     [,1] [,2] [,3]
[1,]    1    3    5
[2,]    2    4    6
>
```

R Objects

Matrices

Console Terminal

R 4.4.1 · ~/

```
> m
     [,1] [,2] [,3]
[1,]    1    3    5
[2,]    2    4    6
> m <- matrix(die, nrow = 2, byrow = TRUE)
> m
     [,1] [,2] [,3]
[1,]    1    2    3
[2,]    4    5    6
>
```

R Objects

Matrices

- `matrix` will fill up the matrix column by column by default, but you can fill the matrix row by row if you include the argument `byrow = TRUE`

Console Terminal

R 4.4.1 · ~/

```
> m
     [,1] [,2] [,3]
[1,]    1    3    5
[2,]    2    4    6
> m <- matrix(die, nrow = 2, byrow = TRUE)
> m
     [,1] [,2] [,3]
[1,]    1    2    3
[2,]    4    5    6
>
```

R Objects

Matrices

- `matrix` will fill up the matrix column by column by default, but you can fill the matrix row by row if you include the argument `byrow = TRUE`
- `matrix` also has other default arguments that you can use to customize your matrix. You can read about them at `matrix`'s help page (accessible by `?matrix`).

Console Terminal

R 4.4.1 · ~/

```
> m
     [,1] [,2] [,3]
[1,]    1    3    5
[2,]    2    4    6
> m <- matrix(die, nrow = 2, byrow = TRUE)
> m
     [,1] [,2] [,3]
[1,]    1    2    3
[2,]    4    5    6
>
```

R Objects

Arrays

Console Terminal

R 4.4.1 · ~/

```
> ar <- array(c(1:3, 11:13, 21:23), dim = c(3, 3, 3))
> ar
, , 1

     [,1] [,2] [,3]
[1,]    1   11   21
[2,]    2   12   22
[3,]    3   13   23

, , 2

     [,1] [,2] [,3]
[1,]    1   11   21
[2,]    2   12   22
[3,]    3   13   23

, , 3

     [,1] [,2] [,3]
[1,]    1   11   21
[2,]    2   12   22
[3,]    3   13   23
```

R Objects

Arrays

- The `array` function creates an n-dimensional array.

Console Terminal

R 4.4.1 · ~/

```
> ar <- array(c(1:3, 11:13, 21:23), dim = c(3, 3, 3))
> ar
, , 1

     [,1] [,2] [,3]
[1,]    1   11   21
[2,]    2   12   22
[3,]    3   13   23

, , 2

     [,1] [,2] [,3]
[1,]    1   11   21
[2,]    2   12   22
[3,]    3   13   23

, , 3

     [,1] [,2] [,3]
[1,]    1   11   21
[2,]    2   12   22
[3,]    3   13   23
```

R Objects

Arrays

- The `array` function creates an n-dimensional array.
- `array` is not as customizeable as `matrix` and basically does the same thing as setting the `dim` attribute.

Console Terminal

R 4.4.1 · ~/

```
> ar <- array(c(1:3, 11:13, 21:23), dim = c(3, 3, 3))
> ar
, , 1

     [,1] [,2] [,3]
[1,]    1   11   21
[2,]    2   12   22
[3,]    3   13   23

, , 2

     [,1] [,2] [,3]
[1,]    1   11   21
[2,]    2   12   22
[3,]    3   13   23

, , 3

     [,1] [,2] [,3]
[1,]    1   11   21
[2,]    2   12   22
[3,]    3   13   23
```

R Objects

Arrays

- The `array` function creates an n-dimensional array.
- `array` is not as customizeable as `matrix` and basically does the same thing as setting the `dim` attribute.
- To use `array`, provide an atomic vector as the first argument, and a vector of dimensions as the second argument, called `dim`

Console Terminal

R 4.4.1 · ~/

```
> ar <- array(c(1:3, 11:13, 21:23), dim = c(3, 3, 3))
> ar
, , 1

     [,1] [,2] [,3]
[1,]    1   11   21
[2,]    2   12   22
[3,]    3   13   23

, , 2

     [,1] [,2] [,3]
[1,]    1   11   21
[2,]    2   12   22
[3,]    3   13   23

, , 3

     [,1] [,2] [,3]
[1,]    1   11   21
[2,]    2   12   22
[3,]    3   13   23
```

R Objects

Class

Console Terminal

R 4.4.1 · ~/

```
> die <- c(1, 2, 3, 4, 5, 6)
> die
[1] 1 2 3 4 5 6
> typeof(die)
[1] "double"
> attributes(die)
NULL
> class(die)
[1] "numeric"
> dim(die) <- c(2, 3)
> typeof(die)
[1] "double"
> attributes(die)
$dim
[1] 2 3

> class(die)
[1] "matrix" "array"
>
```

R Objects

Class

- Notice that changing the dimensions of your object will not change the type of the object, but it *will* change the object's `class` attribute.

Console Terminal

R 4.4.1 · ~/

```
> die <- c(1, 2, 3, 4, 5, 6)
> die
[1] 1 2 3 4 5 6
> typeof(die)
[1] "double"
> attributes(die)
NULL
> class(die)
[1] "numeric"
> dim(die) <- c(2, 3)
> typeof(die)
[1] "double"
> attributes(die)
$dim
[1] 2 3

> class(die)
[1] "matrix" "array"
>
```

R Objects

Class

- Notice that changing the dimensions of your object will not change the type of the object, but it *will* change the object's `class` attribute.
- A matrix is a special case of an atomic vector.

Console Terminal

R 4.4.1 · ~/

```
> die <- c(1, 2, 3, 4, 5, 6)
> die
[1] 1 2 3 4 5 6
> typeof(die)
[1] "double"
> attributes(die)
NULL
> class(die)
[1] "numeric"
> dim(die) <- c(2, 3)
> typeof(die)
[1] "double"
> attributes(die)
$dim
[1] 2 3

> class(die)
[1] "matrix" "array"
>
```

R Objects

Class

- Notice that changing the dimensions of your object will not change the type of the object, but it *will* change the object's `class` attribute.
- A matrix is a special case of an atomic vector.
- Every element in the matrix is still a double, but the elements have been arranged into a new structure.

Console Terminal

R 4.4.1 · ~/

```
> die <- c(1, 2, 3, 4, 5, 6)
> die
[1] 1 2 3 4 5 6
> typeof(die)
[1] "double"
> attributes(die)
NULL
> class(die)
[1] "numeric"
> dim(die) <- c(2, 3)
> typeof(die)
[1] "double"
> attributes(die)
$dim
[1] 2 3

> class(die)
[1] "matrix" "array"
>
```

R Objects

Class

- Notice that changing the dimensions of your object will not change the type of the object, but it *will* change the object's `class` attribute.
- A matrix is a special case of an atomic vector.
- Every element in the matrix is still a double, but the elements have been arranged into a new structure.
- R added a `class` attribute to `die` when you changed its dimensions. Many R functions will specifically look for an object's `class` attribute.

Console Terminal

R 4.4.1 · ~/

```
> die <- c(1, 2, 3, 4, 5, 6)
> die
[1] 1 2 3 4 5 6
> typeof(die)
[1] "double"
> attributes(die)
NULL
> class(die)
[1] "numeric"
> dim(die) <- c(2, 3)
> typeof(die)
[1] "double"
> attributes(die)
$dim
[1] 2 3

> class(die)
[1] "matrix" "array"
>
```

R Objects

Class

- Notice that changing the dimensions of your object will not change the type of the object, but it *will* change the object's `class` attribute.
- A matrix is a special case of an atomic vector.
- Every element in the matrix is still a double, but the elements have been arranged into a new structure.
- R added a `class` attribute to `die` when you changed its dimensions. Many R functions will specifically look for an object's `class` attribute.
- Note that an object's `class` attribute will not always appear when you run `attributes`; you may need to specifically search for it with `class`

Console Terminal

R 4.4.1 · ~/

```
> die <- c(1, 2, 3, 4, 5, 6)
> die
[1] 1 2 3 4 5 6
> typeof(die)
[1] "double"
> attributes(die)
NULL
> class(die)
[1] "numeric"
> dim(die) <- c(2, 3)
> typeof(die)
[1] "double"
> attributes(die)
$dim
[1] 2 3

> class(die)
[1] "matrix" "array"
>
```

R Objects

Dates and Times

Console Terminal

R 4.4.1 · ~/

```
> now <- Sys.time()
> now
[1] "2024-08-09 18:34:18 EDT"
>
```

R Objects

Dates and Times

- The attribute system lets R represent more types of data than just doubles, integers, characters, logicals, complexes, and raws. The time looks like a character string when you display it, but its data type is actually `"double"`, and its class is `"POSIXct"` `"POSIXt"` (it has two classes)

Console Terminal

R 4.4.1 · ~/

```
> now <- Sys.time()
> now
[1] "2024-08-09 18:34:18 EDT"
>
```

R Objects

POSIXct

Console Terminal

R 4.4.1 · ~/

```
> now <- Sys.time()
> now
[1] "2024-08-09 18:34:18 EDT"
> typeof(now)
[1] "double"
> class(now)
[1] "POSIXct" "POSIXt"
> unclass(now)
[1] 1723242859
>
```

R Objects

POSIXct

- POSIXct is a widely used framework for representing dates and times.

Console Terminal

R 4.4.1 · ~/

```
> now <- Sys.time()
> now
[1] "2024-08-09 18:34:18 EDT"
> typeof(now)
[1] "double"
> class(now)
[1] "POSIXct" "POSIXt"
> unclass(now)
[1] 1723242859
>
```

R Objects

POSIXct

- POSIXct is a widely used framework for representing dates and times.
- In the POSIXct framework, each time is represented by the number of seconds that have passed between the time and 12:00 AM January 1st 1970 (UTC).

Console Terminal

R 4.4.1 · ~/

```
> now <- Sys.time()
> now
[1] "2024-08-09 18:34:18 EDT"
> typeof(now)
[1] "double"
> class(now)
[1] "POSIXct" "POSIXt"
> unclass(now)
[1] 1723242859
>
```

R Objects

POSIXct

- POSIXct is a widely used framework for representing dates and times.
- In the POSIXct framework, each time is represented by the number of seconds that have passed between the time and 12:00 AM January 1st 1970 (UTC).
- R creates the time object by building a double vector with one element, `1723242859`. You can see this vector by removing the `class` attribute of `now`, or by using the `unclass` function, which does the same thing

Console Terminal

R 4.4.1 · ~/

```
> now <- Sys.time()
> now
[1] "2024-08-09 18:34:18 EDT"
> typeof(now)
[1] "double"
> class(now)
[1] "POSIXct" "POSIXt"
> unclass(now)
[1] 1723242859
>
```

R Objects

POSIXct

Console Terminal

R 4.4.1 · ~/

```
> mil <- 1000000
> class(mil) <- c("POSIXct", "POSIXt")
> mil
[1] "1970-01-12 08:46:40 EST"
>
```

R Objects

POSIXct

- You can take advantage of this system by giving the `POSIXct` class to random R objects. For example, have you ever wondered what day it was a million seconds after 12:00 a.m. Jan. 1, 1970?

Console Terminal

R 4.4.1 · ~/

```
> mil <- 1000000
> class(mil) <- c("POSIXct", "POSIXt")
> mil
[1] "1970-01-12 08:46:40 EST"
>
```

R Objects

POSIXct

- You can take advantage of this system by giving the `POSIXct` class to random R objects. For example, have you ever wondered what day it was a million seconds after 12:00 a.m. Jan. 1, 1970?
- Jan. 12, 1970. A million seconds goes by faster than you would think. This conversion worked well because the `POSIXct` class does not rely on any additional attributes, but in general, forcing the class of an object is a bad idea.

Console Terminal

R 4.4.1 · ~/

```
> mil <- 1000000
> class(mil) <- c("POSIXct", "POSIXt")
> mil
[1] "1970-01-12 08:46:40 EST"
>
```

R Objects

Factors

Console Terminal

R 4.4.1 · ~/

```
> car <- c("Volkswagen", "Alpine", "Mercedes", "Audi")
> car
[1] "Volkswagen" "Alpine"     "Mercedes"   "Audi"
> typeof(car)
[1] "character"
> attributes(car)
NULL
> car <- factor(car)
> car
[1] Volkswagen Alpine     Mercedes   Audi
Levels: Alpine Audi Mercedes Volkswagen
> typeof(car)
[1] "integer"
> attributes(car)
$levels
[1] "Alpine"     "Audi"       "Mercedes"   "Volkswagen"

$class
[1] "factor"
```

R Objects

Factors

- Factors are R's way of storing categorical information, like ethnicity or eye color.

Console Terminal

R 4.4.1 · ~/

```
> car <- c("Volkswagen", "Alpine", "Mercedes", "Audi")
> car
[1] "Volkswagen" "Alpine"     "Mercedes"   "Audi"
> typeof(car)
[1] "character"
> attributes(car)
NULL
> car <- factor(car)
> car
[1] Volkswagen Alpine     Mercedes   Audi
Levels: Alpine Audi Mercedes Volkswagen
> typeof(car)
[1] "integer"
> attributes(car)
$levels
[1] "Alpine"     "Audi"       "Mercedes"   "Volkswagen"

$class
[1] "factor"
```

R Objects

Factors

- Factors are R's way of storing categorical information, like ethnicity or eye color.
- A factor can only have certain values and these values may have their own idiosyncratic order.

Console Terminal

R 4.4.1 · ~/

```
> car <- c("Volkswagen", "Alpine", "Mercedes", "Audi")
> car
[1] "Volkswagen" "Alpine"     "Mercedes"   "Audi"
> typeof(car)
[1] "character"
> attributes(car)
NULL
> car <- factor(car)
> car
[1] Volkswagen Alpine     Mercedes   Audi
Levels: Alpine Audi Mercedes Volkswagen
> typeof(car)
[1] "integer"
> attributes(car)
$levels
[1] "Alpine"     "Audi"       "Mercedes"   "Volkswagen"

$class
[1] "factor"
```

R Objects

Factors

- Factors are R's way of storing categorical information, like ethnicity or eye color.
- A factor can only have certain values and these values may have their own idiosyncratic order.
- This arrangement makes factors very useful for recording the treatment levels of a study and other categorical variables.

Console Terminal

R 4.4.1 · ~/

```
> car <- c("Volkswagen", "Alpine", "Mercedes", "Audi")
> car
[1] "Volkswagen" "Alpine"     "Mercedes"   "Audi"
> typeof(car)
[1] "character"
> attributes(car)
NULL
> car <- factor(car)
> car
[1] Volkswagen Alpine     Mercedes   Audi
Levels: Alpine Audi Mercedes Volkswagen
> typeof(car)
[1] "integer"
> attributes(car)
$levels
[1] "Alpine"     "Audi"       "Mercedes"   "Volkswagen"

$class
[1] "factor"
```

R Objects

Factors

Console Terminal

R 4.4.1 · ~/

```
> car <- c("Volkswagen", "Alpine", "Mercedes", "Audi")
> car
[1] "Volkswagen" "Alpine"     "Mercedes"   "Audi"
> typeof(car)
[1] "character"
> attributes(car)
NULL
> car <- factor(car)
> car
[1] Volkswagen Alpine     Mercedes   Audi
Levels: Alpine Audi Mercedes Volkswagen
> typeof(car)
[1] "integer"
> attributes(car)
$levels
[1] "Alpine"     "Audi"       "Mercedes"   "Volkswagen"

$class
[1] "factor"
```

R Objects

Factors

- To make a factor, pass an atomic vector into the `factor` function.

Console Terminal

R 4.4.1 · ~/

```
> car <- c("Volkswagen", "Alpine", "Mercedes", "Audi")
> car
[1] "Volkswagen" "Alpine"     "Mercedes"   "Audi"
> typeof(car)
[1] "character"
> attributes(car)
NULL
> car <- factor(car)
> car
[1] Volkswagen Alpine     Mercedes   Audi
Levels: Alpine Audi Mercedes Volkswagen
> typeof(car)
[1] "integer"
> attributes(car)
$levels
[1] "Alpine"     "Audi"       "Mercedes"   "Volkswagen"

$class
[1] "factor"
```

R Objects

Factors

- To make a factor, pass an atomic vector into the `factor` function.
- R will recode the data in the vector as integers and store the results in an integer vector.

Console Terminal

R 4.4.1 · ~/

```
> car <- c("Volkswagen", "Alpine", "Mercedes", "Audi")
> car
[1] "Volkswagen" "Alpine"     "Mercedes"   "Audi"
> typeof(car)
[1] "character"
> attributes(car)
NULL
> car <- factor(car)
> car
[1] Volkswagen Alpine     Mercedes   Audi
Levels: Alpine Audi Mercedes Volkswagen
> typeof(car)
[1] "integer"
> attributes(car)
$levels
[1] "Alpine"     "Audi"       "Mercedes"   "Volkswagen"

$class
[1] "factor"
```

R Objects

Factors

- To make a factor, pass an atomic vector into the `factor` function.
- R will recode the data in the vector as integers and store the results in an integer vector.
- R will also add a `levels` attribute to the integer, which contains a set of labels for displaying the factor values, and a `class` attribute, which contains the class `factor`

Console Terminal

R 4.4.1 · ~/

