CT5102: Programming for Data Analytics

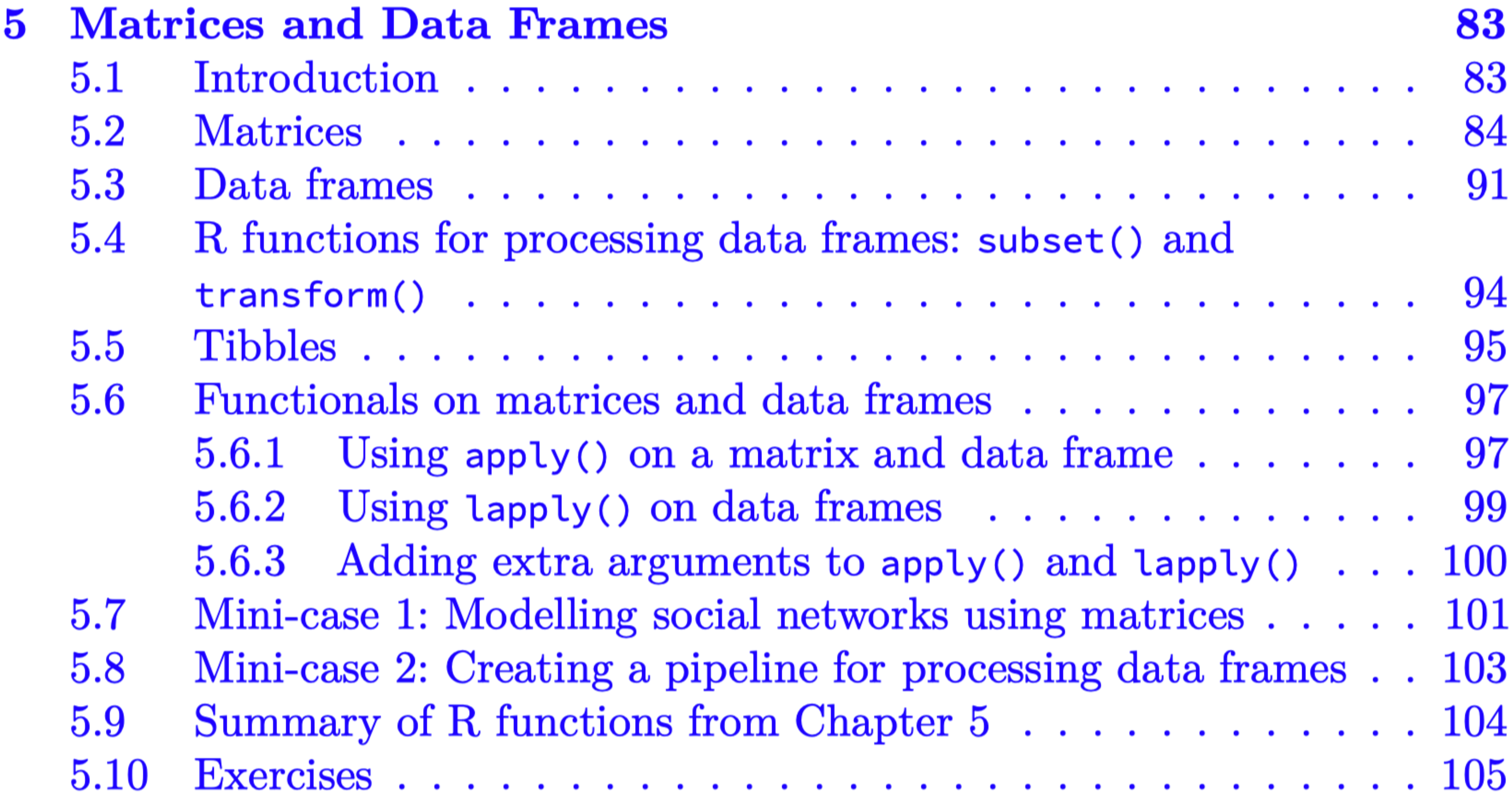
4. Matrices and Data Frames

Prof. Jim Duggan,

School of Computer Science

University of Galway.

https://github.com/JimDuggan/explore_or



5 Matrices and Data Frames 83

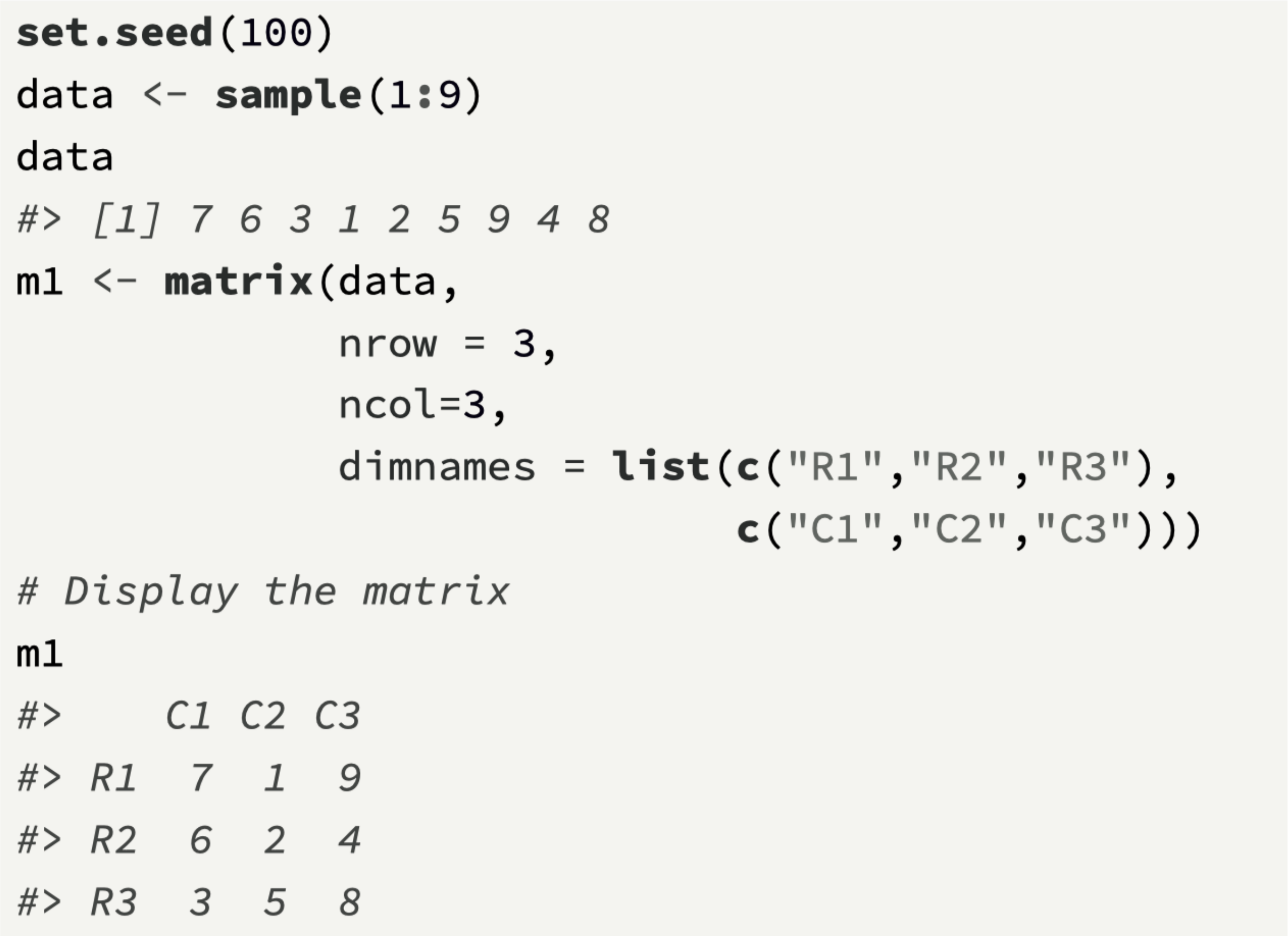
> On an intuitive level, a *data frame* is like a matrix, with a two-dimensional rows-and-columns structure. However, it differs from a matrix in that each column may have a different type.
>
> — Norman Matloff (Matloff, 2011)

Overview

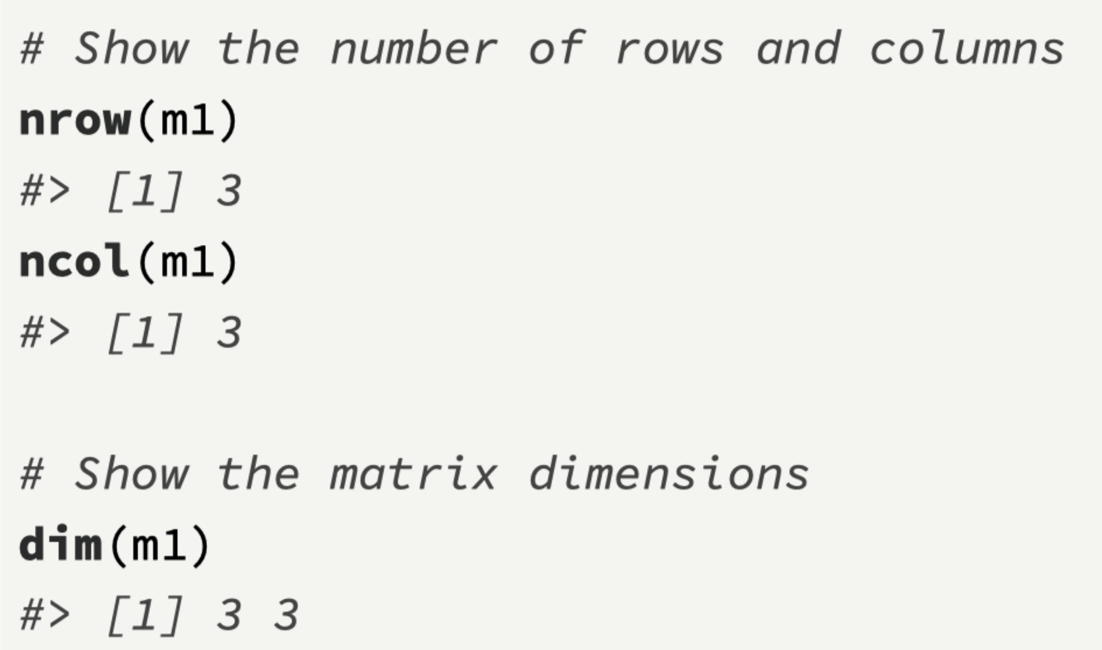
- To date, we have used atomic vectors and lists to store information.
- While these are foundational data structures in R, they do not provide support for processing rectangular data, which is a common format in data science.
- More generally, we can have two types of rectangular (two-dimensional) data of :
  1. The same type, typically numeric, that is stored in a matrix, and
  2. Different types that is stored in a data frame.