```
> car <- c("Volkswagen", "Alpine", "Mercedes", "Audi")
> car
[1] "Volkswagen" "Alpine"     "Mercedes"   "Audi"
> typeof(car)
[1] "character"
> attributes(car)
NULL
> car <- factor(car)
> car
[1] Volkswagen Alpine     Mercedes   Audi
Levels: Alpine Audi Mercedes Volkswagen
> typeof(car)
[1] "integer"
> attributes(car)
$levels
[1] "Alpine"     "Audi"       "Mercedes"   "Volkswagen"

$class
[1] "factor"
```

R Objects

Factors

Console Terminal

R 4.4.1 · ~/

```
> unclass(car)
[1] 4 1 3 2
attr(,"levels")
[1] "Alpine"     "Audi"       "Mercedes"   "Volkswagen"
> car
[1] Volkswagen Alpine     Mercedes   Audi
Levels: Alpine Audi Mercedes Volkswagen
>
```

R Objects

Factors

- You can see exactly how R is storing your factor with `unclass`

```
Console   Terminal
R 4.4.1 · ~/
> unclass(car)
[1] 4 1 3 2
attr(,"levels")
[1] "Alpine"     "Audi"       "Mercedes"   "Volkswagen"
> car
[1] Volkswagen Alpine     Mercedes   Audi
Levels: Alpine Audi Mercedes Volkswagen
>
```

R Objects

Factors

- You can see exactly how R is storing your factor with `unclass`
- R uses the levels attribute when it displays the factor. R will display each `1` as `Alpine`, the first label in the levels vector, each `2` as `Audi`, the second label etc.

Console Terminal

R 4.4.1 · ~/

```
> unclass(car)
[1] 4 1 3 2
attr(,"levels")
[1] "Alpine"     "Audi"       "Mercedes"   "Volkswagen"
> car
[1] Volkswagen Alpine     Mercedes   Audi
Levels: Alpine Audi Mercedes Volkswagen
>
```

R Objects

Factors

Console Terminal

R 4.4.1 · ~/

```
> car
[1] Volkswagen Alpine     Mercedes   Audi
Levels: Alpine Audi Mercedes Volkswagen
> typeof(car)
[1] "integer"
> car <- as.character(car)
> car
[1] "Volkswagen" "Alpine"     "Mercedes"   "Audi"
> typeof(car)
[1] "character"
>
```

R Objects

Factors

- Factors can be confusing since they look like character strings but behave like integers.

Console Terminal

R 4.4.1 · ~/

```
> car
[1] Volkswagen Alpine     Mercedes   Audi
Levels: Alpine Audi Mercedes Volkswagen
> typeof(car)
[1] "integer"
> car <- as.character(car)
> car
[1] "Volkswagen" "Alpine"     "Mercedes"   "Audi"
> typeof(car)
[1] "character"
>
```

R Objects

Factors

- Factors can be confusing since they look like character strings but behave like integers.
- R will often try to convert character strings to factors when you load and create data. In general, you will have a smoother experience if you do NOT let R make factors until you ask for them.

Console Terminal

R 4.4.1 · ~/

```
> car
[1] Volkswagen Alpine     Mercedes   Audi
Levels: Alpine Audi Mercedes Volkswagen
> typeof(car)
[1] "integer"
> car <- as.character(car)
> car
[1] "Volkswagen" "Alpine"     "Mercedes"   "Audi"
> typeof(car)
[1] "character"
>
```

R Objects

Factors

- Factors can be confusing since they look like character strings but behave like integers.
- R will often try to convert character strings to factors when you load and create data. In general, you will have a smoother experience if you do NOT let R make factors until you ask for them.
- You can convert a factor to a character string with the `as.character` function.

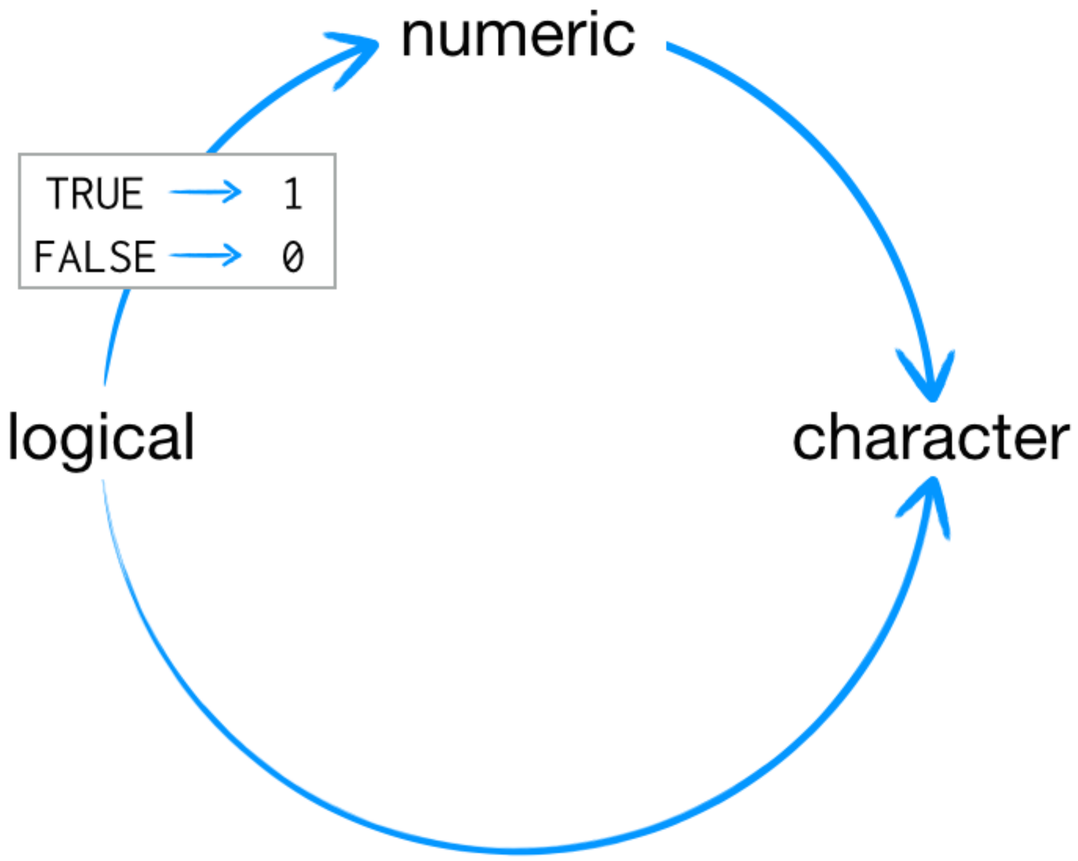
Console Terminal

R 4.4.1 · ~/

```
> car
[1] Volkswagen Alpine     Mercedes   Audi
Levels: Alpine Audi Mercedes Volkswagen
> typeof(car)
[1] "integer"
> car <- as.character(car)
> car
[1] "Volkswagen" "Alpine"     "Mercedes"   "Audi"
> typeof(car)
[1] "character"
>
```

R Objects

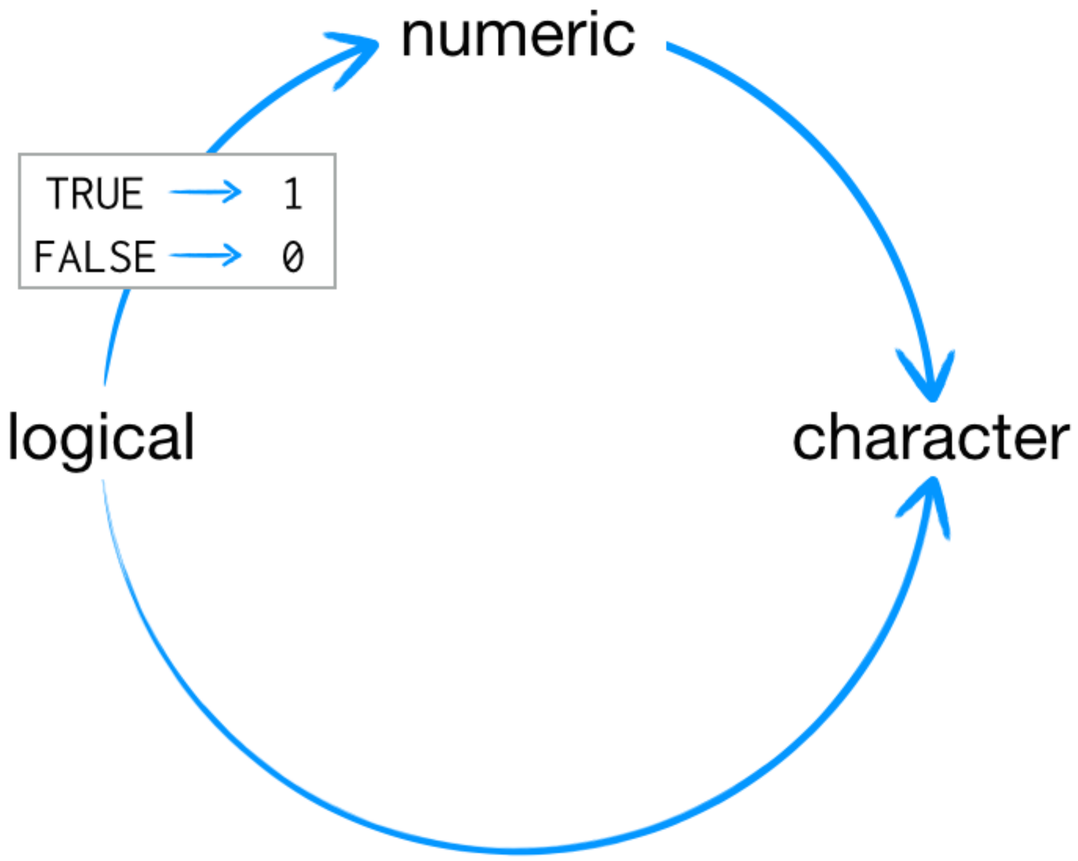
Coercion



R Objects

Coercion

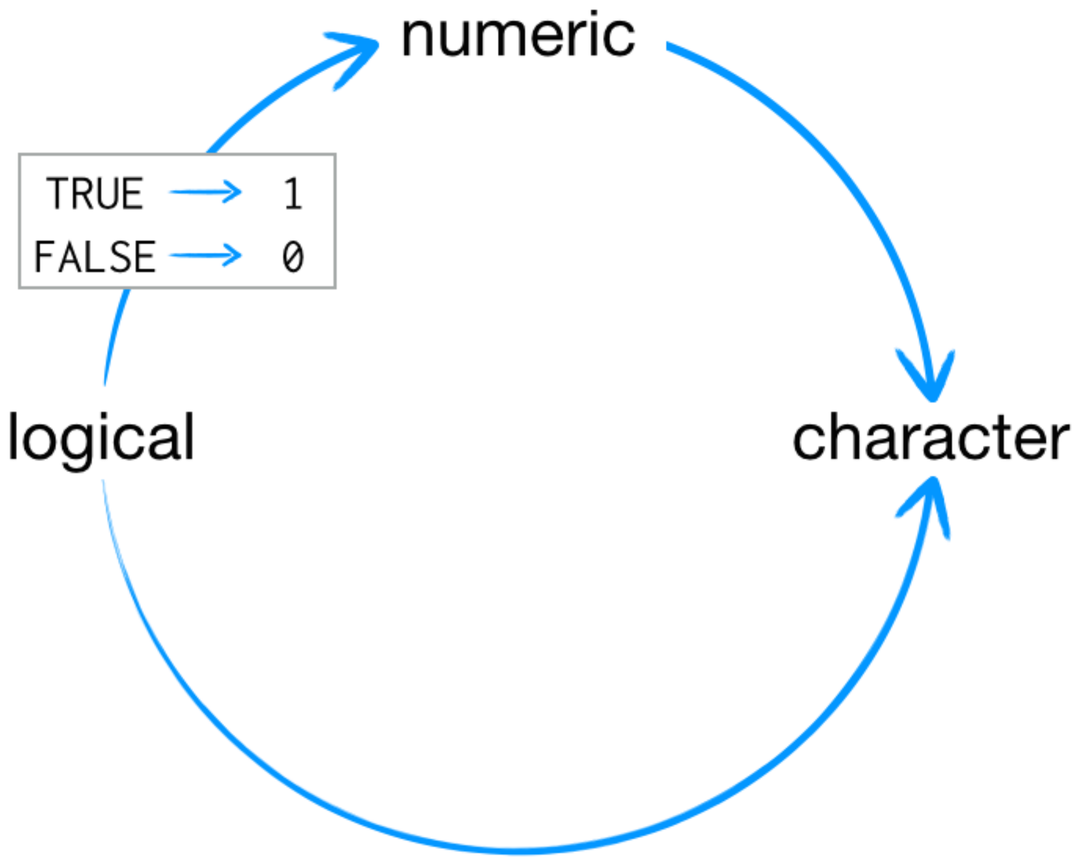
- So how does R coerce data types?



R Objects

Coercion

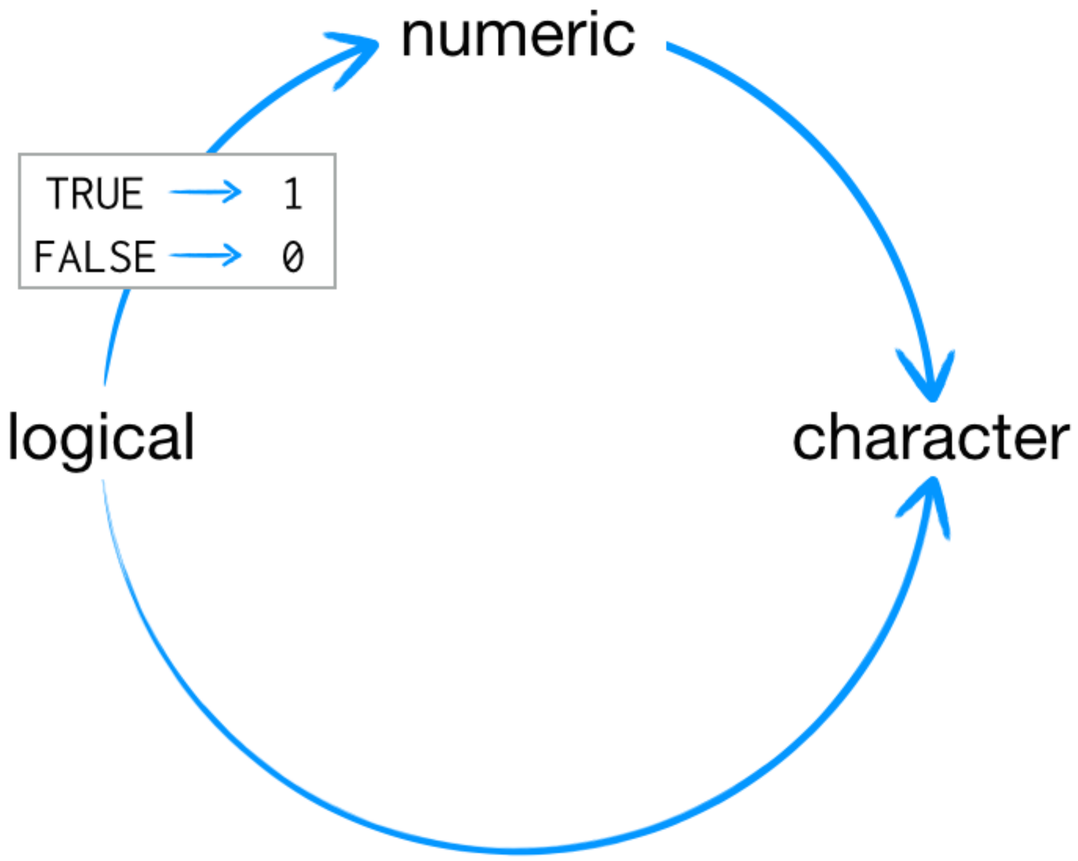
- So how does R coerce data types?
- If a character string is present in an atomic vector, R will convert everything else in the vector to character strings.



R Objects

Coercion

- So how does R coerce data types?
- If a character string is present in an atomic vector, R will convert everything else in the vector to character strings.
- If a vector only contains logicals and numbers, R will convert the logicals to numbers; every `TRUE` becomes a 1, and every `FALSE` becomes a 0.



R Objects

Coercion

Console Terminal

R 4.4.1 · ~/

```
> sum(c(TRUE, TRUE, FALSE, FALSE))
[1] 2
> sum(c(1, 1, 0, 0))
[1] 2
> mean(c(TRUE, TRUE, FALSE, FALSE))
[1] 0.5
>
```

R Objects

Coercion

- R uses the same coercion rules when you try to do math with logical values.

Console Terminal

R 4.4.1 · ~/

```
> sum(c(TRUE, TRUE, FALSE, FALSE))
[1] 2
> sum(c(1, 1, 0, 0))
[1] 2
> mean(c(TRUE, TRUE, FALSE, FALSE))
[1] 0.5
>
```

R Objects

Coercion

- R uses the same coercion rules when you try to do math with logical values.
- This means that `sum` will count the number of `TRUE`s in a logical vector (and `mean` will calculate the proportion of `TRUE`s)

Console Terminal

R 4.4.1 · ~/

```
> sum(c(TRUE, TRUE, FALSE, FALSE))
[1] 2
> sum(c(1, 1, 0, 0))
[1] 2
> mean(c(TRUE, TRUE, FALSE, FALSE))
[1] 0.5
>
```

R Objects

Coercion

R Objects

Coercion

- Many data sets contain multiple types of information.

R Objects

Coercion

- Many data sets contain multiple types of information.
- The inability of vectors, matrices, and arrays to store multiple data types seems like a major limitation.

R Objects

Coercion

- Many data sets contain multiple types of information.
- The inability of vectors, matrices, and arrays to store multiple data types seems like a major limitation.
- So why bother with them?

R Objects

Coercion

- Many data sets contain multiple types of information.
- The inability of vectors, matrices, and arrays to store multiple data types seems like a major limitation.
- So why bother with them?
- In some cases, using only a single type of data is a huge advantage. Vectors, matrices, and arrays make it very easy to do math on large sets of numbers because R knows that it can manipulate each value the same way.

R Objects

Coercion

- Many data sets contain multiple types of information.
- The inability of vectors, matrices, and arrays to store multiple data types seems like a major limitation.
- So why bother with them?
- In some cases, using only a single type of data is a huge advantage. Vectors, matrices, and arrays make it very easy to do math on large sets of numbers because R knows that it can manipulate each value the same way.
- Operations with vectors, matrices, and arrays also tend to be fast because the objects are so simple to store in memory.

R Objects

Lists

Console Terminal

R 4.4.1 · ~/

```
> list1 <- list(100:103, "R", list(TRUE, FALSE))
> list1
[[1]]
[1] 100 101 102 103

[[2]]
[1] "R"

[[3]]
[[3]][[1]]
[1] TRUE

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

R Objects

Lists

- Lists group data into a one-dimensional set.

Console Terminal

R 4.4.1 · ~/

```
> list1 <- list(100:103, "R", list(TRUE, FALSE))
> list1
[[1]]
[1] 100 101 102 103

[[2]]
[1] "R"

[[3]]
[[3]][[1]]
[1] TRUE

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

R Objects

Lists

- Lists group data into a one-dimensional set.
- However, lists do not group together individual values. They group together R objects.

Console Terminal

R 4.4.1 · ~/

```
> list1 <- list(100:103, "R", list(TRUE, FALSE))
> list1
[[1]]
[1] 100 101 102 103

[[2]]
[1] "R"

[[3]]
[[3]][[1]]
[1] TRUE

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

R Objects

Lists

- Lists group data into a one-dimensional set.
- However, lists do not group together individual values. They group together R objects.
- For example, you can make a list that contains a numeric vector of length 31 in its first element, a character vector of length 1 in its second element, and a new list of length 2 in its third element. To do this, use the `list` function.

Console Terminal

R 4.4.1 · ~/

```
> list1 <- list(100:103, "R", list(TRUE, FALSE))
> list1
[[1]]
[1] 100 101 102 103

[[2]]
[1] "R"

[[3]]
[[3]][[1]]
[1] TRUE

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

R Objects

Lists

Console Terminal

R 4.4.1 · ~/

```
> list1 <- list(100:103, "R", list(TRUE, FALSE))
> list1
[[1]]
[1] 100 101 102 103

[[2]]
[1] "R"

[[3]]
[[3]][[1]]
[1] TRUE

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

R Objects

Lists

- The double-bracketed indexes tell you which element of the list is being displayed.

Console Terminal

R 4.4.1 · ~/

```
> list1 <- list(100:103, "R", list(TRUE, FALSE))
> list1
[[1]]
[1] 100 101 102 103

[[2]]
[1] "R"

[[3]]
[[3]][[1]]
[1] TRUE

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

R Objects

Lists

- The double-bracketed indexes tell you which element of the list is being displayed.
- The single-bracket indexes tell you which subelement of an element is being displayed.

Console Terminal

R 4.4.1 · ~/

```
> list1 <- list(100:103, "R", list(TRUE, FALSE))
> list1
[[1]]
[1] 100 101 102 103

[[2]]
[1] "R"

[[3]]
[[3]][[1]]
[1] TRUE

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

R Objects

Lists

- The double-bracketed indexes tell you which element of the list is being displayed.
- The single-bracket indexes tell you which subelement of an element is being displayed.
- For example, `100` is the first subelement of the first element in the list. `"R"` is the first sub-element of the second element.

Console Terminal

R 4.4.1 · ~/

```
> list1 <- list(100:103, "R", list(TRUE, FALSE))
> list1
[[1]]
[1] 100 101 102 103

[[2]]
[1] "R"

[[3]]
[[3]][[1]]
[1] TRUE

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

R Objects

Lists

- The double-bracketed indexes tell you which element of the list is being displayed.
- The single-bracket indexes tell you which subelement of an element is being displayed.
- For example, `100` is the first subelement of the first element in the list. `"R"` is the first sub-element of the second element.
- This two-system notation arises because each element of a list can be *any* R object, including a new vector (or list) with its own indexes.

Console Terminal

R 4.4.1 · ~/

```
> list1 <- list(100:103, "R", list(TRUE, FALSE))
> list1
[[1]]
[1] 100 101 102 103

[[2]]
[1] "R"

[[3]]
[[3]][[1]]
[1] TRUE

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

R Objects

DataFrames

Intro-to-R.R | new_metadata

Filter

	genotype	celltype	replicate	samplemeans	age_in_days
sample1	Wt	typeA	1	10.266102	40
sample2	Wt	typeA	2	10.849759	32
sample3	Wt	typeA	3	9.452517	38
sample4	KO	typeA	1	15.833872	35
sample5	KO	typeA	2	15.590184	41
sample6	KO	typeA	3	15.551529	32
sample7	Wt	typeB	1	15.522219	34
sample8	Wt	typeB	2	13.808281	26
sample9	Wt	typeB	3	14.108399	28
sample10	KO	typeB	1	10.743292	28
sample11	KO	typeB	2	10.778318	30
sample12	KO	typeB	3	9.754733	32

Showing 1 to 12 of 12 entries, 5 total columns

R Objects

DataFrames

- Data frames are the two-dimensional version of a list.

Intro-to-R.R | new_metadata

Filter

	genotype	celltype	replicate	samplemeans	age_in_days
sample1	Wt	typeA	1	10.266102	40
sample2	Wt	typeA	2	10.849759	32
sample3	Wt	typeA	3	9.452517	38
sample4	KO	typeA	1	15.833872	35
sample5	KO	typeA	2	15.590184	41
sample6	KO	typeA	3	15.551529	32
sample7	Wt	typeB	1	15.522219	34
sample8	Wt	typeB	2	13.808281	26
sample9	Wt	typeB	3	14.108399	28
sample10	KO	typeB	1	10.743292	28
sample11	KO	typeB	2	10.778318	30
sample12	KO	typeB	3	9.754733	32

Showing 1 to 12 of 12 entries, 5 total columns

R Objects

DataFrames

- Data frames are the two-dimensional version of a list.
- They are far and away the most useful storage structure for data analysis, and they provide an ideal way to store an entire deck of cards.

Intro-to-R.R | new_metadata

Filter

	genotype	celltype	replicate	samplemeans	age_in_days
sample1	Wt	typeA	1	10.266102	40
sample2	Wt	typeA	2	10.849759	32
sample3	Wt	typeA	3	9.452517	38
sample4	KO	typeA	1	15.833872	35
sample5	KO	typeA	2	15.590184	41
sample6	KO	typeA	3	15.551529	32
sample7	Wt	typeB	1	15.522219	34
sample8	Wt	typeB	2	13.808281	26
sample9	Wt	typeB	3	14.108399	28
sample10	KO	typeB	1	10.743292	28
sample11	KO	typeB	2	10.778318	30
sample12	KO	typeB	3	9.754733	32

Showing 1 to 12 of 12 entries, 5 total columns

R Objects

DataFrames

- Data frames are the two-dimensional version of a list.
- They are far and away the most useful storage structure for data analysis, and they provide an ideal way to store an entire deck of cards.
- You can think of a data frame as R's equivalent to the Excel spreadsheet because it stores data in a similar format.

Intro-to-R.R | new_metadata

Filter

	genotype	celltype	replicate	samplemeans	age_in_days
sample1	Wt	typeA	1	10.266102	40
sample2	Wt	typeA	2	10.849759	32
sample3	Wt	typeA	3	9.452517	38
sample4	KO	typeA	1	15.833872	35
sample5	KO	typeA	2	15.590184	41
sample6	KO	typeA	3	15.551529	32
sample7	Wt	typeB	1	15.522219	34
sample8	Wt	typeB	2	13.808281	26
sample9	Wt	typeB	3	14.108399	28
sample10	KO	typeB	1	10.743292	28
sample11	KO	typeB	2	10.778318	30
sample12	KO	typeB	3	9.754733	32

Showing 1 to 12 of 12 entries, 5 total columns

R Objects

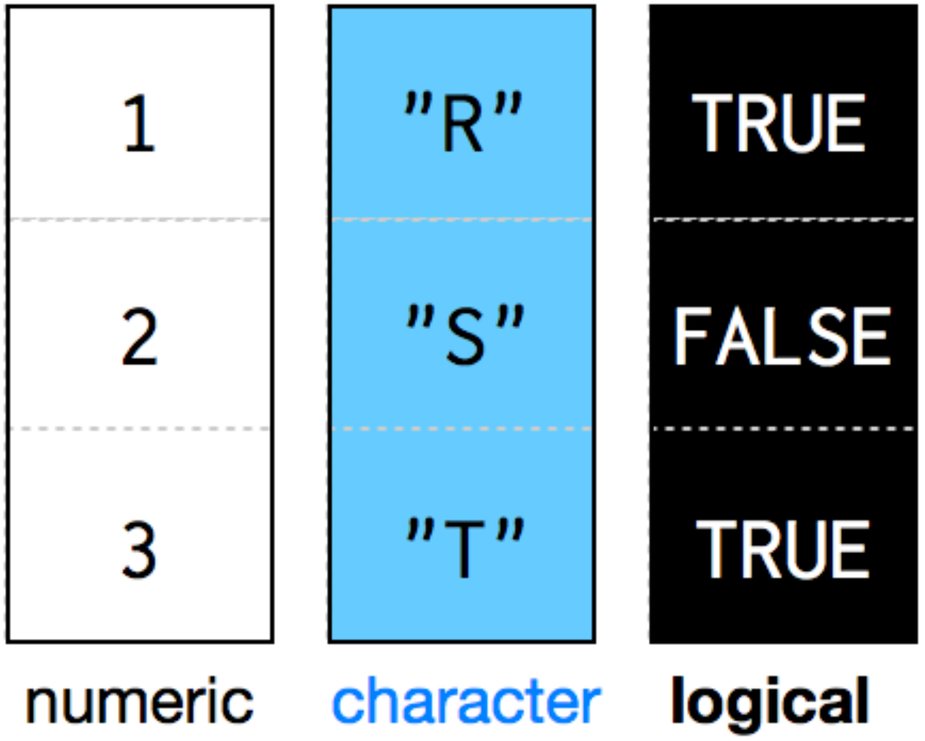
DataFrames

numeric	character	logical
1	"R"	TRUE
2	"S"	FALSE
3	"T"	TRUE

R Objects

DataFrames

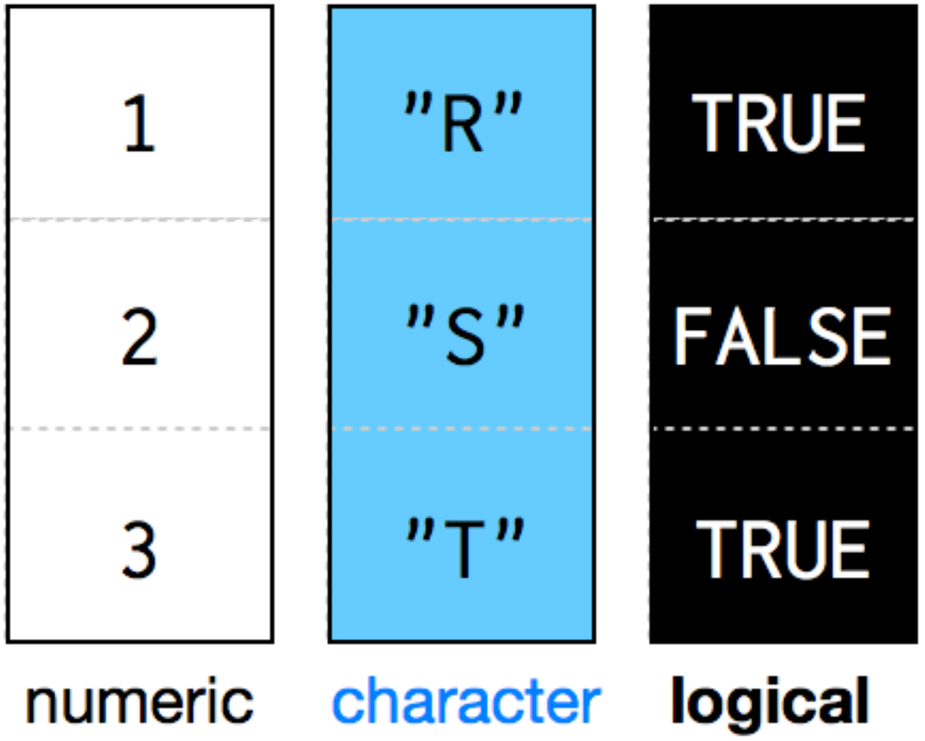
- Data frames group vectors together into a two-dimensional table. Each vector becomes a column in the table.



R Objects

DataFrames

- Data frames group vectors together into a two-dimensional table. Each vector becomes a column in the table.
- As a result, each column of a data frame can contain a different type of data; but within a column, every cell must be the same type of data.



R Objects

DataFrames

Console Terminal

R 4.4.1 · ~/

```
> df <- data.frame(face = c("ace", "two", "six"),
+                  suit = c("clubs", "clubs", "clubs"),
+                  value = c(1, 2, 3))
> df
  face  suit value
1  ace clubs     1
2  two clubs     2
3  six clubs     3
>
```

R Objects

DataFrames

- Creating a data frame by hand takes a lot of typing, but you can do it with the `data.frame` function.

Console Terminal

R 4.4.1 · ~/

```
> df <- data.frame(face = c("ace", "two", "six"),
+                  suit = c("clubs", "clubs", "clubs"),
+                  value = c(1, 2, 3))
> df
  face  suit value
1  ace clubs     1
2  two clubs     2
3  six clubs     3
>
```

R Objects

DataFrames

- Creating a data frame by hand takes a lot of typing, but you can do it with the `data.frame` function.
- Give `data.frame` any number of vectors, each separated with a comma.

Console Terminal

R 4.4.1 · ~/

```
> df <- data.frame(face = c("ace", "two", "six"),
+                  suit = c("clubs", "clubs", "clubs"),
+                  value = c(1, 2, 3))
> df
  face  suit value
1  ace clubs     1
2  two clubs     2
3  six clubs     3
>
```

R Objects

DataFrames

- Creating a data frame by hand takes a lot of typing, but you can do it with the `data.frame` function.
- Give `data.frame` any number of vectors, each separated with a comma.
- Each vector should be set equal to a name that describes the vector. `data.frame` will turn each vector into a column of the new data frame.

Console Terminal

R 4.4.1 · ~/

```
> df <- data.frame(face = c("ace", "two", "six"),
+                  suit = c("clubs", "clubs", "clubs"),
+                  value = c(1, 2, 3))
> df
  face  suit value
1  ace clubs     1
2  two clubs     2
3  six clubs     3
>
```

R Objects

DataFrames

Console Terminal

R 4.4.1 · ~/

```
> df <- data.frame(face = c("ace", "two", "six"),
+                  suit = c("clubs", "clubs", "clubs"),
+                  value = c(1, 2, 3))
> df
  face  suit value
1  ace clubs     1
2  two clubs     2
3  six clubs     3
>
```

R Objects

DataFrames

- You'll need to make sure that each vector is the same length.

Console Terminal

R 4.4.1 · ~/

```
> df <- data.frame(face = c("ace", "two", "six"),
+                  suit = c("clubs", "clubs", "clubs"),
+                  value = c(1, 2, 3))
> df
  face  suit value
1  ace clubs     1
2  two clubs     2
3  six clubs     3
>
```

R Objects

DataFrames

- You'll need to make sure that each vector is the same length.
- In the previous code, I named the arguments in `data.frame` `face`, `suit`, and `value`, but you can name the arguments whatever you like.

Console Terminal

R 4.4.1 · ~/

```
> df <- data.frame(face = c("ace", "two", "six"),
+                  suit = c("clubs", "clubs", "clubs"),
+                  value = c(1, 2, 3))
> df
  face  suit value
1  ace clubs     1
2  two clubs     2
3  six clubs     3
>
```

R Objects

DataFrames

- You'll need to make sure that each vector is the same length.
- In the previous code, I named the arguments in `data.frame` `face`, `suit`, and `value`, but you can name the arguments whatever you like.
- `data.frame` will use your argument names to label the columns of the data frame.