(4.1) Matrices

- a matrix is a two-dimensional structure, with rows and columns, that contains the same type.
- it is an atomic vector in two dimensions, and is created using the matrix() function, with the following arguments:
  - data, which are the initial values, contained in an atomic vector, supplied to the matrix,
  - nrow, the desired number of rows,
  - ncol, the desired number of columns,
  - byrow, a logical value (default is FALSE), that specifies what way to fill the matrix with data, either filled by row or by column,
  - dimnames, a list of length 2 giving row and column names, respectively.



```
set.seed(100)
data <- sample(1:9)
data
#> [1] 7 6 3 1 2 5 9 4 8
m1 <- matrix(data,
             nrow = 3,
             ncol=3,
             dimnames = list(c("R1","R2","R3"),
                             c("C1","C2","C3")))
# Display the matrix
m1
#>    C1 C2 C3
#> R1  7  1  9
#> R2  6  2  4
#> R3  3  5  8
```



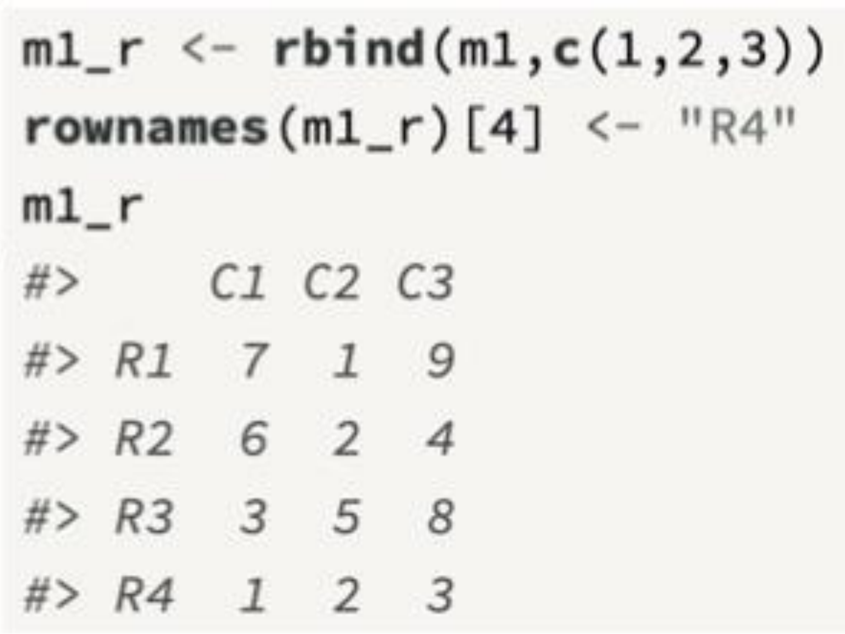
```
# Show the number of rows and columns
nrow(m1)
#> [1] 3
ncol(m1)
#> [1] 3

# Show the matrix dimensions
dim(m1)
#> [1] 3 3
```

Useful points

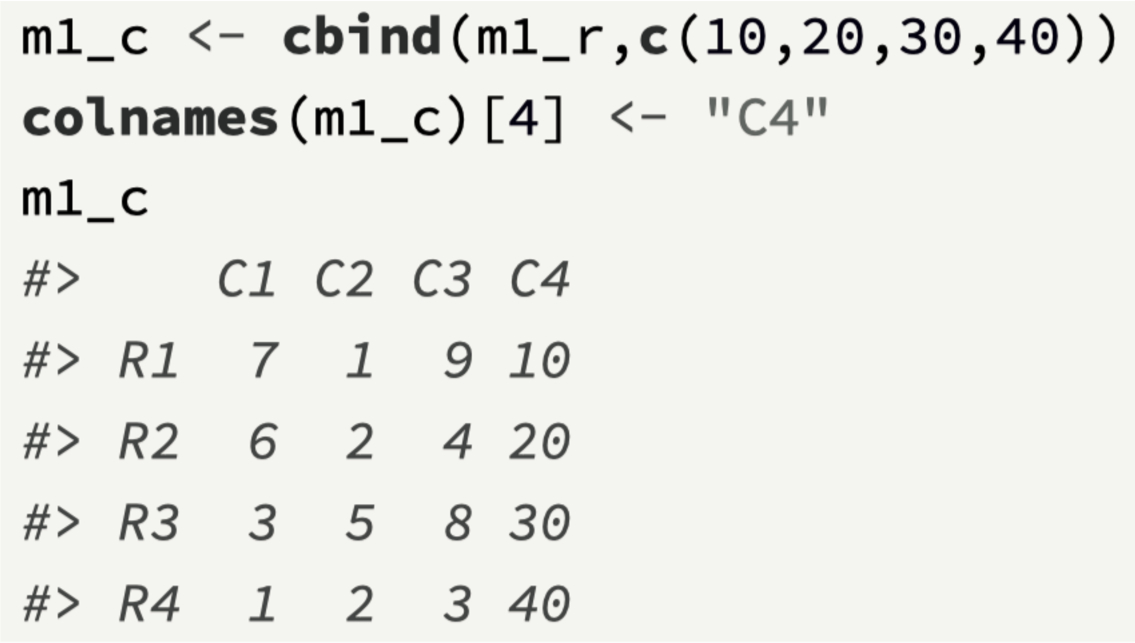
- The matrix is populated by column order as the default.
- If by_row was set to TRUE, then the matrix would be populated by row order.
- The row and column names are set using the dimnames argument. This is not required, and row names and column names can always be set on a matrix using the functions rownames() and colnames().
- The functions nrow() and ncol() can be used to return the matrix dimensions.
- The function dim() provides information on the matrix dimensions, and can also be used to resize a matrix, for example, converting a 3 × 3 to a 1 × 9.

Growing a matrix

- An important property of a matrix is that it can be extended
  - – Add rows using rbind()
  - – Add columns with cbind()

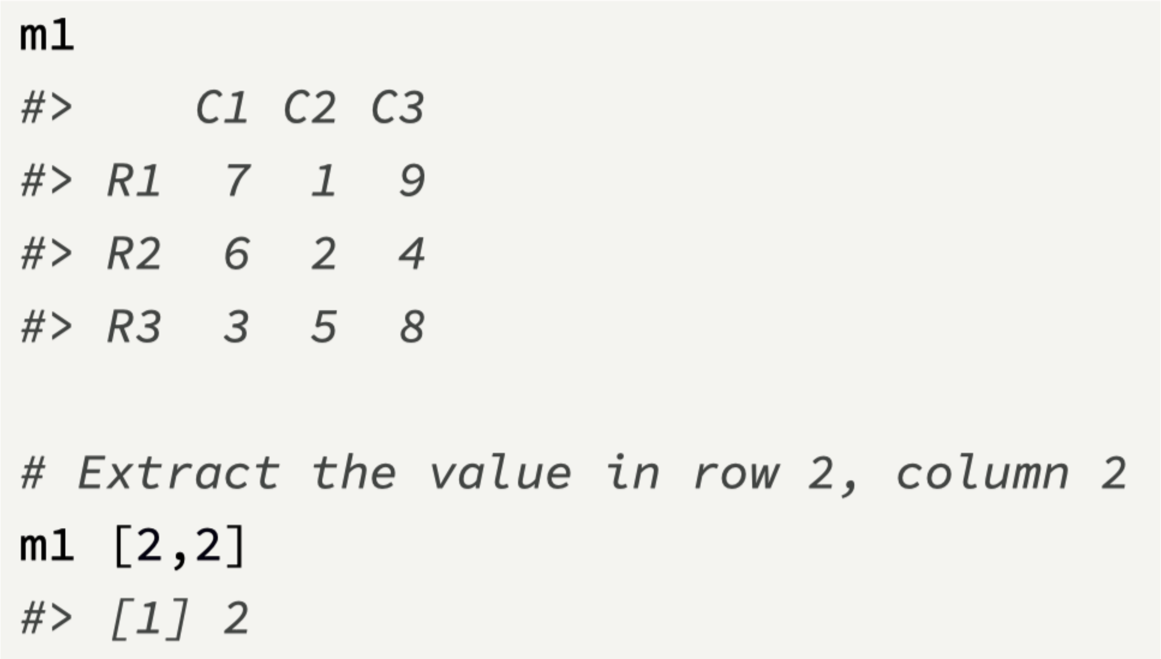


```
m1_r <- rbind(m1,c(1,2,3))
rownames(m1_r)[4] <- "R4"
m1_r
#>     C1 C2 C3
#> R1   7  1  9
#> R2   6  2  4
#> R3   3  5  8
#> R4   1  2  3
```

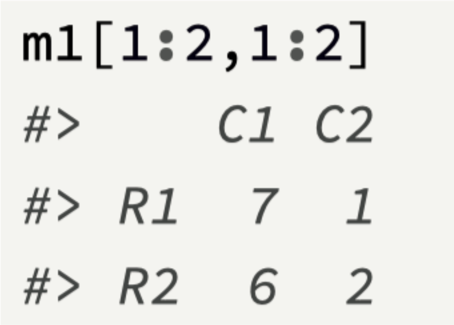


```
m1_c <- cbind(m1_r,c(10,20,30,40))
colnames(m1_c)[4] <- "C4"
m1_c
#>     C1 C2 C3 C4
#> R1   7  1  9 10
#> R2   6  2  4 20
#> R3   3  5  8 30
#> R4   1  2  3 40
```

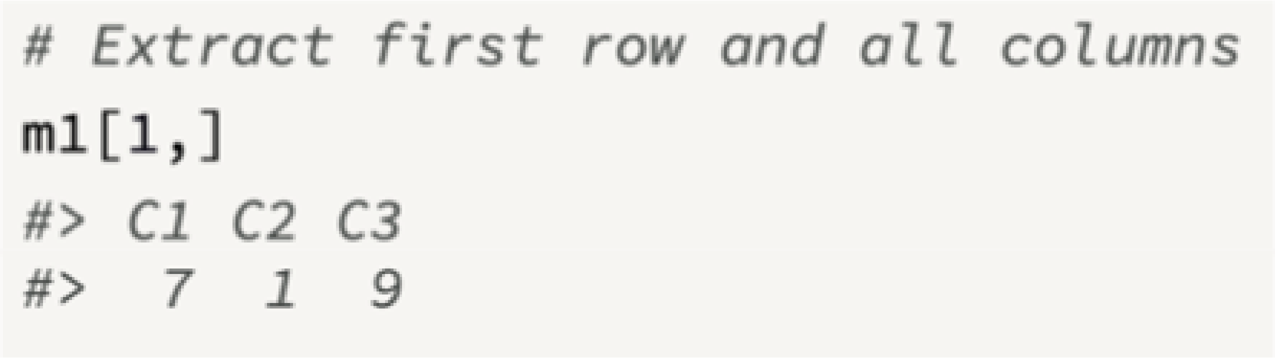
Subsetting a matrix, like vectors with 2 dimensions



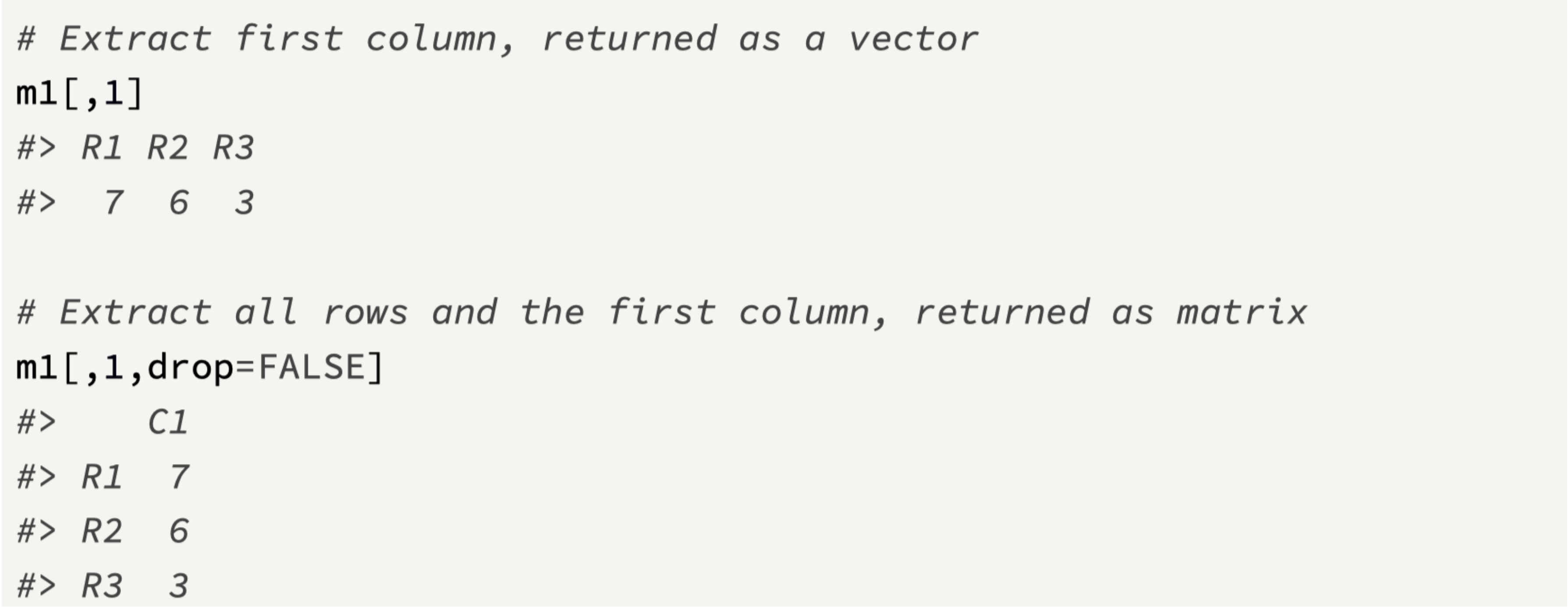
```
m1
#>     C1 C2 C3
#> R1   7  1  9
#> R2   6  2  4
#> R3   3  5  8

# Extract the value in row 2, column 2
m1 [2,2]
#> [1] 2
```



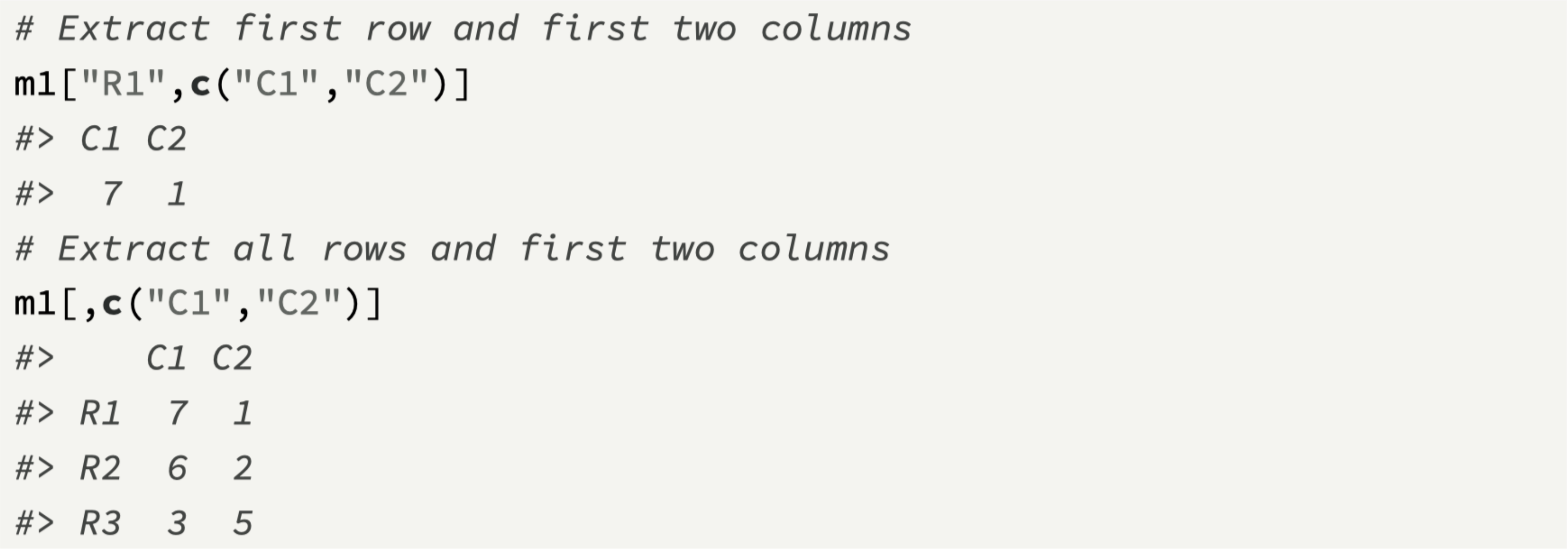
```
m1[1:2,1:2]
#>     C1 C2
#> R1   7  1
#> R2   6  2
```



```
# Extract first row and all columns
m1[1,]
#> C1 C2 C3
#>  7  1  9
```

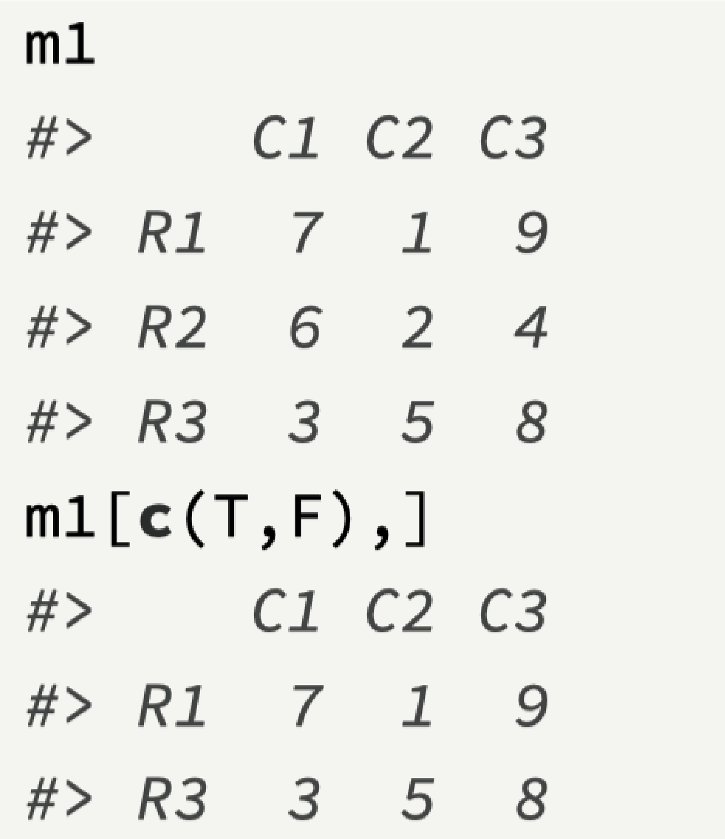


```
# Extract first column, returned as a vector
m1[,1]
#> R1 R2 R3
#>  7  6  3

# Extract all rows and the first column, returned as matrix
m1[,1,drop=FALSE]
#>    C1
#> R1  7
#> R2  6
#> R3  3
```