Console Terminal

R 4.4.1 · ~/

```
> df <- data.frame(face = c("ace", "two", "six"),
+                  suit = c("clubs", "clubs", "clubs"),
+                  value = c(1, 2, 3))
> df
  face  suit value
1  ace clubs     1
2  two clubs     2
3  six clubs     3
>
```

R Objects

DataFrames

Console Terminal

R 4.4.1 · ~/

```
> df
  face  suit value
1  ace clubs     1
2  two clubs     2
3  six clubs     3
> typeof(df)
[1] "list"
> class(df)
[1] "data.frame"
> str(df)
'data.frame':   3 obs. of  3 variables:
 $ face : chr  "ace" "two" "six"
 $ suit : chr  "clubs" "clubs" "clubs"
 $ value: num  1 2 3
>
```

R Objects

DataFrames

- If you look at the type of a data frame, you will see that it is a list.

Console Terminal

R 4.4.1 · ~/

```
> df
  face  suit value
1  ace clubs     1
2  two clubs     2
3  six clubs     3
> typeof(df)
[1] "list"
> class(df)
[1] "data.frame"
> str(df)
'data.frame':   3 obs. of  3 variables:
 $ face : chr  "ace" "two" "six"
 $ suit : chr  "clubs" "clubs" "clubs"
 $ value: num  1 2 3
>
```

R Objects

DataFrames

- If you look at the type of a data frame, you will see that it is a list.
- In fact, each data frame is a list with class `data.frame`.

Console Terminal

R 4.4.1 · ~/

```
> df
  face  suit value
1  ace clubs     1
2  two clubs     2
3  six clubs     3
> typeof(df)
[1] "list"
> class(df)
[1] "data.frame"
> str(df)
'data.frame':   3 obs. of  3 variables:
 $ face : chr  "ace" "two" "six"
 $ suit : chr  "clubs" "clubs" "clubs"
 $ value: num  1 2 3
>
```

R Objects

DataFrames

- If you look at the type of a data frame, you will see that it is a list.
- In fact, each data frame is a list with class `data.frame`.
- You can see what types of objects are grouped together by a list with the `str` function.

Console Terminal

R 4.4.1 · ~/

```
> df
  face  suit value
1  ace clubs     1
2  two clubs     2
3  six clubs     3
> typeof(df)
[1] "list"
> class(df)
[1] "data.frame"
> str(df)
'data.frame':   3 obs. of  3 variables:
 $ face : chr  "ace" "two" "six"
 $ suit : chr  "clubs" "clubs" "clubs"
 $ value: num  1 2 3
>
```

R Objects

DataFrames

Console Terminal

R 4.4.1 · ~/

```
> deck <- data.frame(
+     face = c("king", "queen", "jack", "ten", "nine", "eight", "seven", "six",
+              "five", "four", "three", "two", "ace", "king", "queen", "jack", "ten",
+              "nine", "eight", "seven", "six", "five", "four", "three", "two", "ace",
+              "king", "queen", "jack", "ten", "nine", "eight", "seven", "six", "five",
+              "four", "three", "two", "ace", "king", "queen", "jack", "ten", "nine",
+              "eight", "seven", "six", "five", "four", "three", "two", "ace"),
+     suit = c("spades", "spades", "spades", "spades", "spades", "spades",
+              "spades", "spades", "spades", "spades", "spades", "spades", "spades",
+              "clubs", "clubs", "clubs", "clubs", "clubs", "clubs", "clubs", "clubs",
+              "clubs", "clubs", "clubs", "clubs", "clubs", "diamonds", "diamonds",
+              "diamonds", "diamonds", "diamonds", "diamonds", "diamonds", "diamonds",
+              "diamonds", "diamonds", "diamonds", "diamonds", "diamonds", "hearts",
+              "hearts", "hearts", "hearts", "hearts", "hearts", "hearts", "hearts",
+              "hearts", "hearts", "hearts", "hearts", "hearts"),
+     value = c(13, 12, 11, 10, 9, 8, 7, 6, 5, 4, 3, 2, 1, 13, 12, 11, 10, 9, 8,
+               7, 6, 5, 4, 3, 2, 1, 13, 12, 11, 10, 9, 8, 7, 6, 5, 4, 3, 2, 1, 13, 12, 11,
+               10, 9, 8, 7, 6, 5, 4, 3, 2, 1)
+ )
```

R Objects

DataFrames

- A data frame is a great way to build an entire deck of cards.

Console Terminal

R 4.4.1 · ~/

```
> deck <- data.frame(
+     face = c("king", "queen", "jack", "ten", "nine", "eight", "seven", "six",
+              "five", "four", "three", "two", "ace", "king", "queen", "jack", "ten",
+              "nine", "eight", "seven", "six", "five", "four", "three", "two", "ace",
+              "king", "queen", "jack", "ten", "nine", "eight", "seven", "six", "five",
+              "four", "three", "two", "ace", "king", "queen", "jack", "ten", "nine",
+              "eight", "seven", "six", "five", "four", "three", "two", "ace"),
+     suit = c("spades", "spades", "spades", "spades", "spades", "spades",
+              "spades", "spades", "spades", "spades", "spades", "spades", "spades",
+              "clubs", "clubs", "clubs", "clubs", "clubs", "clubs", "clubs", "clubs",
+              "clubs", "clubs", "clubs", "clubs", "clubs", "diamonds", "diamonds",
+              "diamonds", "diamonds", "diamonds", "diamonds", "diamonds", "diamonds",
+              "diamonds", "diamonds", "diamonds", "diamonds", "diamonds", "hearts",
+              "hearts", "hearts", "hearts", "hearts", "hearts", "hearts", "hearts",
+              "hearts", "hearts", "hearts", "hearts", "hearts"),
+     value = c(13, 12, 11, 10, 9, 8, 7, 6, 5, 4, 3, 2, 1, 13, 12, 11, 10, 9, 8,
+               7, 6, 5, 4, 3, 2, 1, 13, 12, 11, 10, 9, 8, 7, 6, 5, 4, 3, 2, 1, 13, 12, 11,
+               10, 9, 8, 7, 6, 5, 4, 3, 2, 1)
+ )
```

R Objects

DataFrames

- A data frame is a great way to build an entire deck of cards.
- You can make each row in the data frame a playing card, and each column a type of value—each with its own appropriate data type.

Console Terminal

R 4.4.1 · ~/

```
> deck <- data.frame(
+     face = c("king", "queen", "jack", "ten", "nine", "eight", "seven", "six",
+              "five", "four", "three", "two", "ace", "king", "queen", "jack", "ten",
+              "nine", "eight", "seven", "six", "five", "four", "three", "two", "ace",
+              "king", "queen", "jack", "ten", "nine", "eight", "seven", "six", "five",
+              "four", "three", "two", "ace", "king", "queen", "jack", "ten", "nine",
+              "eight", "seven", "six", "five", "four", "three", "two", "ace"),
+     suit = c("spades", "spades", "spades", "spades", "spades", "spades",
+              "spades", "spades", "spades", "spades", "spades", "spades", "spades",
+              "clubs", "clubs", "clubs", "clubs", "clubs", "clubs", "clubs", "clubs",
+              "clubs", "clubs", "clubs", "clubs", "clubs", "diamonds", "diamonds",
+              "diamonds", "diamonds", "diamonds", "diamonds", "diamonds", "diamonds",
+              "diamonds", "diamonds", "diamonds", "diamonds", "diamonds", "hearts",
+              "hearts", "hearts", "hearts", "hearts", "hearts", "hearts", "hearts",
+              "hearts", "hearts", "hearts", "hearts", "hearts"),
+     value = c(13, 12, 11, 10, 9, 8, 7, 6, 5, 4, 3, 2, 1, 13, 12, 11, 10, 9, 8,
+               7, 6, 5, 4, 3, 2, 1, 13, 12, 11, 10, 9, 8, 7, 6, 5, 4, 3, 2, 1, 13, 12, 11,
+               10, 9, 8, 7, 6, 5, 4, 3, 2, 1)
+ )
```

R Objects

DataFrames

- A data frame is a great way to build an entire deck of cards.
- You can make each row in the data frame a playing card, and each column a type of value—each with its own appropriate data type.
- You could create this data frame with `data.frame`, but look at the typing involved! You need to write three vectors, each with 52 elements.

Console Terminal

R 4.4.1 · ~/

```
> deck <- data.frame(
+     face = c("king", "queen", "jack", "ten", "nine", "eight", "seven", "six",
+              "five", "four", "three", "two", "ace", "king", "queen", "jack", "ten",
+              "nine", "eight", "seven", "six", "five", "four", "three", "two", "ace",
+              "king", "queen", "jack", "ten", "nine", "eight", "seven", "six", "five",
+              "four", "three", "two", "ace", "king", "queen", "jack", "ten", "nine",
+              "eight", "seven", "six", "five", "four", "three", "two", "ace"),
+     suit = c("spades", "spades", "spades", "spades", "spades", "spades",
+              "spades", "spades", "spades", "spades", "spades", "spades", "spades",
+              "clubs", "clubs", "clubs", "clubs", "clubs", "clubs", "clubs", "clubs",
+              "clubs", "clubs", "clubs", "clubs", "clubs", "diamonds", "diamonds",
+              "diamonds", "diamonds", "diamonds", "diamonds", "diamonds", "diamonds",
+              "diamonds", "diamonds", "diamonds", "diamonds", "diamonds", "hearts",
+              "hearts", "hearts", "hearts", "hearts", "hearts", "hearts", "hearts",
+              "hearts", "hearts", "hearts", "hearts", "hearts"),
+     value = c(13, 12, 11, 10, 9, 8, 7, 6, 5, 4, 3, 2, 1, 13, 12, 11, 10, 9, 8,
+               7, 6, 5, 4, 3, 2, 1, 13, 12, 11, 10, 9, 8, 7, 6, 5, 4, 3, 2, 1, 13, 12, 11,
+               10, 9, 8, 7, 6, 5, 4, 3, 2, 1)
+ )
```

R Objects

Loading data

Console Terminal

R 4.4.1 · ~/

```
> deck <- data.frame(
+     face = c("king", "queen", "jack", "ten", "nine", "eight", "seven", "six",
+              "five", "four", "three", "two", "ace", "king", "queen", "jack", "ten",
+              "nine", "eight", "seven", "six", "five", "four", "three", "two", "ace",
+              "king", "queen", "jack", "ten", "nine", "eight", "seven", "six", "five",
+              "four", "three", "two", "ace", "king", "queen", "jack", "ten", "nine",
+              "eight", "seven", "six", "five", "four", "three", "two", "ace"),
+     suit = c("spades", "spades", "spades", "spades", "spades", "spades",
+              "spades", "spades", "spades", "spades", "spades", "spades", "spades",
+              "clubs", "clubs", "clubs", "clubs", "clubs", "clubs", "clubs", "clubs",
+              "clubs", "clubs", "clubs", "clubs", "clubs", "diamonds", "diamonds",
+              "diamonds", "diamonds", "diamonds", "diamonds", "diamonds", "diamonds",
+              "diamonds", "diamonds", "diamonds", "diamonds", "diamonds", "hearts",
+              "hearts", "hearts", "hearts", "hearts", "hearts", "hearts", "hearts",
+              "hearts", "hearts", "hearts", "hearts", "hearts"),
+     value = c(13, 12, 11, 10, 9, 8, 7, 6, 5, 4, 3, 2, 1, 13, 12, 11, 10, 9, 8,
+               7, 6, 5, 4, 3, 2, 1, 13, 12, 11, 10, 9, 8, 7, 6, 5, 4, 3, 2, 1, 13, 12, 11,
+               10, 9, 8, 7, 6, 5, 4, 3, 2, 1)
+ )
```

R Objects

Loading data

- You should avoid typing large data sets in by hand whenever possible.

Console Terminal

R 4.4.1 · ~/

```
> deck <- data.frame(
+     face = c("king", "queen", "jack", "ten", "nine", "eight", "seven", "six",
+              "five", "four", "three", "two", "ace", "king", "queen", "jack", "ten",
+              "nine", "eight", "seven", "six", "five", "four", "three", "two", "ace",
+              "king", "queen", "jack", "ten", "nine", "eight", "seven", "six", "five",
+              "four", "three", "two", "ace", "king", "queen", "jack", "ten", "nine",
+              "eight", "seven", "six", "five", "four", "three", "two", "ace"),
+     suit = c("spades", "spades", "spades", "spades", "spades", "spades",
+              "spades", "spades", "spades", "spades", "spades", "spades", "spades",
+              "clubs", "clubs", "clubs", "clubs", "clubs", "clubs", "clubs", "clubs",
+              "clubs", "clubs", "clubs", "clubs", "clubs", "diamonds", "diamonds",
+              "diamonds", "diamonds", "diamonds", "diamonds", "diamonds", "diamonds",
+              "diamonds", "diamonds", "diamonds", "diamonds", "diamonds", "hearts",
+              "hearts", "hearts", "hearts", "hearts", "hearts", "hearts", "hearts",
+              "hearts", "hearts", "hearts", "hearts", "hearts"),
+     value = c(13, 12, 11, 10, 9, 8, 7, 6, 5, 4, 3, 2, 1, 13, 12, 11, 10, 9, 8,
+               7, 6, 5, 4, 3, 2, 1, 13, 12, 11, 10, 9, 8, 7, 6, 5, 4, 3, 2, 1, 13, 12, 11,
+               10, 9, 8, 7, 6, 5, 4, 3, 2, 1)
+ )
```

R Objects

Loading data

- You should avoid typing large data sets in by hand whenever possible.
- Typing invites typos and errors.

Console Terminal

R 4.4.1 · ~/

```
> deck <- data.frame(
+     face = c("king", "queen", "jack", "ten", "nine", "eight", "seven", "six",
+              "five", "four", "three", "two", "ace", "king", "queen", "jack", "ten",
+              "nine", "eight", "seven", "six", "five", "four", "three", "two", "ace",
+              "king", "queen", "jack", "ten", "nine", "eight", "seven", "six", "five",
+              "four", "three", "two", "ace", "king", "queen", "jack", "ten", "nine",
+              "eight", "seven", "six", "five", "four", "three", "two", "ace"),
+     suit = c("spades", "spades", "spades", "spades", "spades", "spades",
+              "spades", "spades", "spades", "spades", "spades", "spades", "spades",
+              "clubs", "clubs", "clubs", "clubs", "clubs", "clubs", "clubs", "clubs",
+              "clubs", "clubs", "clubs", "clubs", "clubs", "diamonds", "diamonds",
+              "diamonds", "diamonds", "diamonds", "diamonds", "diamonds", "diamonds",
+              "diamonds", "diamonds", "diamonds", "diamonds", "diamonds", "hearts",
+              "hearts", "hearts", "hearts", "hearts", "hearts", "hearts", "hearts",
+              "hearts", "hearts", "hearts", "hearts", "hearts"),
+     value = c(13, 12, 11, 10, 9, 8, 7, 6, 5, 4, 3, 2, 1, 13, 12, 11, 10, 9, 8,
+               7, 6, 5, 4, 3, 2, 1, 13, 12, 11, 10, 9, 8, 7, 6, 5, 4, 3, 2, 1, 13, 12, 11,
+               10, 9, 8, 7, 6, 5, 4, 3, 2, 1)
+ )
```

R Objects

Loading data

- You should avoid typing large data sets in by hand whenever possible.
- Typing invites typos and errors.
- It is always better to acquire large data sets as a computer file.

Console Terminal

R 4.4.1 · ~/

```
> deck <- data.frame(
+     face = c("king", "queen", "jack", "ten", "nine", "eight", "seven", "six",
+              "five", "four", "three", "two", "ace", "king", "queen", "jack", "ten",
+              "nine", "eight", "seven", "six", "five", "four", "three", "two", "ace",
+              "king", "queen", "jack", "ten", "nine", "eight", "seven", "six", "five",
+              "four", "three", "two", "ace", "king", "queen", "jack", "ten", "nine",
+              "eight", "seven", "six", "five", "four", "three", "two", "ace"),
+     suit = c("spades", "spades", "spades", "spades", "spades", "spades",
+              "spades", "spades", "spades", "spades", "spades", "spades", "spades",
+              "clubs", "clubs", "clubs", "clubs", "clubs", "clubs", "clubs", "clubs",
+              "clubs", "clubs", "clubs", "clubs", "clubs", "diamonds", "diamonds",
+              "diamonds", "diamonds", "diamonds", "diamonds", "diamonds", "diamonds",
+              "diamonds", "diamonds", "diamonds", "diamonds", "diamonds", "hearts",
+              "hearts", "hearts", "hearts", "hearts", "hearts", "hearts", "hearts",
+              "hearts", "hearts", "hearts", "hearts", "hearts"),
+     value = c(13, 12, 11, 10, 9, 8, 7, 6, 5, 4, 3, 2, 1, 13, 12, 11, 10, 9, 8,
+               7, 6, 5, 4, 3, 2, 1, 13, 12, 11, 10, 9, 8, 7, 6, 5, 4, 3, 2, 1, 13, 12, 11,
+               10, 9, 8, 7, 6, 5, 4, 3, 2, 1)
+ )
```

R Objects

Loading data

- You should avoid typing large data sets in by hand whenever possible.
- Typing invites typos and errors.
- It is always better to acquire large data sets as a computer file.
- You can then ask R to read the file and store the contents as an object.