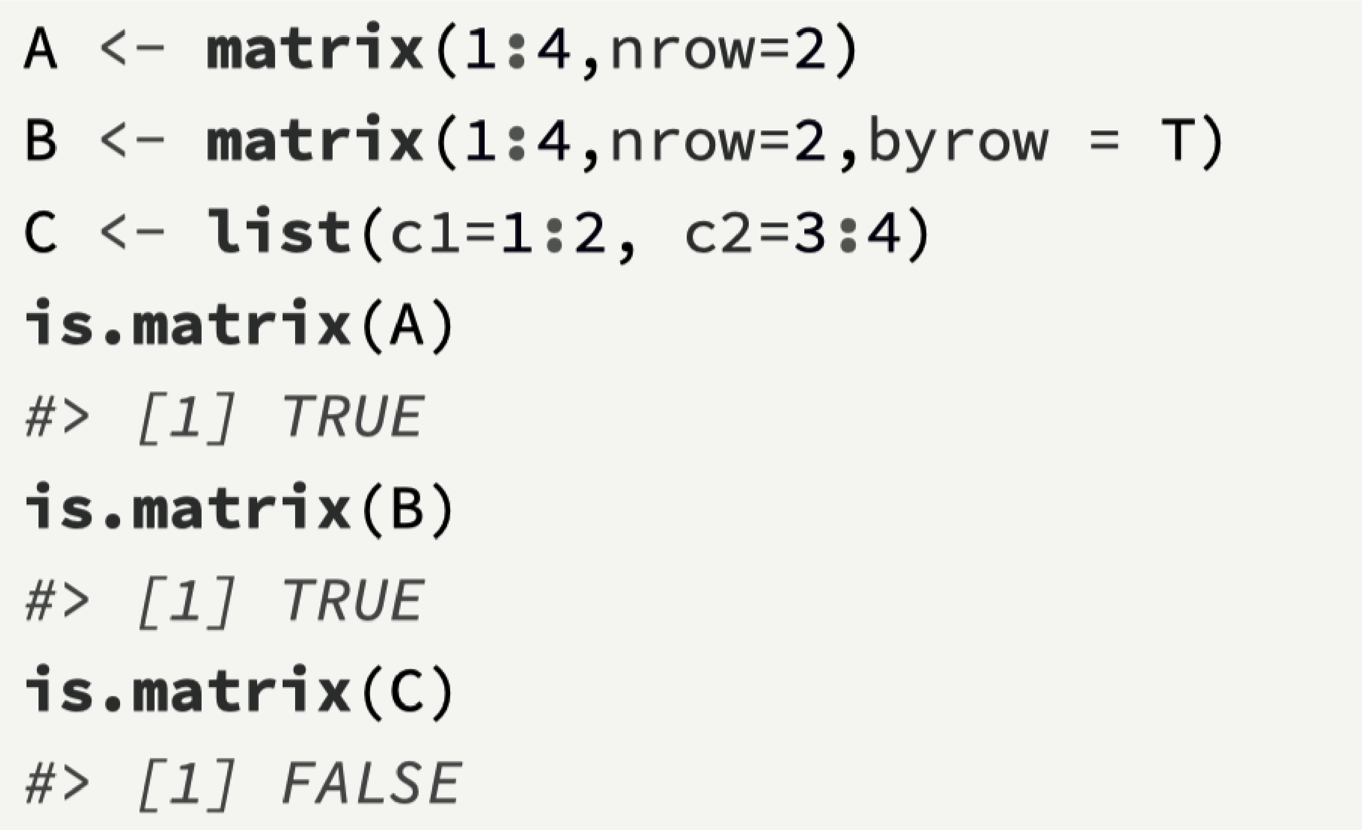


```
# Extract first row and first two columns
m1["R1",c("C1","C2")]
#> C1 C2
#>  7  1
# Extract all rows and first two columns
m1[,c("C1","C2")]
#>    C1 C2
#> R1  7  1
#> R2  6  2
#> R3  3  5
```

Using logical vectors and is.matrix()

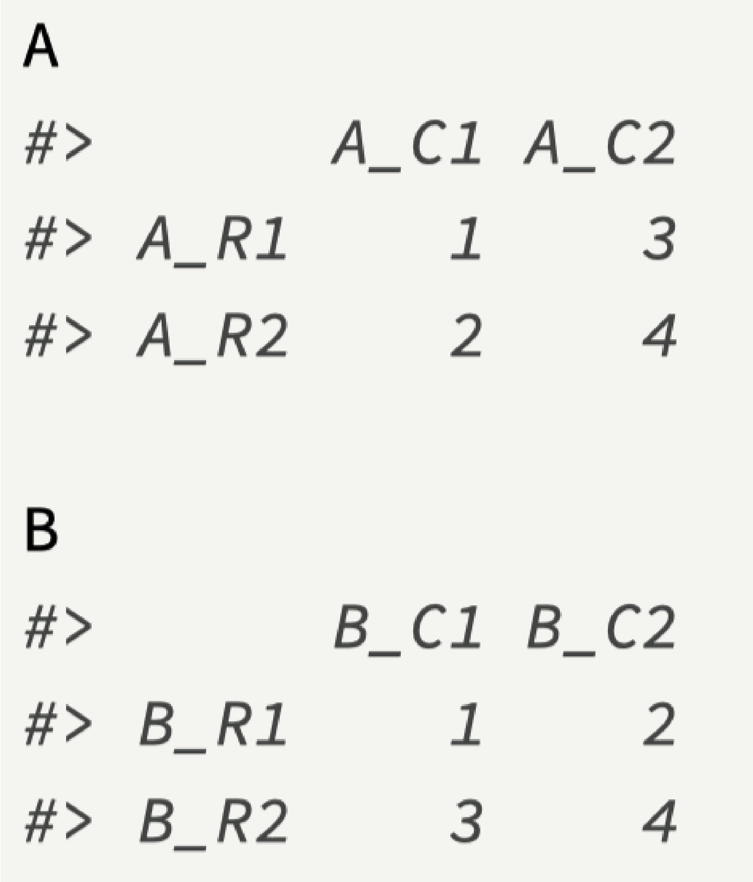


```
m1
#>     C1 C2 C3
#> R1   7  1  9
#> R2   6  2  4
#> R3   3  5  8
m1[c(T,F),]
#>     C1 C2 C3
#> R1   7  1  9
#> R3   3  5  8
```

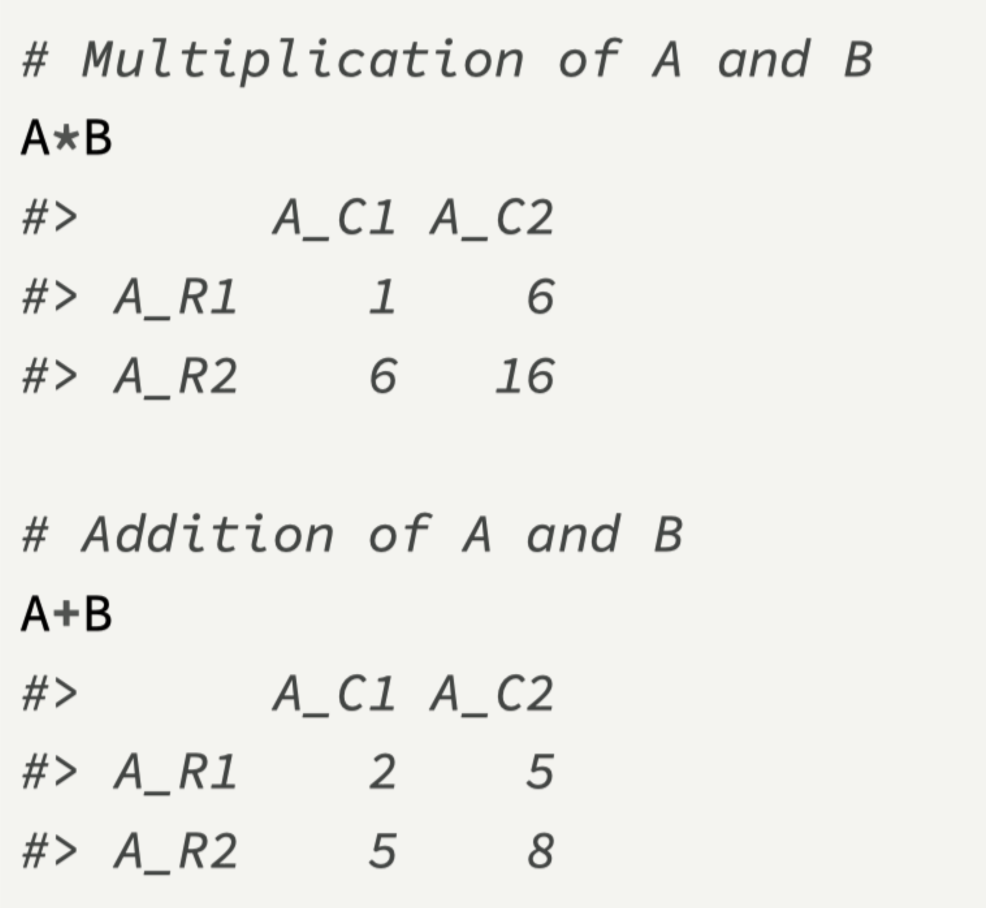


```
A <- matrix(1:4,nrow=2)
B <- matrix(1:4,nrow=2,byrow = T)
C <- list(c1=1:2, c2=3:4)
is.matrix(A)
#> [1] TRUE
is.matrix(B)
#> [1] TRUE
is.matrix(C)
#> [1] FALSE
```

Arithmetic Operators – Element-wide basis

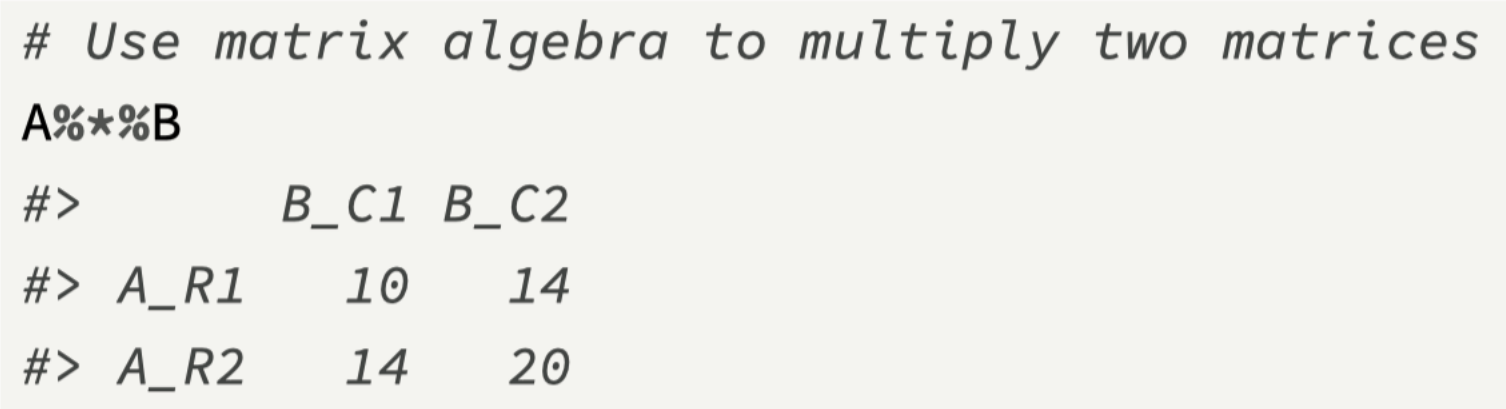


```
A
#>        A_C1 A_C2
#> A_R1      1    3
#> A_R2      2    4

B
#>        B_C1 B_C2
#> B_R1      1    2
#> B_R2      3    4
```

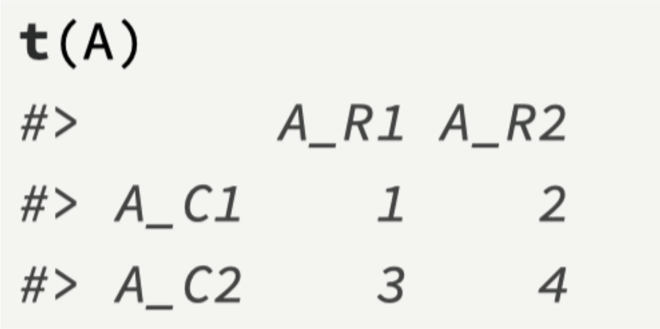


```
# Multiplication of A and B
A*B
#>      A_C1 A_C2
#> A_R1    1    6
#> A_R2    6   16

# Addition of A and B
A+B
#>      A_C1 A_C2
#> A_R1    2    5
#> A_R2    5    8
```

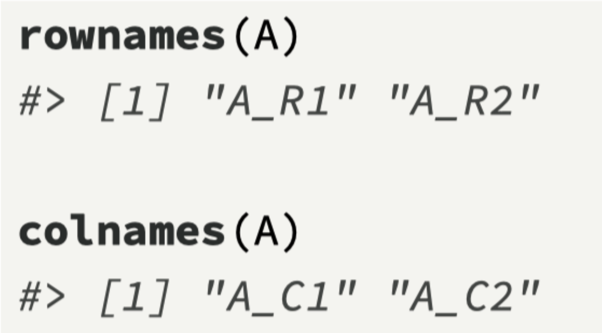
Other useful matrix functions



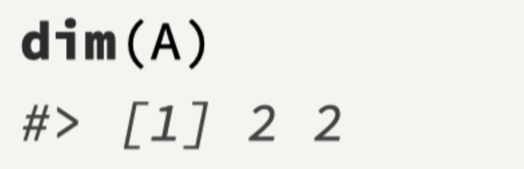
```
# Use matrix algebra to multiply two matrices
A%*%B
#>       B_C1 B_C2
#> A_R1    10   14
#> A_R2    14   20
```



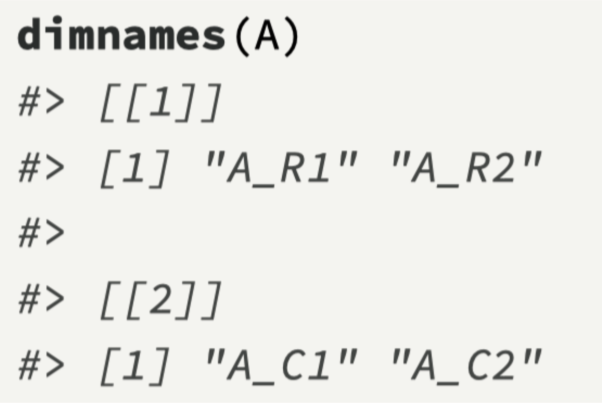
```
t(A)
#>      A_R1 A_R2
#> A_C1    1    2
#> A_C2    3    4
```



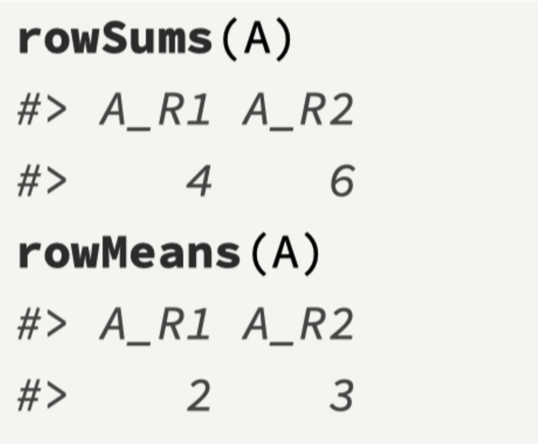
```
rownames(A)
#> [1] "A_R1" "A_R2"

colnames(A)
#> [1] "A_C1" "A_C2"
```



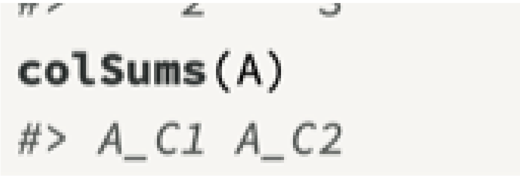
```
dim(A)
#> [1] 2 2
```



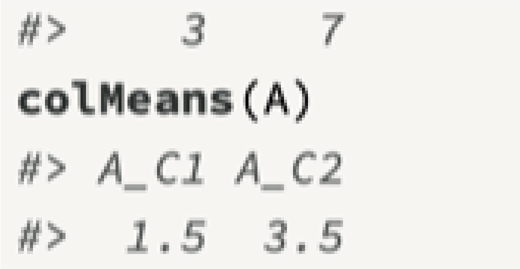
```
dimnames(A)
#> [[1]]
#> [1] "A_R1" "A_R2"
#>
#> [[2]]
#> [1] "A_C1" "A_C2"
```



```
rowSums(A)
#> A_R1 A_R2
#>    4    6
rowMeans(A)
#> A_R1 A_R2
#>    2    3
```



```
#>    2    3
colSums(A)
#> A_C1 A_C2
```



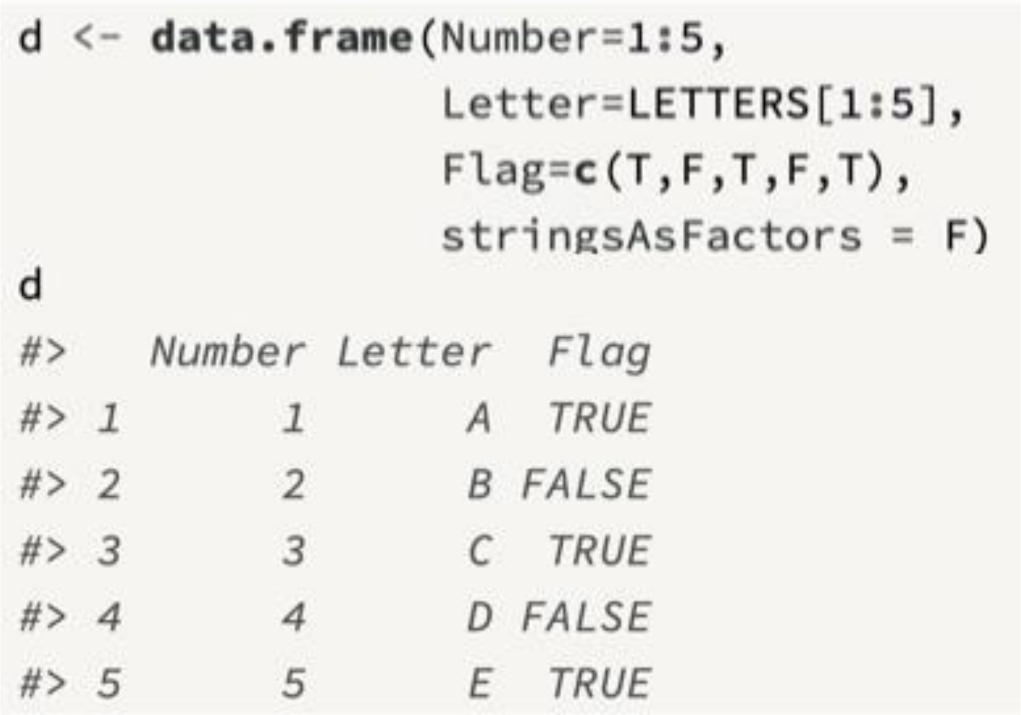
```
#>    3    7
colMeans(A)
#> A_C1 A_C2
#>  1.5  3.5
```

Matrices - Summary

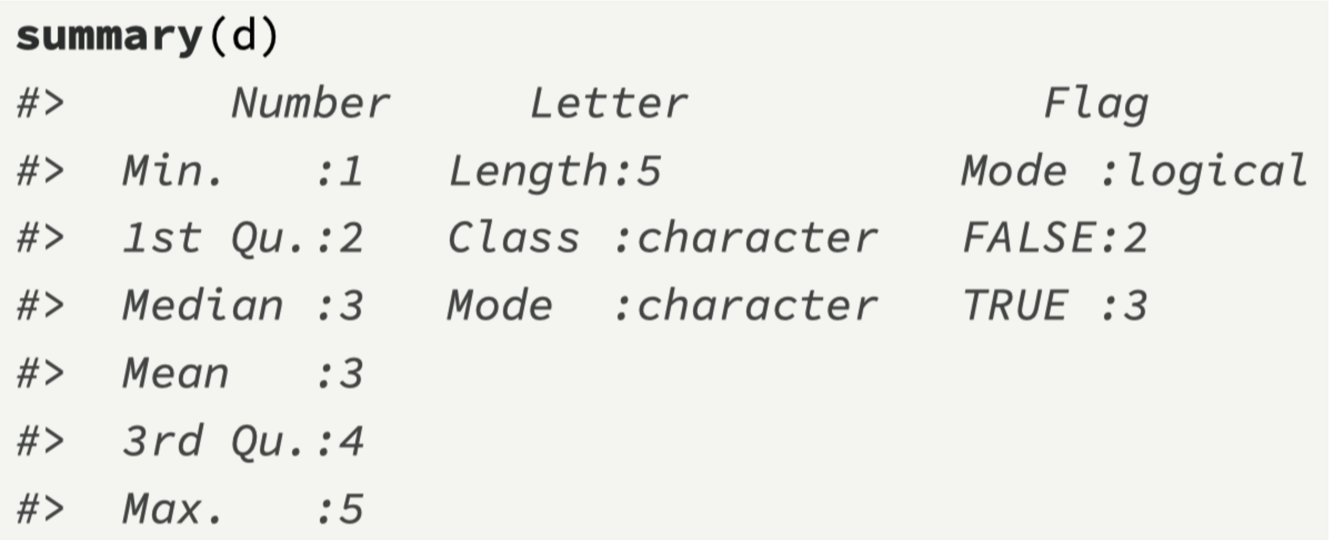
- R provides good support for problems that require matrix manipulation, *but all values need to be the same type*
- Matrices need to be defined using the matrix() function.
- Many of the subsetting commands used for atomic vectors can also be used for matrices, and that includes referencing elements by the row/column name.
- Matrices can be extended easily, using functions such as cbind() and rbind().