Console Terminal

R 4.4.1 · ~/

```
> deck <- data.frame(
+     face = c("king", "queen", "jack", "ten", "nine", "eight", "seven", "six",
+              "five", "four", "three", "two", "ace", "king", "queen", "jack", "ten",
+              "nine", "eight", "seven", "six", "five", "four", "three", "two", "ace",
+              "king", "queen", "jack", "ten", "nine", "eight", "seven", "six", "five",
+              "four", "three", "two", "ace", "king", "queen", "jack", "ten", "nine",
+              "eight", "seven", "six", "five", "four", "three", "two", "ace"),
+     suit = c("spades", "spades", "spades", "spades", "spades", "spades",
+              "spades", "spades", "spades", "spades", "spades", "spades", "spades",
+              "clubs", "clubs", "clubs", "clubs", "clubs", "clubs", "clubs", "clubs",
+              "clubs", "clubs", "clubs", "clubs", "clubs", "diamonds", "diamonds",
+              "diamonds", "diamonds", "diamonds", "diamonds", "diamonds", "diamonds",
+              "diamonds", "diamonds", "diamonds", "diamonds", "diamonds", "hearts",
+              "hearts", "hearts", "hearts", "hearts", "hearts", "hearts", "hearts",
+              "hearts", "hearts", "hearts", "hearts", "hearts"),
+     value = c(13, 12, 11, 10, 9, 8, 7, 6, 5, 4, 3, 2, 1, 13, 12, 11, 10, 9, 8,
+               7, 6, 5, 4, 3, 2, 1, 13, 12, 11, 10, 9, 8, 7, 6, 5, 4, 3, 2, 1, 13, 12, 11,
+               10, 9, 8, 7, 6, 5, 4, 3, 2, 1)
+ )
```

R Objects

Loading data

- You should avoid typing large data sets in by hand whenever possible.
- Typing invites typos and errors.
- It is always better to acquire large data sets as a computer file.
- You can then ask R to read the file and store the contents as an object.
- I'll send you a file that contains a data frame of playing-card information, so don't worry about typing in the code.

Console Terminal

R 4.4.1 · ~/

```
> deck <- data.frame(
+     face = c("king", "queen", "jack", "ten", "nine", "eight", "seven", "six",
+              "five", "four", "three", "two", "ace", "king", "queen", "jack", "ten",
+              "nine", "eight", "seven", "six", "five", "four", "three", "two", "ace",
+              "king", "queen", "jack", "ten", "nine", "eight", "seven", "six", "five",
+              "four", "three", "two", "ace", "king", "queen", "jack", "ten", "nine",
+              "eight", "seven", "six", "five", "four", "three", "two", "ace"),
+     suit = c("spades", "spades", "spades", "spades", "spades", "spades",
+              "spades", "spades", "spades", "spades", "spades", "spades", "spades",
+              "clubs", "clubs", "clubs", "clubs", "clubs", "clubs", "clubs", "clubs",
+              "clubs", "clubs", "clubs", "clubs", "clubs", "diamonds", "diamonds",
+              "diamonds", "diamonds", "diamonds", "diamonds", "diamonds", "diamonds",
+              "diamonds", "diamonds", "diamonds", "diamonds", "diamonds", "hearts",
+              "hearts", "hearts", "hearts", "hearts", "hearts", "hearts", "hearts",
+              "hearts", "hearts", "hearts", "hearts", "hearts"),
+     value = c(13, 12, 11, 10, 9, 8, 7, 6, 5, 4, 3, 2, 1, 13, 12, 11, 10, 9, 8,
+               7, 6, 5, 4, 3, 2, 1, 13, 12, 11, 10, 9, 8, 7, 6, 5, 4, 3, 2, 1, 13, 12, 11,
+               10, 9, 8, 7, 6, 5, 4, 3, 2, 1)
+ )
```

R Objects

Loading data

deck

face	suit	value
king	spades	13
queen	spades	12
jack	spades	11
ten	spades	10
nine	spades	9
eight	spades	8
seven	spades	7
six	spades	6
five	spades	5

R Objects

Loading data

- You can load the `deck` data frame from the file Data on the page course.

deck

face	suit	value
king	spades	13
queen	spades	12
jack	spades	11
ten	spades	10
nine	spades	9
eight	spades	8
seven	spades	7
six	spades	6
five	spades	5

R Objects

Loading data

- You can load the `deck` data frame from the file Data on the page course.
- *deck.csv* is a comma-separated values file, or CSV for short.

deck

face	suit	value
king	spades	13
queen	spades	12
jack	spades	11
ten	spades	10
nine	spades	9
eight	spades	8
seven	spades	7
six	spades	6
five	spades	5

R Objects

Loading data

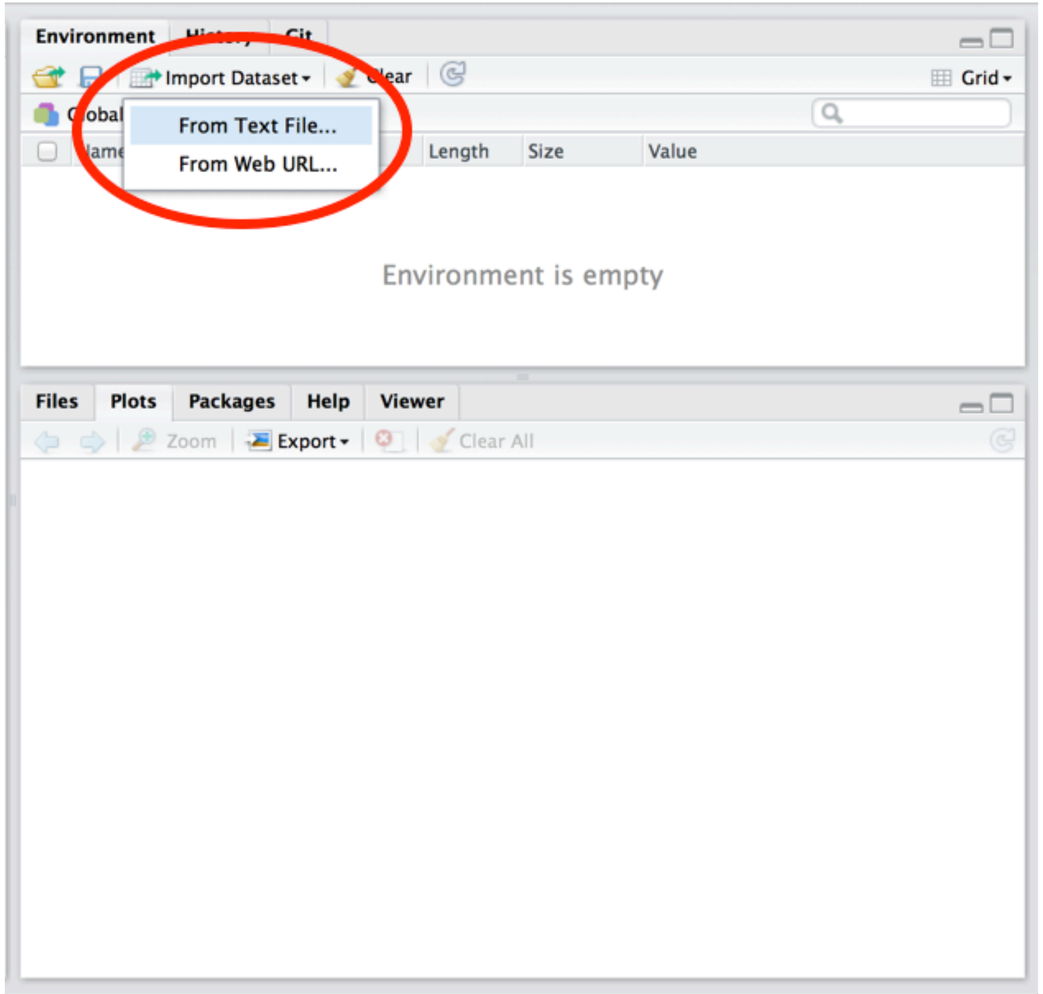
- You can load the `deck` data frame from the file Data on the page course.
- *deck.csv* is a comma-separated values file, or CSV for short.
- CSVs are plain-text files, which means you can open them in a text editor.

deck

face	suit	value
king	spades	13
queen	spades	12
jack	spades	11
ten	spades	10
nine	spades	9
eight	spades	8
seven	spades	7
six	spades	6
five	spades	5

R Objects

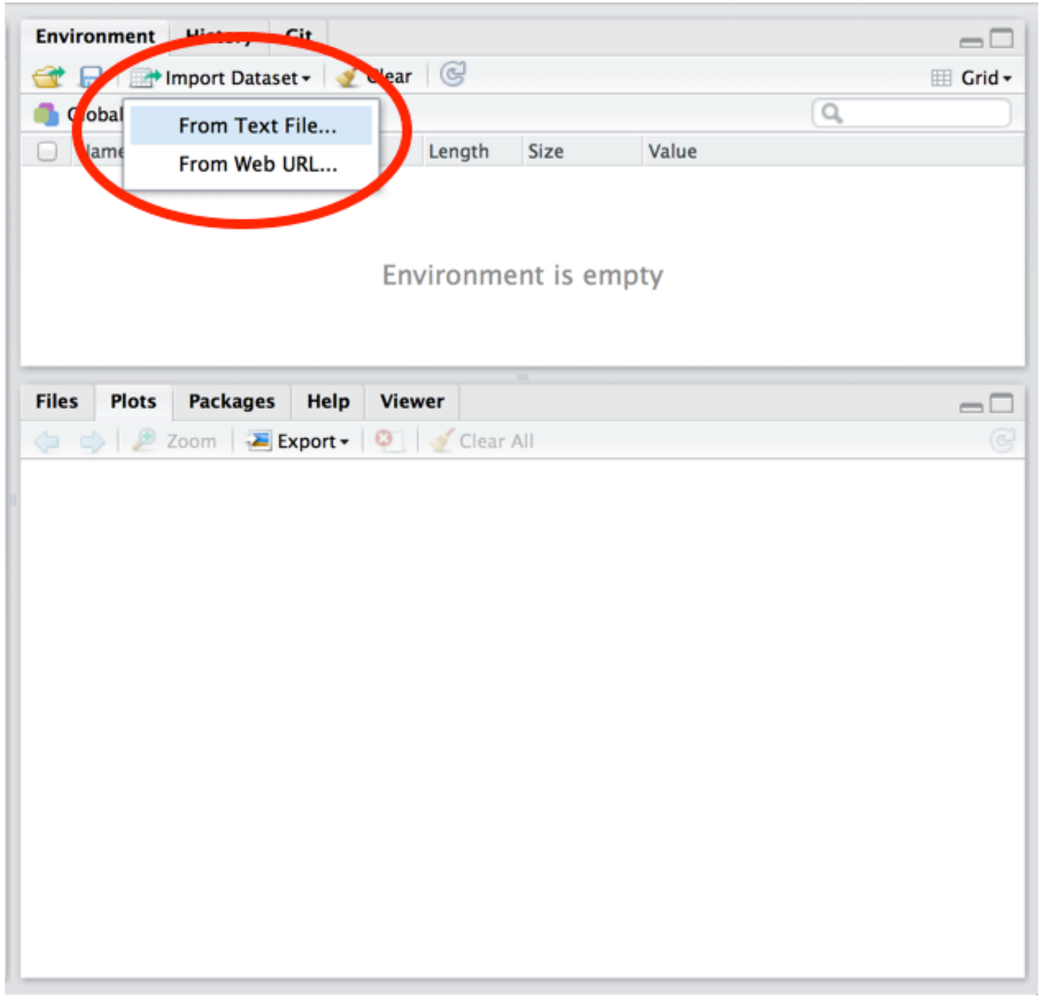
Loading data



R Objects

Loading data

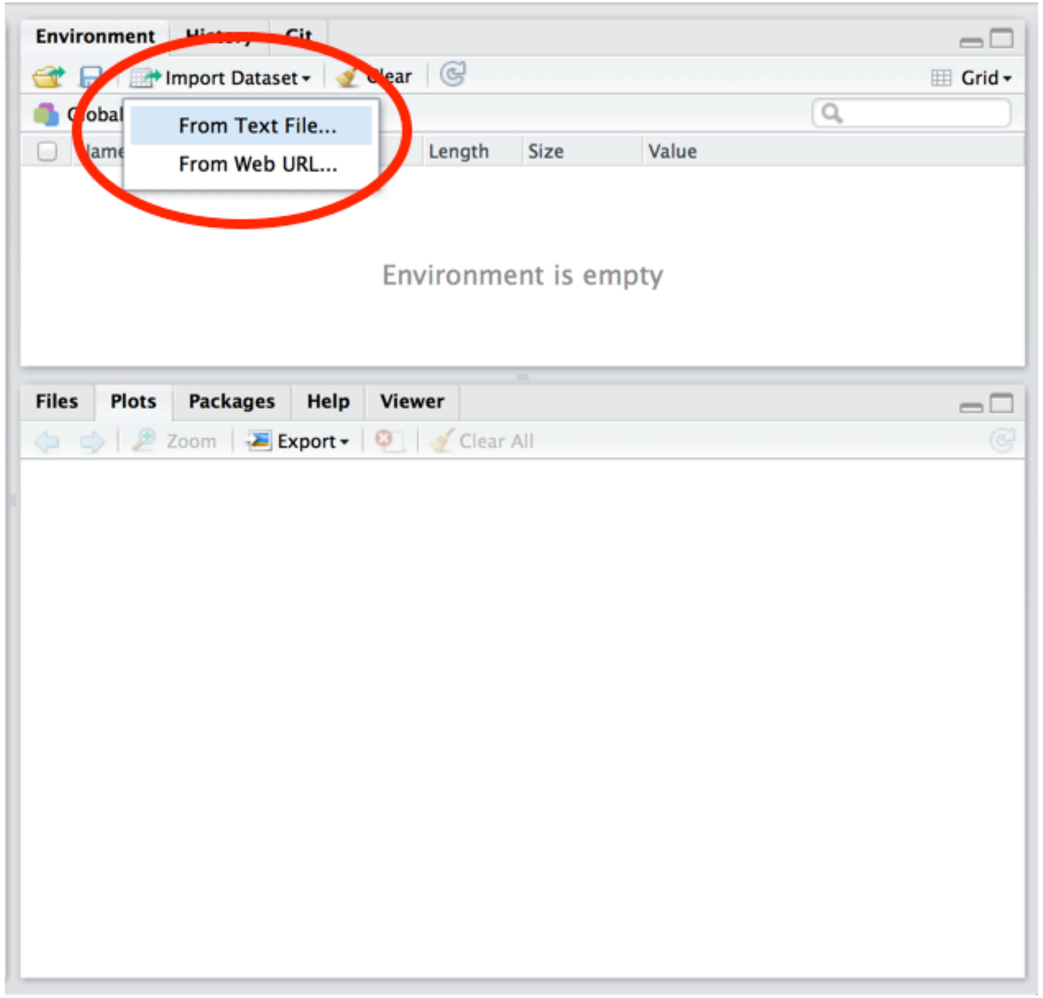
- To load a plain-text file into R, click the Import Dataset icon in RStudio



R Objects

Loading data

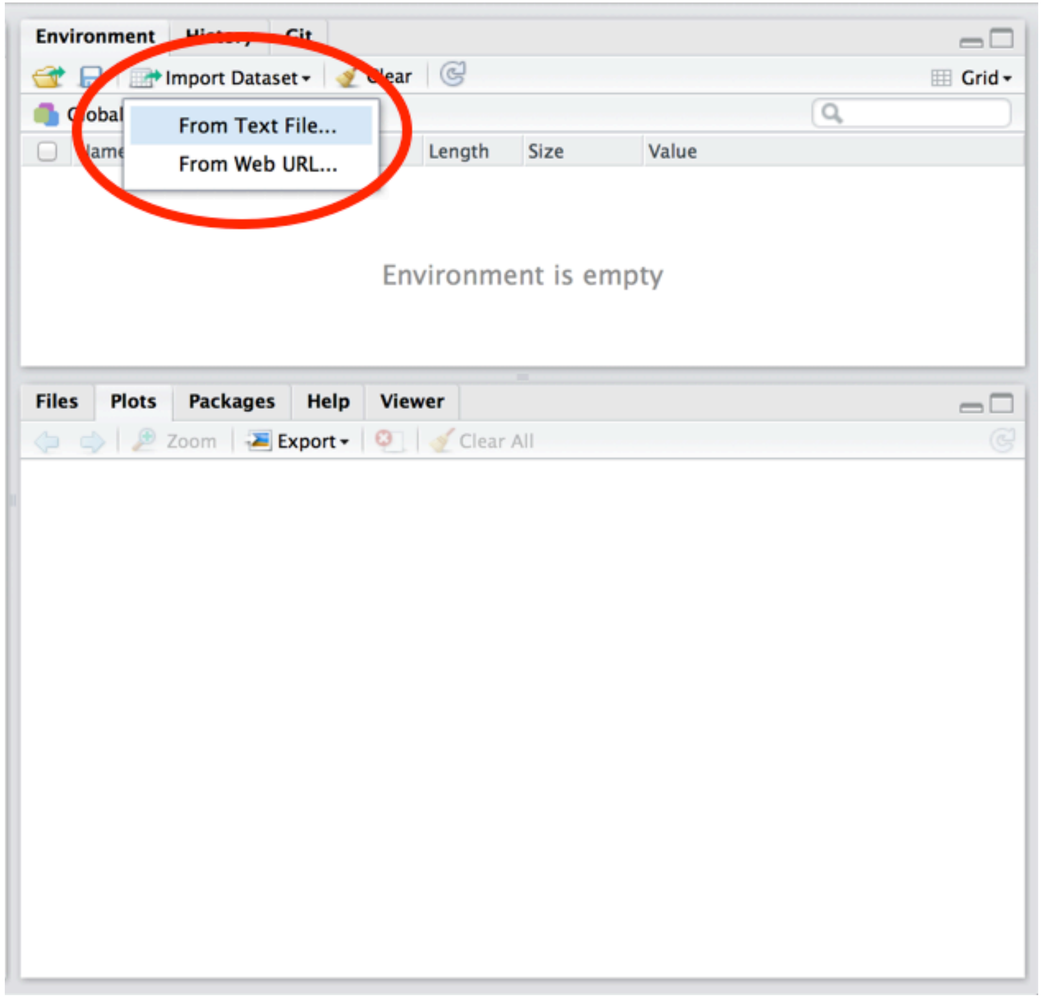
- To load a plain-text file into R, click the Import Dataset icon in RStudio
- RStudio will ask you to select the file you want to import, then it will open a wizard to help you import the data



R Objects

Loading data

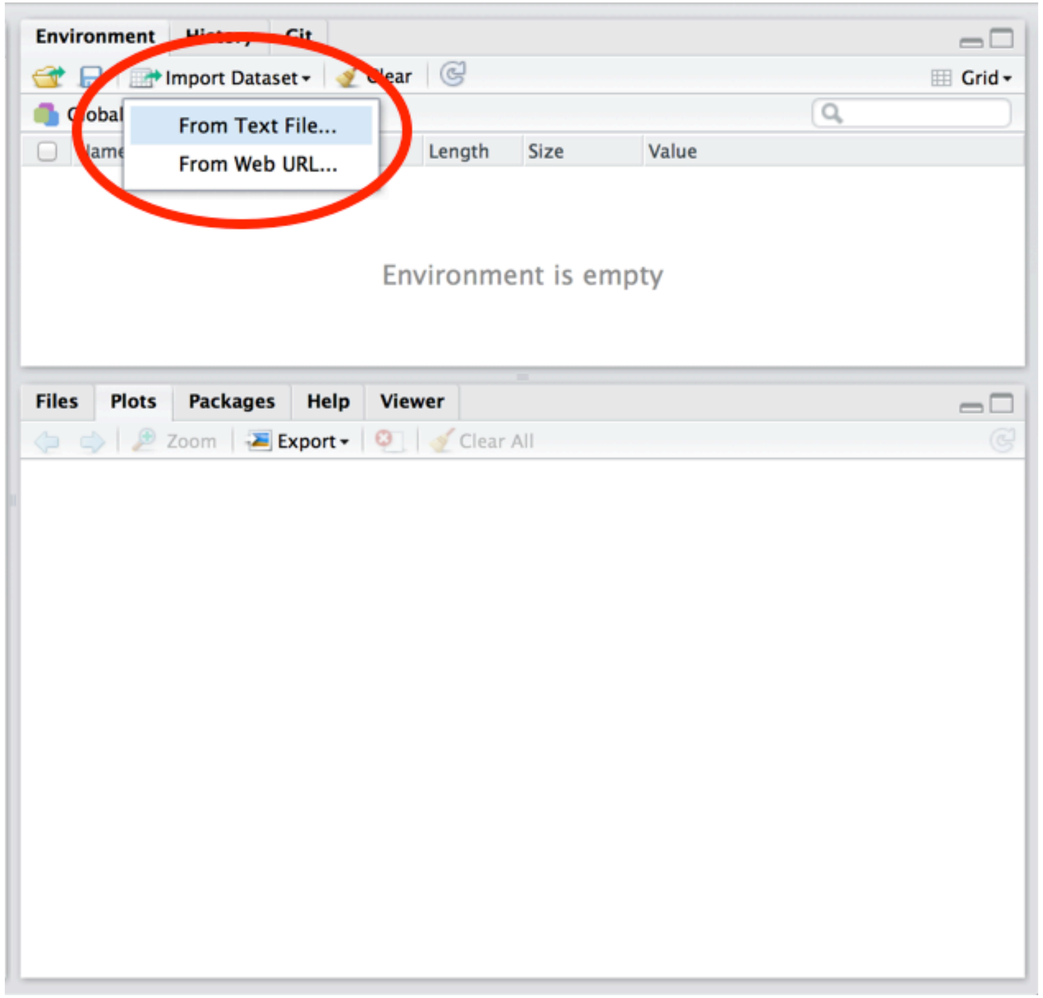
- To load a plain-text file into R, click the Import Dataset icon in RStudio
- RStudio will ask you to select the file you want to import, then it will open a wizard to help you import the data
- Use the wizard to tell RStudio what name to give the data set.



R Objects

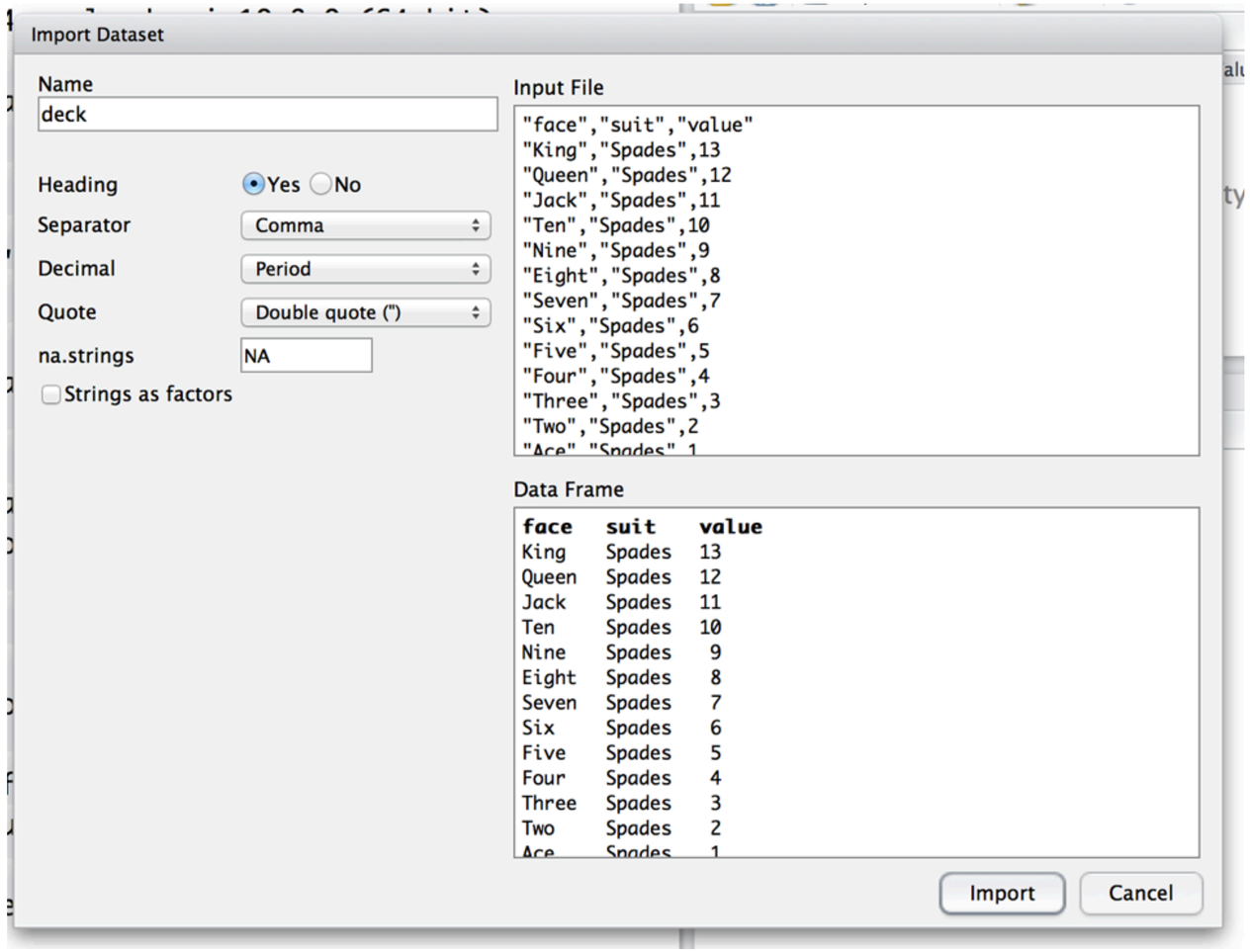
Loading data

- To load a plain-text file into R, click the Import Dataset icon in RStudio
- RStudio will ask you to select the file you want to import, then it will open a wizard to help you import the data
- Use the wizard to tell RStudio what name to give the data set.
- Tell RStudio which character the data set uses as a separator, which character represents decimals, whether the data set comes with a row of column names.



R Objects

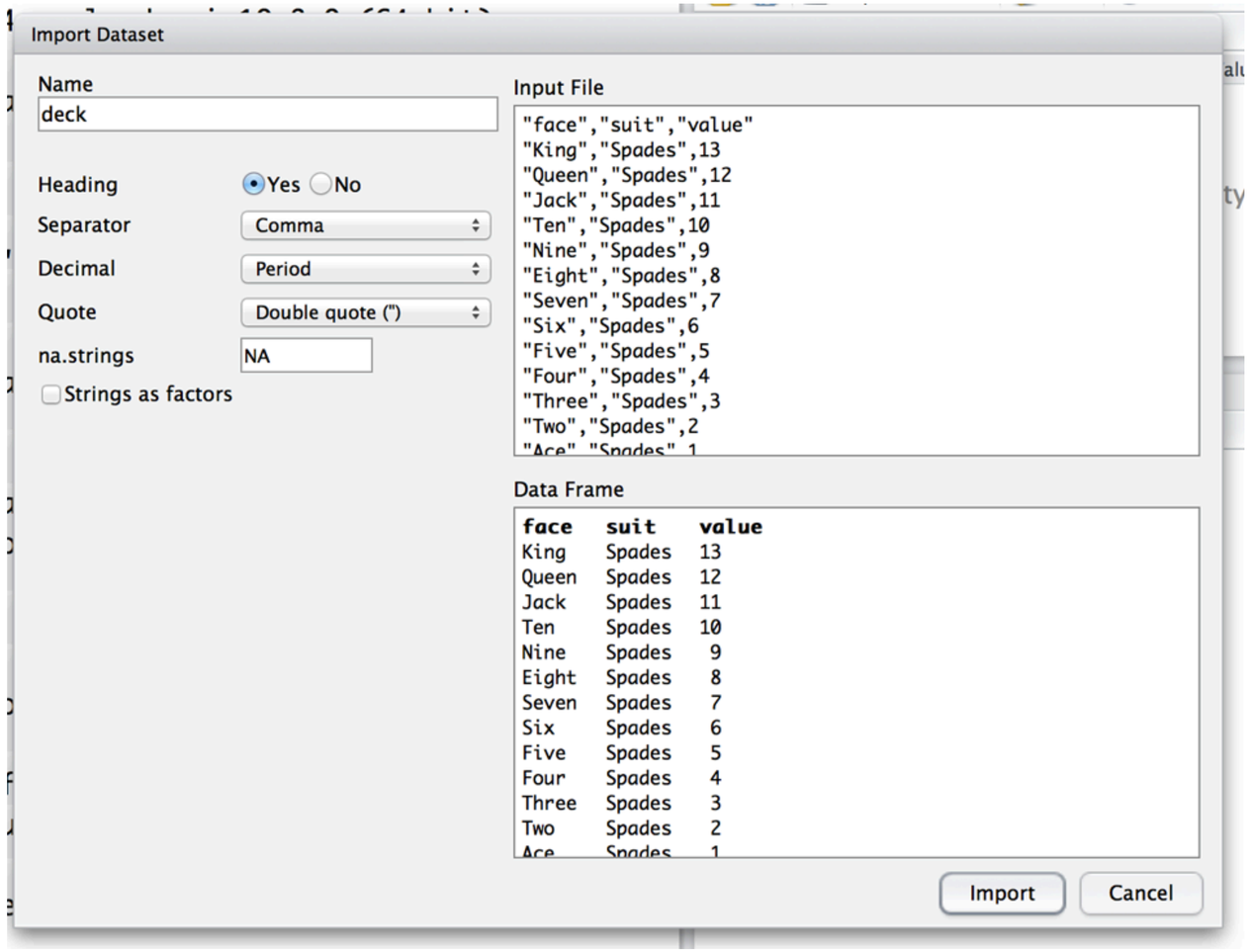
Loading data



R Objects

Loading data

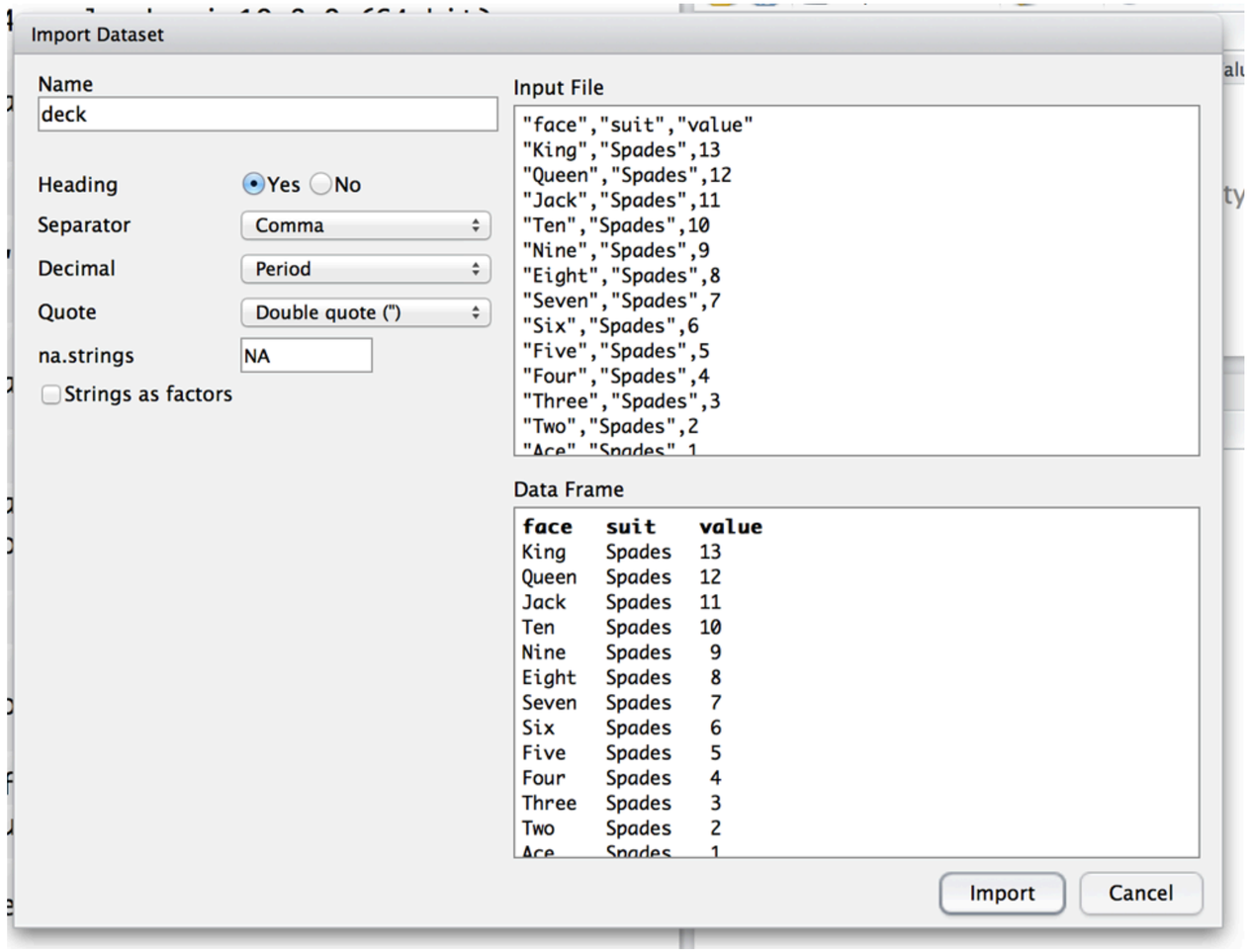
- To load a plain-text file into R, click the Import Dataset icon in RStudio