(4.2) Data Frames

- A data frame is similar to a matrix, with a two-dimensional row and column structure, while on a technical level, a data frame is a list, with the elements of that list containing equal length vectors (Matloff, 2011).
- It’s defined using the data.frame() function
- The elements (columns) of a data frame can be of different types
- The data frame, with its row and column structure, will be familiar to anyone who has used a spreadsheet, where each column is a variable, and every row is an observation.
- The data frame, and its successor, the tibble, will be used extensively during this course



```
d <- data.frame(Number=1:5,
                Letter=LETTERS[1:5],
                Flag=c(T,F,T,F,T),
                stringsAsFactors = F)
d
#>   Number Letter  Flag
#> 1      1      A  TRUE
#> 2      2      B FALSE
#> 3      3      C  TRUE
#> 4      4      D FALSE
#> 5      5      E  TRUE
```

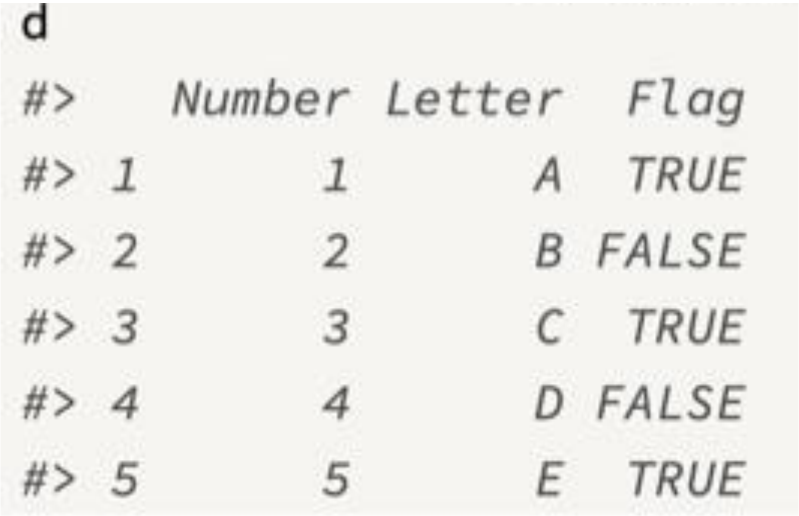


```
summary(d)
#>      Number     Letter               Flag
#>  Min.   :1   Length:5           Mode :logical
#>  1st Qu.:2   Class :character   FALSE:2
#>  Median :3   Mode  :character   TRUE :3
#>  Mean   :3
#>  3rd Qu.:4
#>  Max.   :5
```

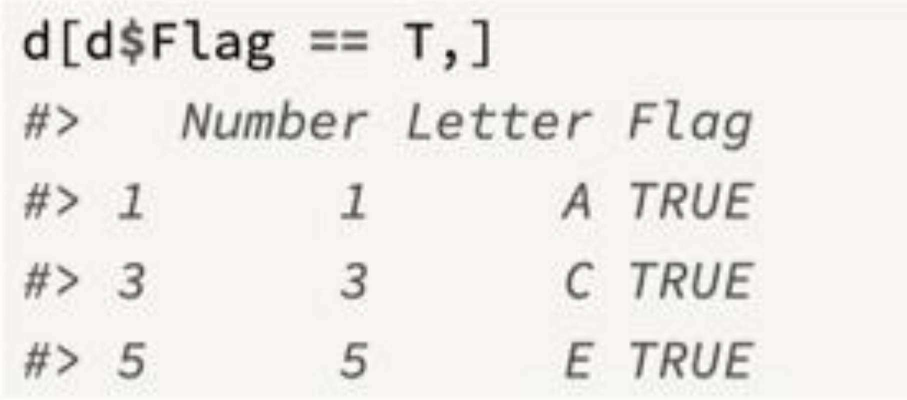
Activities on a data frame

- An important activity that is required with a data frame is to be able to: (1) subset rows, (2) subset columns, and (3) add new columns.
- Because a data frame is a list and also shares properties of a matrix, we can combine subsetting mechanisms from both of these data structures to subset a data frame
- We can access a data frame column using the $ operator.

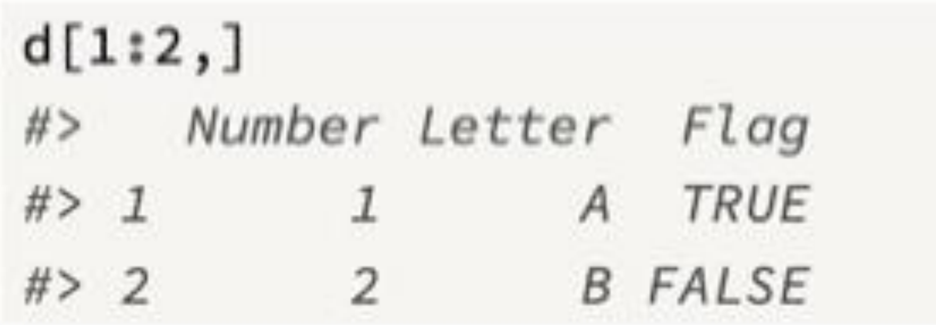
Subsetting examples



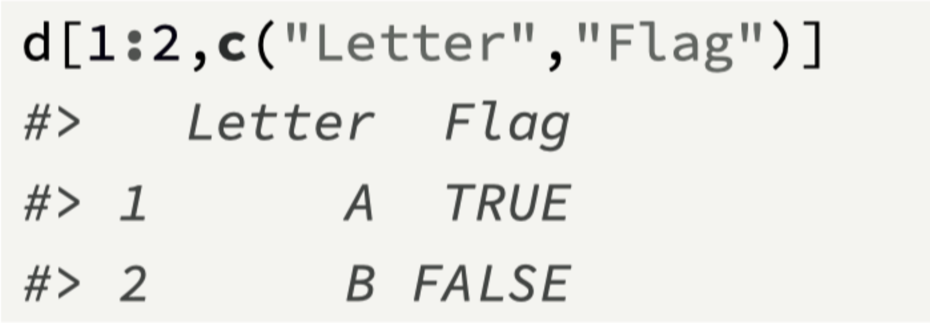
```
d
#>   Number Letter  Flag
#> 1      1      A  TRUE
#> 2      2      B FALSE
#> 3      3      C  TRUE
#> 4      4      D FALSE
#> 5      5      E  TRUE
```



```
d[d$Flag == T,]
#>   Number Letter Flag
#> 1      1      A TRUE
#> 3      3      C TRUE
#> 5      5      E TRUE
```

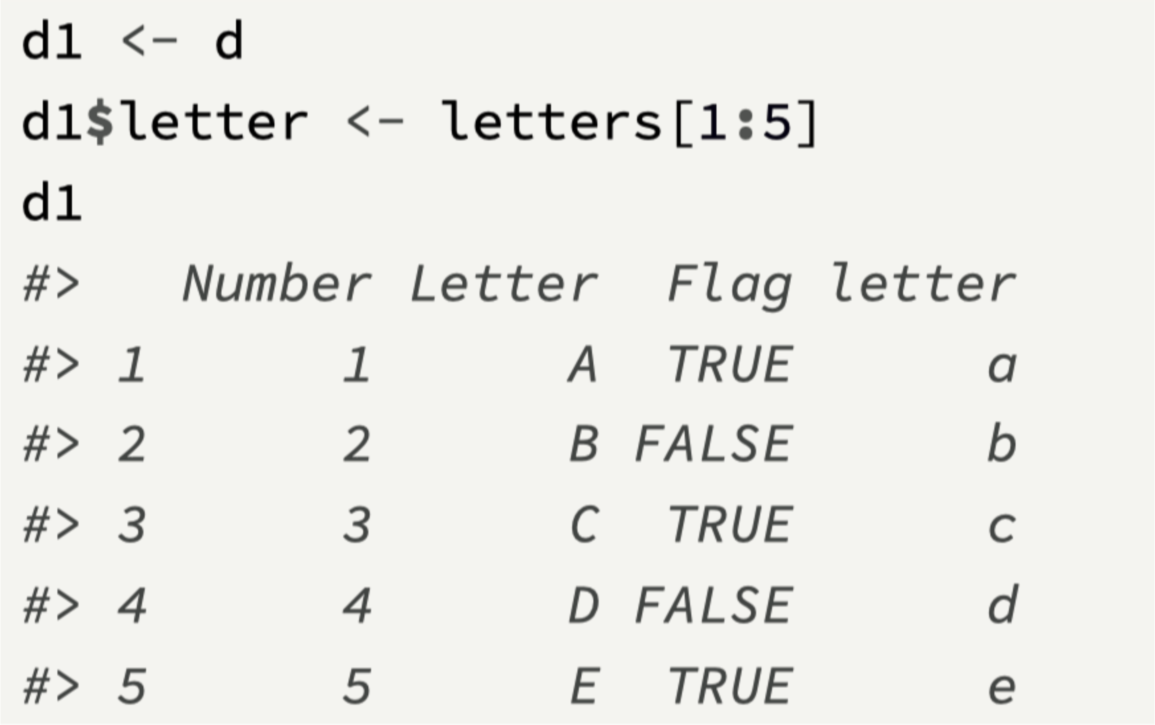


```
d[1:2,]
#>   Number Letter  Flag
#> 1      1      A  TRUE
#> 2      2      B FALSE
```