R Objects

Loading data

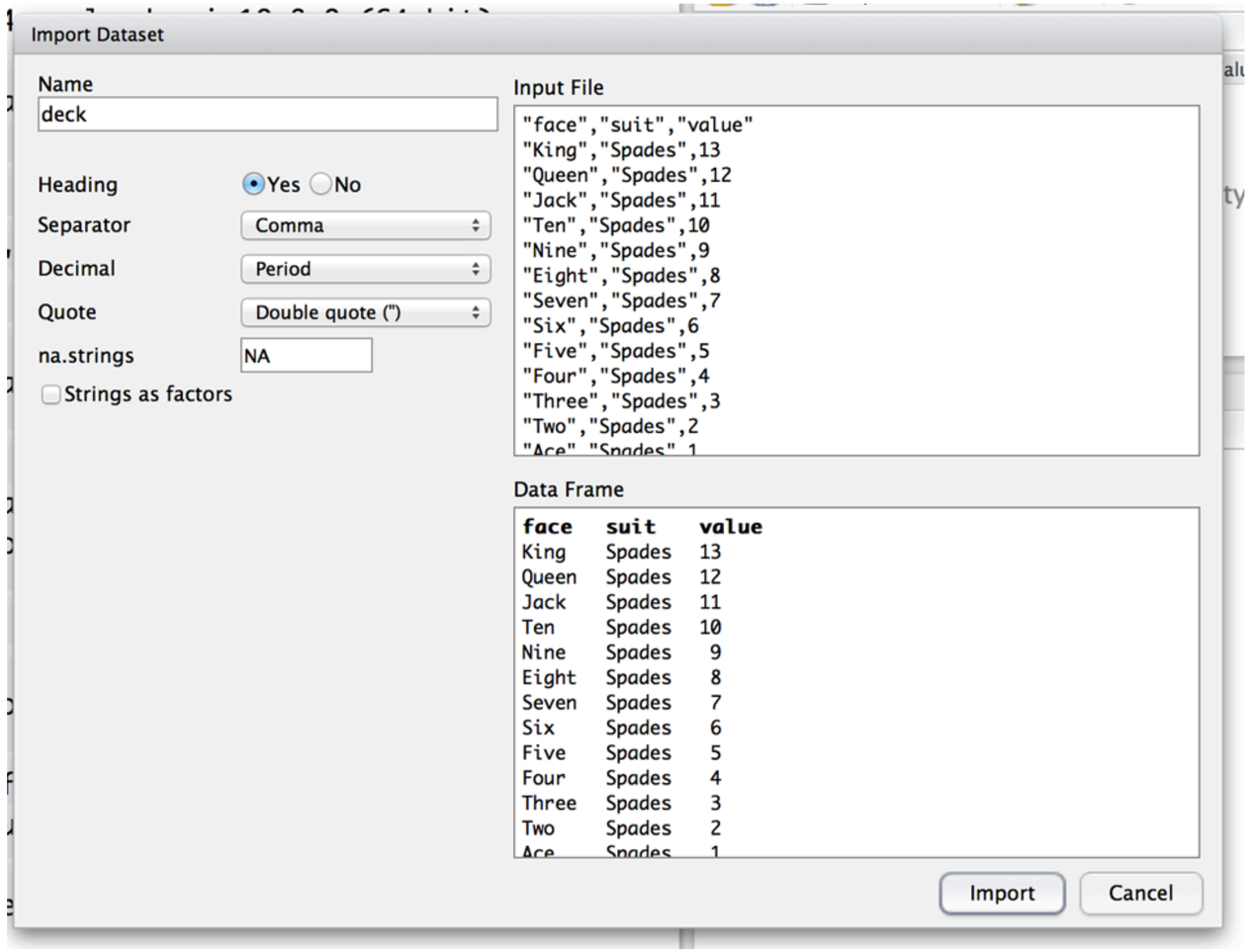
- To load a plain-text file into R, click the Import Dataset icon in RStudio
- RStudio will ask you to select the file you want to import, then it will open a wizard to help you import the data



R Objects

Loading data

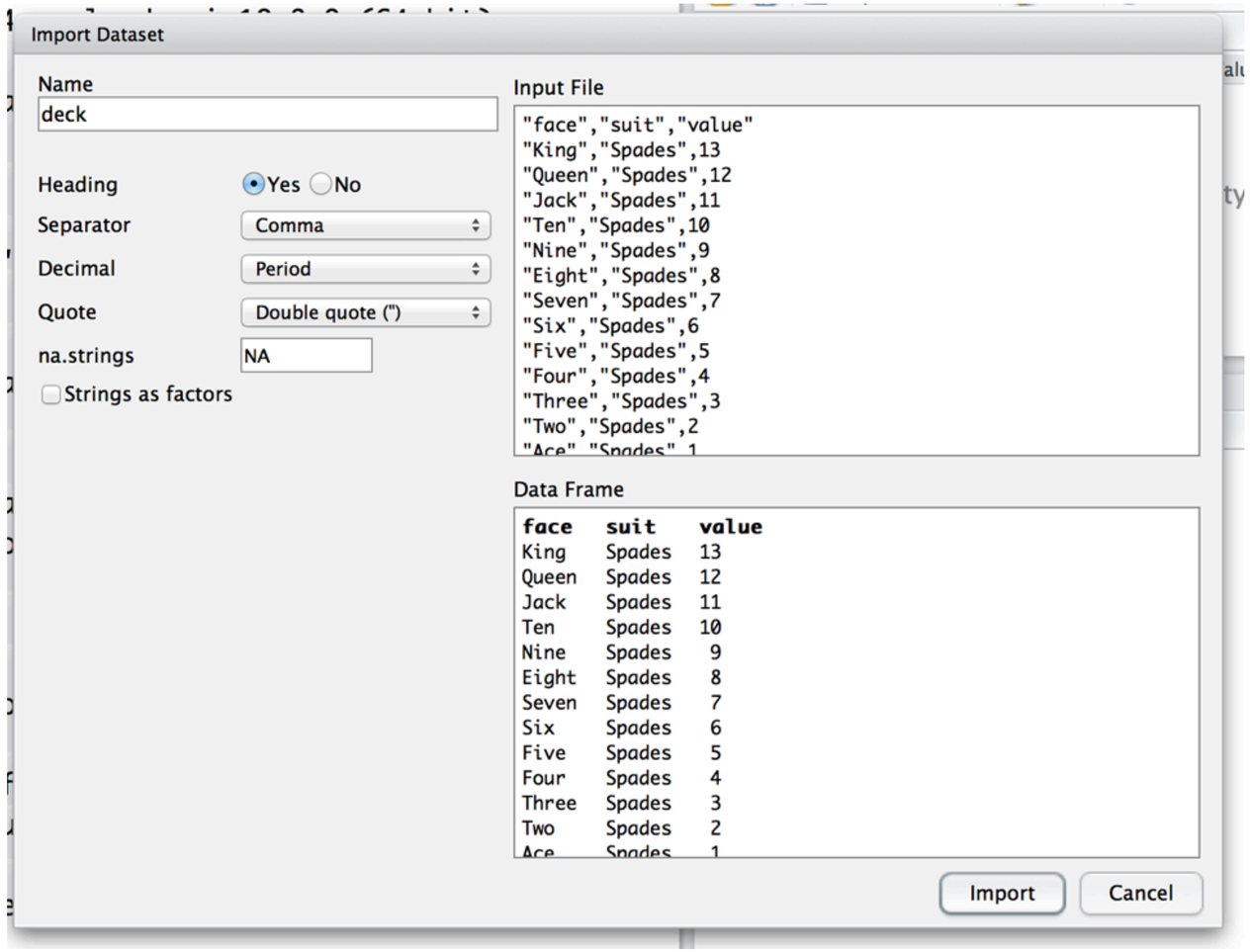
- To load a plain-text file into R, click the Import Dataset icon in RStudio
- RStudio will ask you to select the file you want to import, then it will open a wizard to help you import the data
- Use the wizard to tell RStudio what name to give the data set.



R Objects

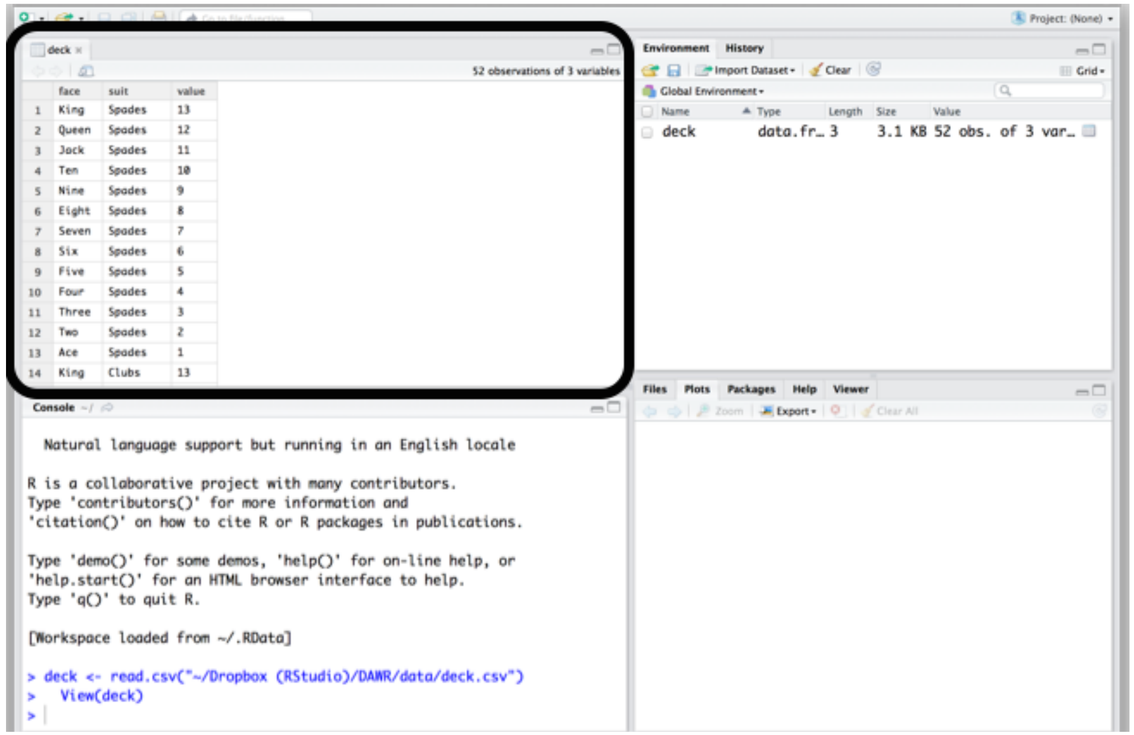
Loading data

- To load a plain-text file into R, click the Import Dataset icon in RStudio
- RStudio will ask you to select the file you want to import, then it will open a wizard to help you import the data
- Use the wizard to tell RStudio what name to give the data set.
- Tell RStudio which character the data set uses as a separator, which character represents decimals, whether the data set comes with a row of column names.



R Objects

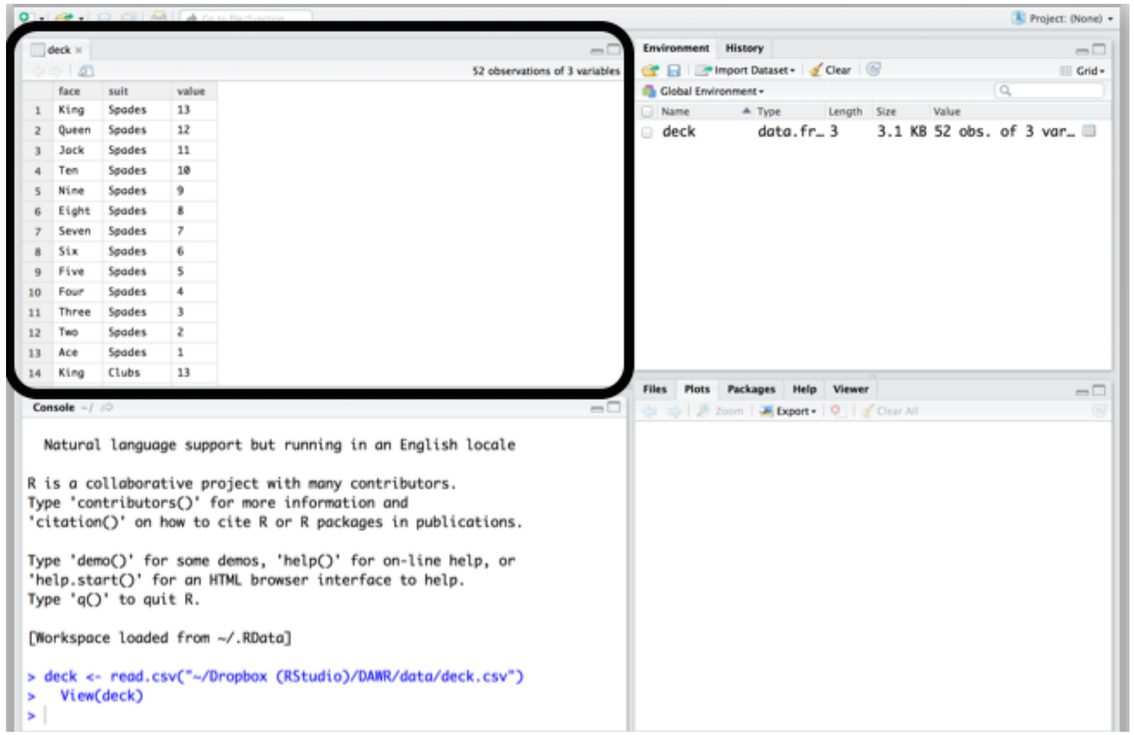
Loading data



R Objects

Loading data

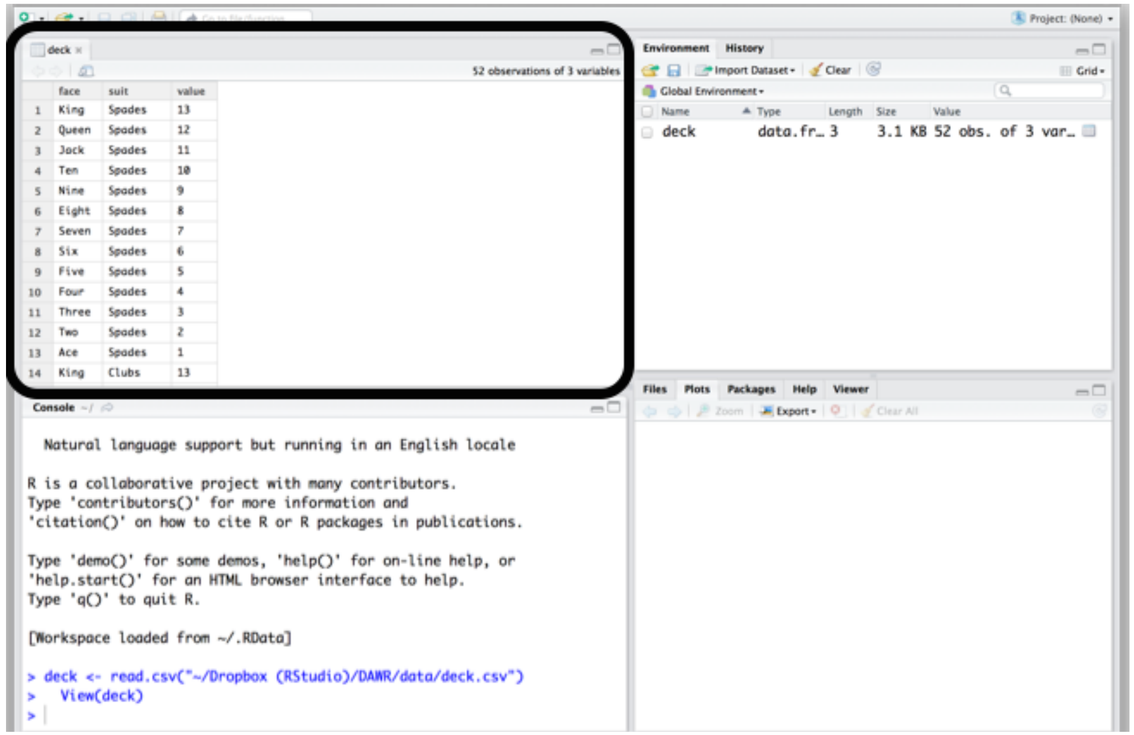
- RStudio will read in the data and save it to a data frame.