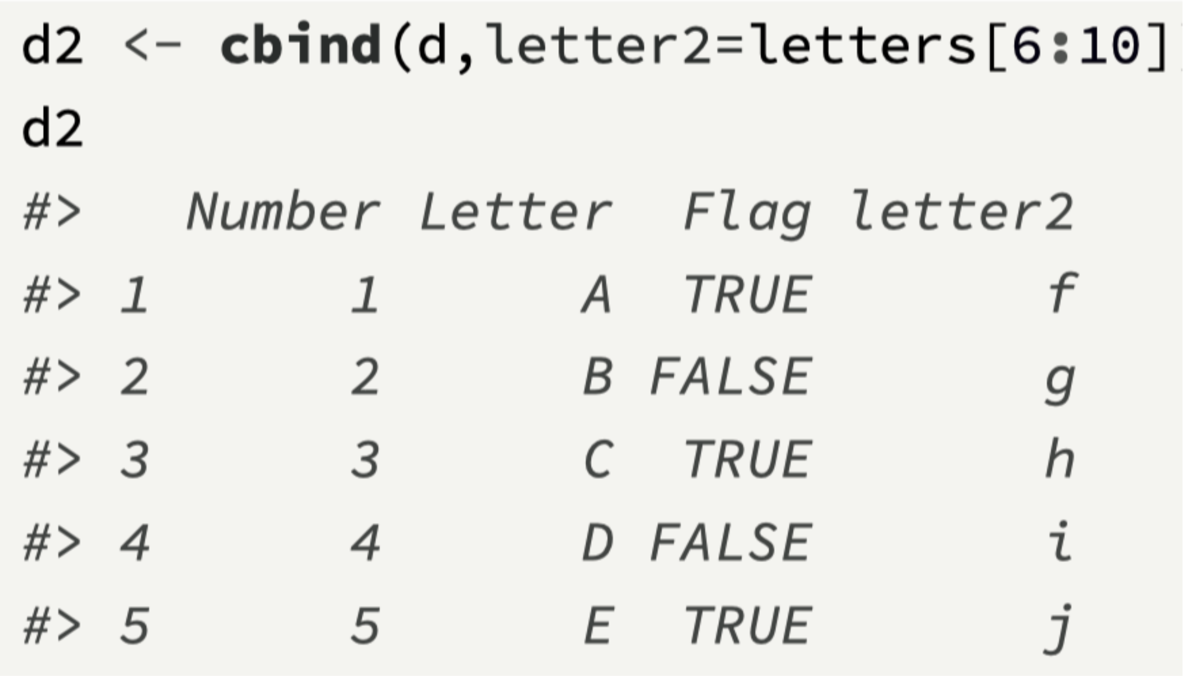


```
d[1:2,c("Letter","Flag")]
#>   Letter  Flag
#> 1      A  TRUE
#> 2      B FALSE
```

Adding a new column



```
d1 <- d
d1$letter <- letters[1:5]
d1
#>   Number Letter  Flag letter
#> 1      1      A  TRUE      a
#> 2      2      B FALSE      b
#> 3      3      C  TRUE      c
#> 4      4      D FALSE      d
#> 5      5      E  TRUE      e
```

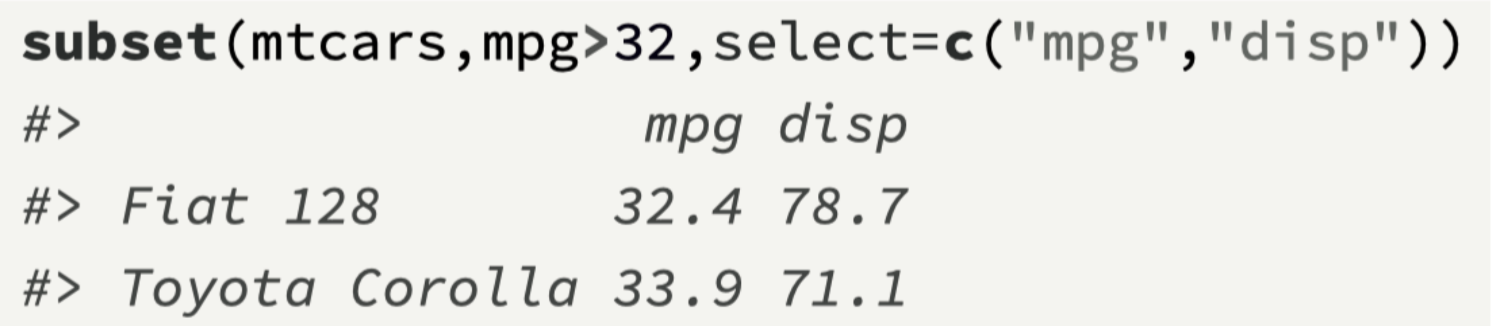


```
d2 <- cbind(d,letter2=letters[6:10]
d2
#>   Number Letter  Flag letter2
#> 1      1      A  TRUE       f
#> 2      2      B FALSE       g
#> 3      3      C  TRUE       h
#> 4      4      D FALSE       i
#> 5      5      E  TRUE       j
```

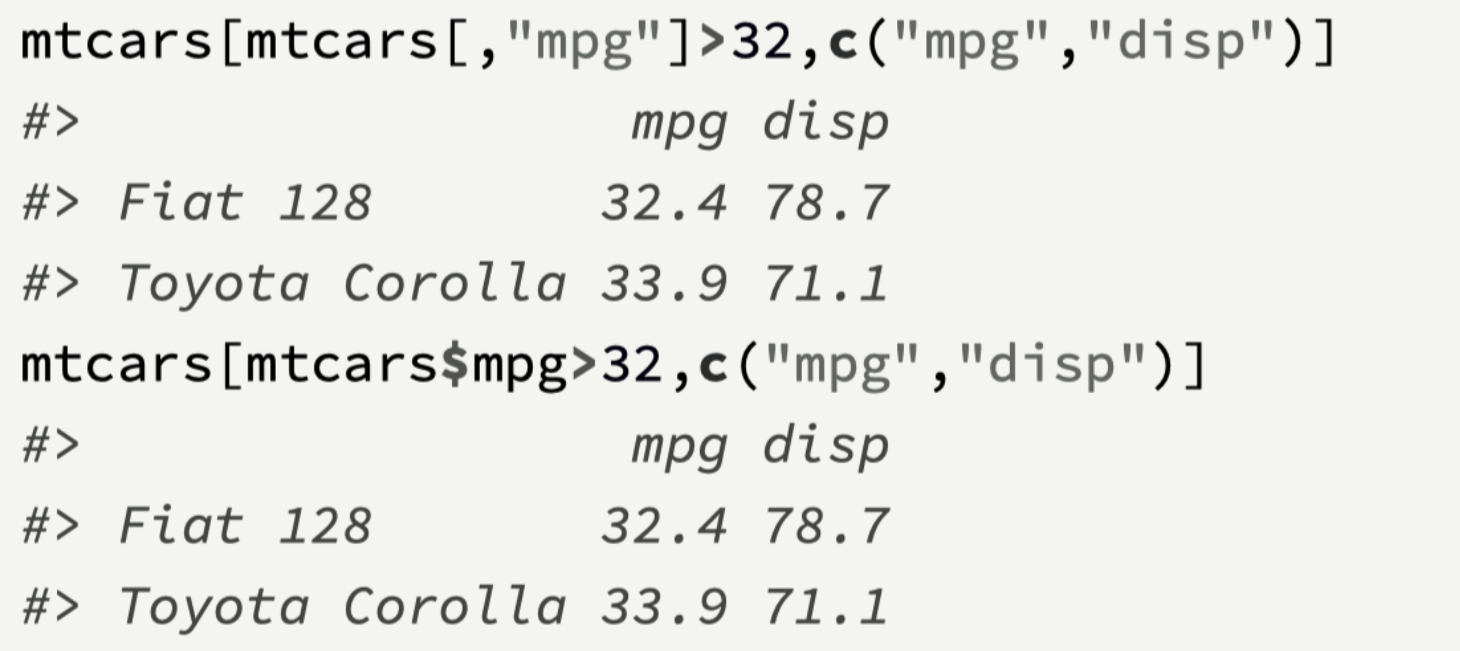
The subset() function

- The function subset(x, subset, select) returns subsets of vectors, matrices, or data frames that meet specified conditions.
- The main arguments to provide when subsetting data frames are:
  - x, the object to be subsetted,
  - subset, a logical expression indicating which rows should be kept,
  - select, which indicates the columns to be selected from the data frame. If this is not present, all columns are returned.

Examples (and alternatives)



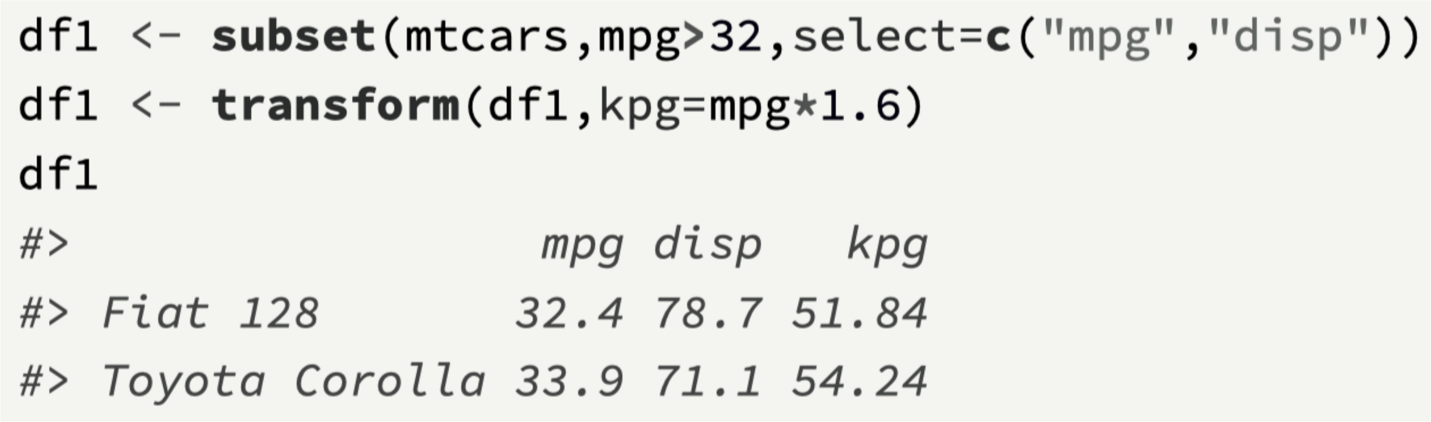
```
subset(mtcars,mpg>32,select=c("mpg","disp"))
#>                 mpg disp
#> Fiat 128       32.4 78.7
#> Toyota Corolla 33.9 71.1
```