R Objects

Loading data

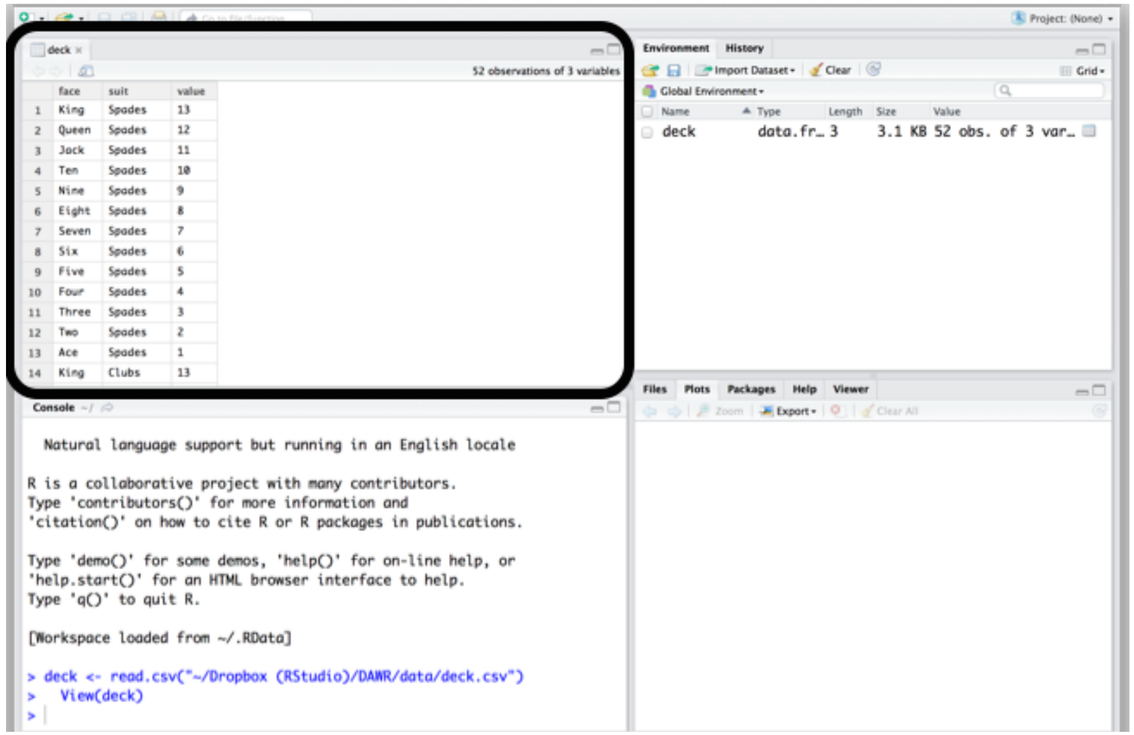
- RStudio will read in the data and save it to a data frame.
- RStudio will also open a data viewer, so you can see your new data in a spreadsheet format.



R Objects

Loading data

- RStudio will read in the data and save it to a data frame.
- RStudio will also open a data viewer, so you can see your new data in a spreadsheet format.
- If all worked well, your file should appear in a View tab of RStudio.



R Objects

Saving data

Console | Terminal | Render | Background Jobs

R 4.4.1 · /cloud/project/

```
> write.csv(deck, file = "cards.csv", row.names = FALSE)
```

R Objects

Saving data

- Before we go any further, let's save a copy of `deck` as a new *.csv* file.

Console | Terminal | Render | Background Jobs

R 4.4.1 · /cloud/project/

```
> write.csv(deck, file = "cards.csv", row.names = FALSE)
```

R Objects

Saving data

- Before we go any further, let's save a copy of `deck` as a new *.csv* file.
- That way you can email it to a colleague, store it on a thumb drive, or open it in a different program.

Console | Terminal | Render | Background Jobs

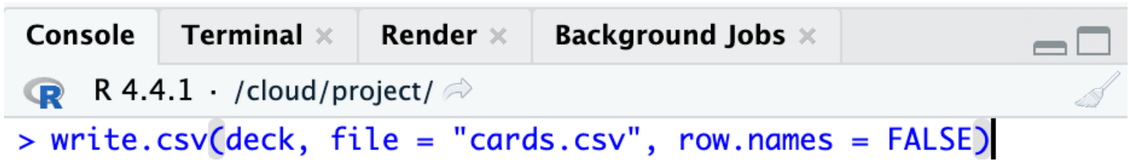
R 4.4.1 · /cloud/project/

```
> write.csv(deck, file = "cards.csv", row.names = FALSE)
```

R Objects

Saving data

- Before we go any further, let's save a copy of `deck` as a new *.csv* file.
- That way you can email it to a colleague, store it on a thumb drive, or open it in a different program.
- You can save any data frame in R to a *.csv* file with the command `write.csv`



R Objects

Saving data

Console | Terminal | Render | Background Jobs

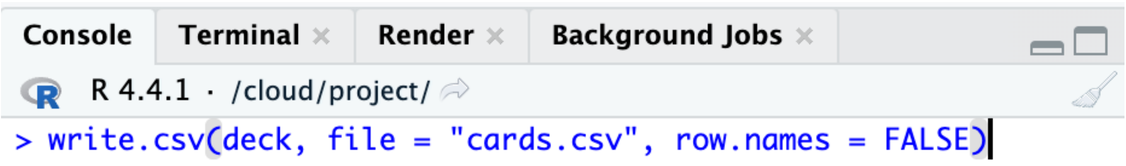
R 4.4.1 · /cloud/project/

```
> write.csv(deck, file = "cards.csv", row.names = FALSE)
```

R Objects

Saving data

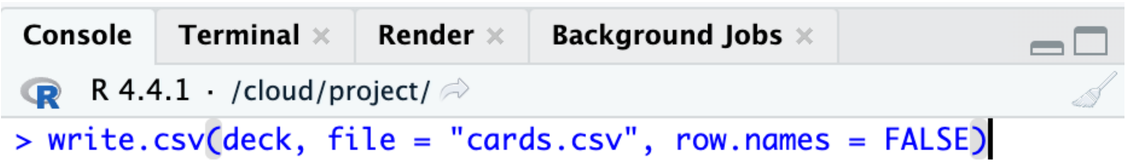
- To see where your working directory is, run `getwd()`



R Objects

Saving data

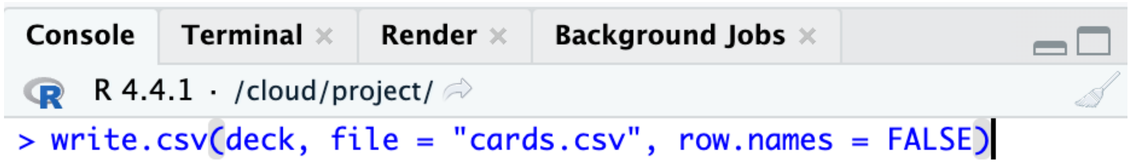
- To see where your working directory is, run `getwd()`
- To change the location of your working directory, visit Session > Set Working Directory > Choose Directory in the RStudio menu bar.



R Objects

Saving data

- To see where your working directory is, run `getwd()`
- To change the location of your working directory, visit Session > Set Working Directory > Choose Directory in the RStudio menu bar.
- You can customize the save process with `write.csv`'s large set of optional arguments (see `?write.csv` for details).



R Objects

Saving data

Console | Terminal | Render | Background Jobs

R 4.4.1 · /cloud/project/

```
> write.csv(deck, file = "cards.csv", row.names = FALSE)
```

R Objects

Saving data

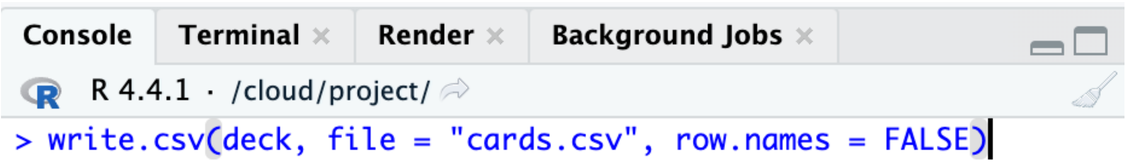
- There are three arguments that you should use *every* time you run `write.csv`.

```
Console   Terminal ×   Render ×   Background Jobs ×
R 4.4.1 · /cloud/project/
> write.csv(deck, file = "cards.csv", row.names = FALSE)
```

R Objects

Saving data

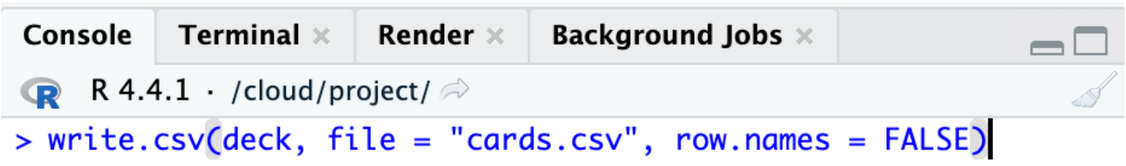
- There are three arguments that you should use *every* time you run `write.csv`.
- Add the argument `row.names = FALSE`. This will prevent R from adding a column of numbers at the start of your data frame.



R Objects

Saving data

- There are three arguments that you should use *every* time you run `write.csv`.
- Add the argument `row.names = FALSE`. This will prevent R from adding a column of numbers at the start of your data frame.
- You now have a virtual deck of cards to work with.



R Notation

Introduction

R Notation

Introduction

- Now that you have a deck of cards, you need a way to do card-like things with it.

R Notation

Introduction

- Now that you have a deck of cards, you need a way to do card-like things with it.
- First, you'll want to reshuffle the deck from time to time.

R Notation

Introduction

- Now that you have a deck of cards, you need a way to do card-like things with it.
- First, you'll want to reshuffle the deck from time to time.
- Next, you'll want to deal cards from the deck (one card at a time, whatever card is on top—we're not cheaters).

R Notation

Introduction

```
deal(deck)

##   face    suit value
##   king spades    13
```

R Notation

Introduction

- You'll need to work with the individual values inside your data frame

```
deal(deck)

##   face    suit value
##   king spades    13
```

R Notation

Introduction

- You'll need to work with the individual values inside your data frame
- For example, to deal a card from the top of your deck, you'll need to write a function that selects the first row of values in your data frame.

```
deal(deck)

##   face   suit value
##   king spades    13
```

R Notation

Introduction

- You'll need to work with the individual values inside your data frame
- For example, to deal a card from the top of your deck, you'll need to write a function that selects the first row of values in your data frame.
- Like this

```
deal(deck)

##   face   suit value
##   king spades    13
```

R Notation

Selecting Values

```
deck[ , ]
```

R Notation

Selecting Values

- R has a notation system that lets you extract values from R objects.

```
deck[ , ]
```

R Notation

Selecting Values

- R has a notation system that lets you extract values from R objects.
- To extract a value or set of values from a data frame, write the data frame's name followed by a pair of hard brackets.

```
deck[ , ]
```

R Notation

Selecting Values

- R has a notation system that lets you extract values from R objects.
- To extract a value or set of values from a data frame, write the data frame's name followed by a pair of hard brackets.
- Between the brackets will go two indexes separated by a comma.

```
deck[ , ]
```

R Notation

Selecting Values

- R has a notation system that lets you extract values from R objects.
- To extract a value or set of values from a data frame, write the data frame's name followed by a pair of hard brackets.
- Between the brackets will go two indexes separated by a comma.
- The indexes tell R which values to return.

```
deck[ , ]
```

R Notation

Selecting Values

- R has a notation system that lets you extract values from R objects.
- To extract a value or set of values from a data frame, write the data frame's name followed by a pair of hard brackets.
- Between the brackets will go two indexes separated by a comma.
- The indexes tell R which values to return.
- R will use the first index to subset the rows of the data frame and the second index to subset the columns.

```
deck[ , ]
```

R Notation

Selecting Values

```
deck[ , ]
```

R Notation

Selecting Values

- You have a choice when it comes to writing indexes.

```
deck[ , ]
```

R Notation

Selecting Values

- You have a choice when it comes to writing indexes.
- There are six different ways to write an index for R.

```
deck[ , ]
```

R Notation

Selecting Values

- You have a choice when it comes to writing indexes.
- There are six different ways to write an index for R.
- Positive integers

```
deck[ , ]
```

R Notation

Selecting Values

- You have a choice when it comes to writing indexes.
- There are six different ways to write an index for R.
- Positive integers
- Negative integers

```
deck[ , ]
```

R Notation

Selecting Values

- You have a choice when it comes to writing indexes.
- There are six different ways to write an index for R.
- Positive integers
- Negative integers
- Zero

```
deck[ , ]
```

R Notation

Selecting Values

- You have a choice when it comes to writing indexes.
- There are six different ways to write an index for R.
- Positive integers
- Negative integers
- Zero
- Blank spaces

```
deck[ , ]
```

R Notation

Selecting Values

- You have a choice when it comes to writing indexes.
- There are six different ways to write an index for R.
- Positive integers
- Negative integers
- Zero
- Blank spaces
- Logical values

```
deck[ , ]
```

R Notation

Selecting Values

- You have a choice when it comes to writing indexes.
- There are six different ways to write an index for R.
- Positive integers
- Negative integers
- Zero
- Blank spaces
- Logical values
- Names

```
deck[ , ]
```

R Notation

Selecting Values - Positive int.

```
deck[1, c(1, 2, 3)]
## face   suit value
## king spades    13
```

R Notation

Selecting Values - Positive int.

- R treats positive integers just like *ij* notation in linear algebra: `deck[i,j]` will return the value of `deck` that is in the *ith* row and the *jth* column.

```
deck[1, c(1, 2, 3)]

## face   suit value

## king spades    13
```

R Notation

Selecting Values - Positive int.

- R treats positive integers just like *ij* notation in linear algebra: `deck[i,j]` will return the value of `deck` that is in the *ith* row and the *jth* column.
- To extract more than one value, use a vector of positive integers.

```
deck[1, c(1, 2, 3)]

## face   suit value

## king spades    13
```

R Notation

Selecting Values - Positive int.

- R treats positive integers just like *ij* notation in linear algebra: `deck[i,j]` will return the value of `deck` that is in the *ith* row and the *jth* column.
- To extract more than one value, use a vector of positive integers.
- For example, you can return the first row of `deck` with `deck[1, c(1, 2, 3)]` or `deck[1, 1:3]`

```
deck[1, c(1, 2, 3)]

## face   suit value

## king spades    13
```

R Notation

Selecting Values - Positive int.

- R treats positive integers just like *ij* notation in linear algebra: `deck[i,j]` will return the value of `deck` that is in the *ith* row and the *jth* column.
- To extract more than one value, use a vector of positive integers.
- For example, you can return the first row of `deck` with `deck[1, c(1, 2, 3)]` or `deck[1, 1:3]`
- For example, you can return the first row of `deck` with `deck[1, c(1, 2, 3)]` or `deck[1, 1:3]`:

```
deck[1, c(1, 2, 3)]

## face   suit value

## king spades    13
```

R Notation

Selecting Values

```
new <- deck[1, c(1, 2, 3)]
new
## face   suit value
## king spades    13
```

R Notation

Selecting Values

- Note that R won't actually remove the selecting values from `deck`.

```
new <- deck[1, c(1, 2, 3)]
new
## face   suit value
## king spades    13
```

R Notation

Selecting Values

- Note that R won't actually remove the selecting values from `deck`.
- R will give you a new set of values which are copies of the original values.

```
new <- deck[1, c(1, 2, 3)]
new
## face   suit value
## king spades    13
```

R Notation

Selecting Values

- Note that R won't actually remove the selecting values from `deck`.
- R will give you a new set of values which are copies of the original values.
- You can then save this new set to an R object with R's assignment operator.

```
new <- deck[1, c(1, 2, 3)]
new
## face   suit value
## king spades    13
```

R Notation

Selecting Values

6	1	3	6	10	5

```
vec[5]
```

John	1940	guitar
Paul	1941	bass
George	1943	guitar
Ringo	1940	drums

```
df[2, c(2,3)]
```

R Notation

Selecting Values

```
vec <- c(6, 1, 3, 6, 10, 5)

vec[1:3]

## 6 1 3
```

R Notation

Selecting Values

```
deck[1:2, 1:2]
##  face   suit
##  king spades
## queen spades
```

R Notation

Selecting Values

- If you select two or more columns from a data frame, R will return a new data frame.

```
deck[1:2, 1:2]
##  face   suit
##  king spades
## queen spades
```

R Notation

Selecting Values

- If you select two or more columns from a data frame, R will return a new data frame.
- If you select a single column, R will return a vector.

```
deck[1:2, 1:2]
##  face   suit
##  king spades
## queen spades
```

R Notation

Selecting Values

- If you select two or more columns from a data frame, R will return a new data frame.
- If you select a single column, R will return a vector.
- If you would prefer a data frame instead, you can add the optional argument `drop = FALSE` between the brackets

```
deck[1:2, 1:2]
##  face   suit
##  king spades
## queen spades
```

R Notation

Selecting Values

- If you select two or more columns from a data frame, R will return a new data frame.
- If you select a single column, R will return a vector.
- If you would prefer a data frame instead, you can add the optional argument `drop = FALSE` between the brackets

```
deck[1:2, 1]

##  "king"  "queen"
```

R Notation

Selecting Values

- If you select two or more columns from a data frame, R will return a new data frame.
- If you select a single column, R will return a vector.
- If you would prefer a data frame instead, you can add the optional argument `drop = FALSE` between the brackets

```
deck[1:2, 1, drop = FALSE]
##  face
##  king
## queen
```

R Notation

Selecting Values

```
new <- deck[1, c(1, 2, 3)]
new
## face   suit value
## king spades    13
```

R Notation

Selecting Values

- Note that R won't actually remove the selecting values from `deck`.

```
new <- deck[1, c(1, 2, 3)]
new
## face   suit value
## king spades    13
```

R Notation

Selecting Values

- Note that R won't actually remove the selecting values from `deck`.
- R will give you a new set of values which are copies of the original values.

```
new <- deck[1, c(1, 2, 3)]
new
## face   suit value
## king spades    13
```

R Notation

Selecting Values

- Note that R won't actually remove the selecting values from `deck`.
- R will give you a new set of values which are copies of the original values.
- You can then save this new set to an R object with R's assignment operator.

```
new <- deck[1, c(1, 2, 3)]
new
## face   suit value
## king spades    13
```