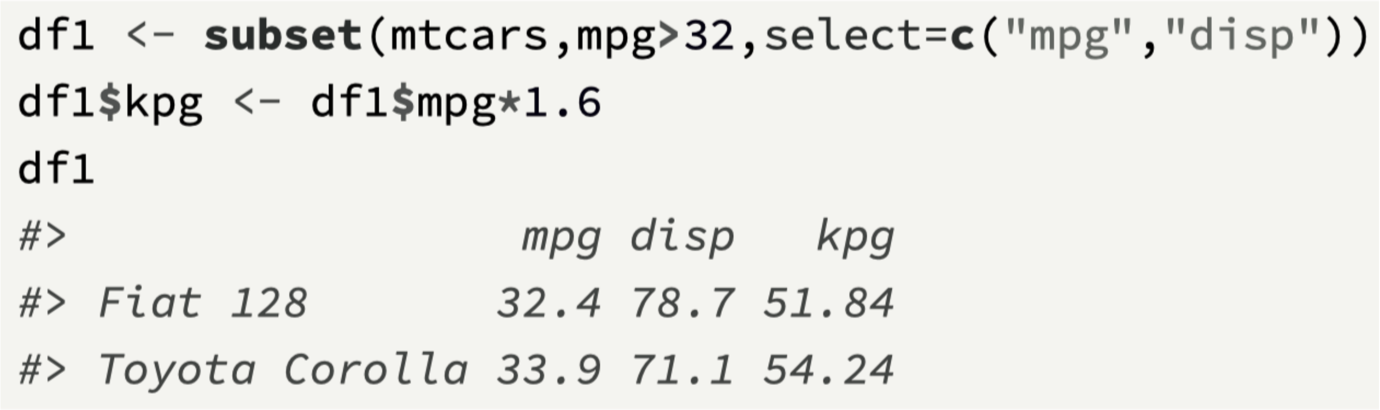
```
mtcars[mtcars[,"mpg"]>32,c("mpg","disp")]
#>                 mpg disp
#> Fiat 128       32.4 78.7
#> Toyota Corolla 33.9 71.1
mtcars[mtcars$mpg>32,c("mpg","disp")]
#>                 mpg disp
#> Fiat 128       32.4 78.7
#> Toyota Corolla 33.9 71.1
```

The transform() function

- A second function that can be used to manipulate data frames is transform(data, ...), which takes in the following arguments:
  - data, which is the data frame,
  - ... which are additional arguments that capture the details of how the new column is created.



```
df1 <- subset(mtcars,mpg>32,select=c("mpg","disp"))
df1 <- transform(df1,kpg=mpg*1.6)
df1
#>                 mpg disp   kpg
#> Fiat 128       32.4 78.7 51.84
#> Toyota Corolla 33.9 71.1 54.24
```

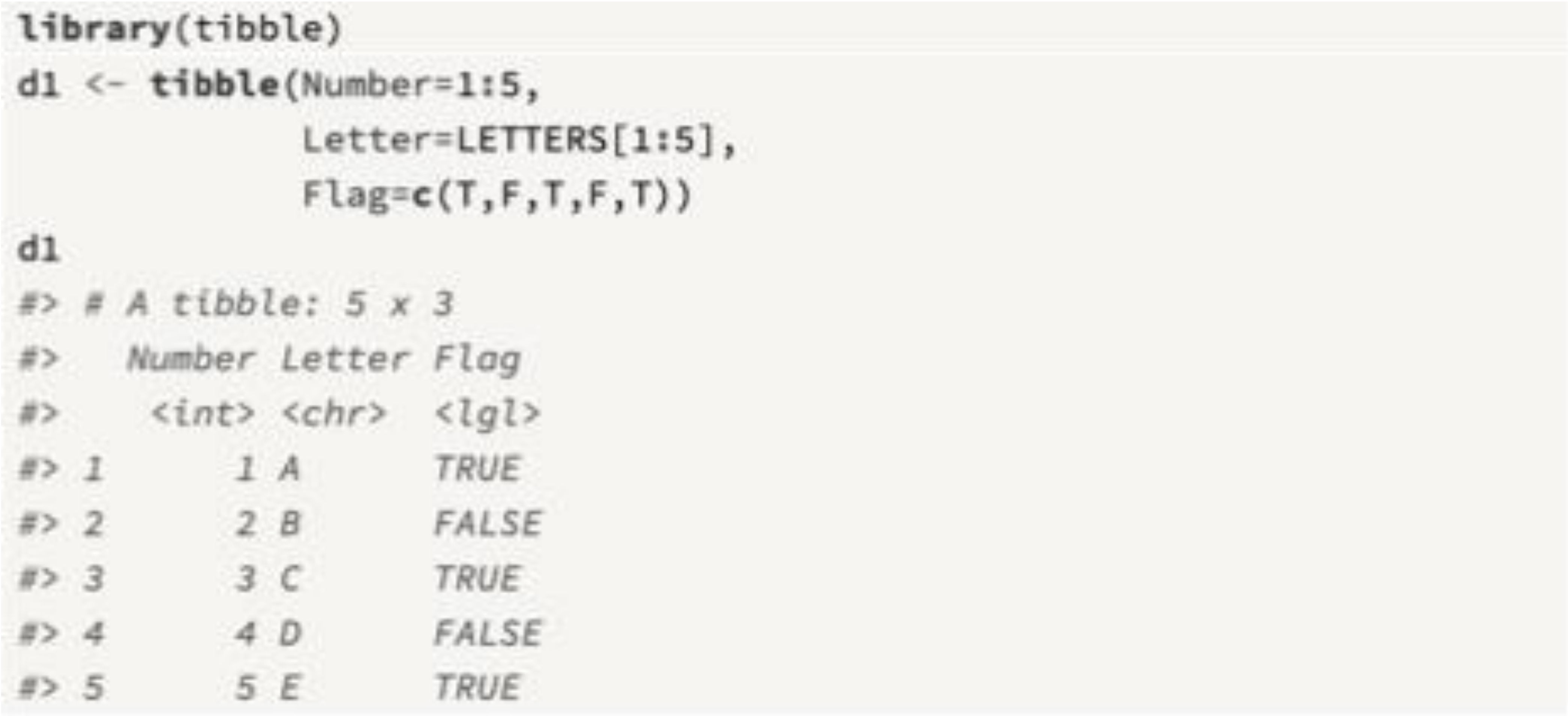


```
df1 <- subset(mtcars,mpg>32,select=c("mpg","disp"))
df1$kpg <- df1$mpg*1.6
df1
#>                 mpg disp   kpg
#> Fiat 128       32.4 78.7 51.84
#> Toyota Corolla 33.9 71.1 54.24
```

(4.4) Tibbles

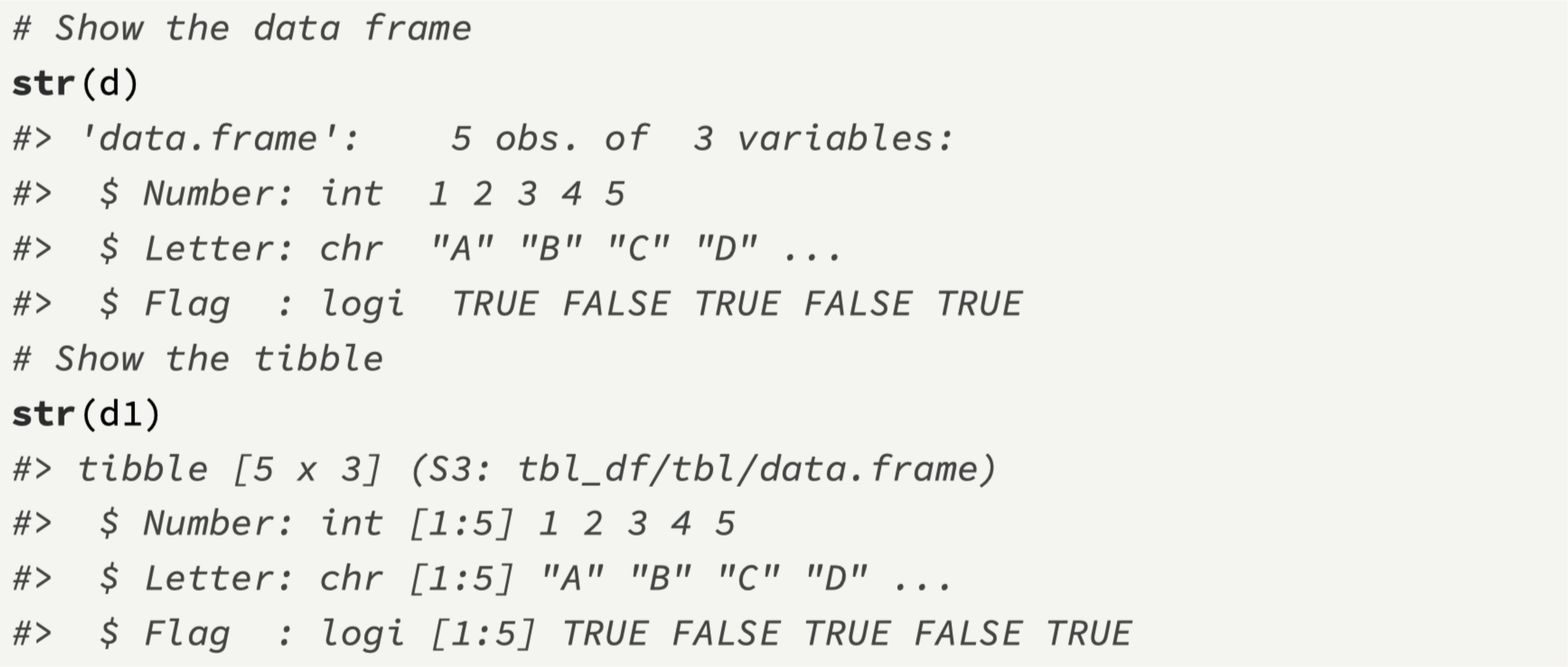
- Tibbles are a type of data frame; however they alter some data frame behaviors:
  - Printing, where tibbles only show the first ten rows, and limit the visible columns to those that fit on the screen. The type is also displayed for each variable.
  - Subsetting, where a tibble is always returned, and also partial matching is not supported.

Previous example – using a tibble



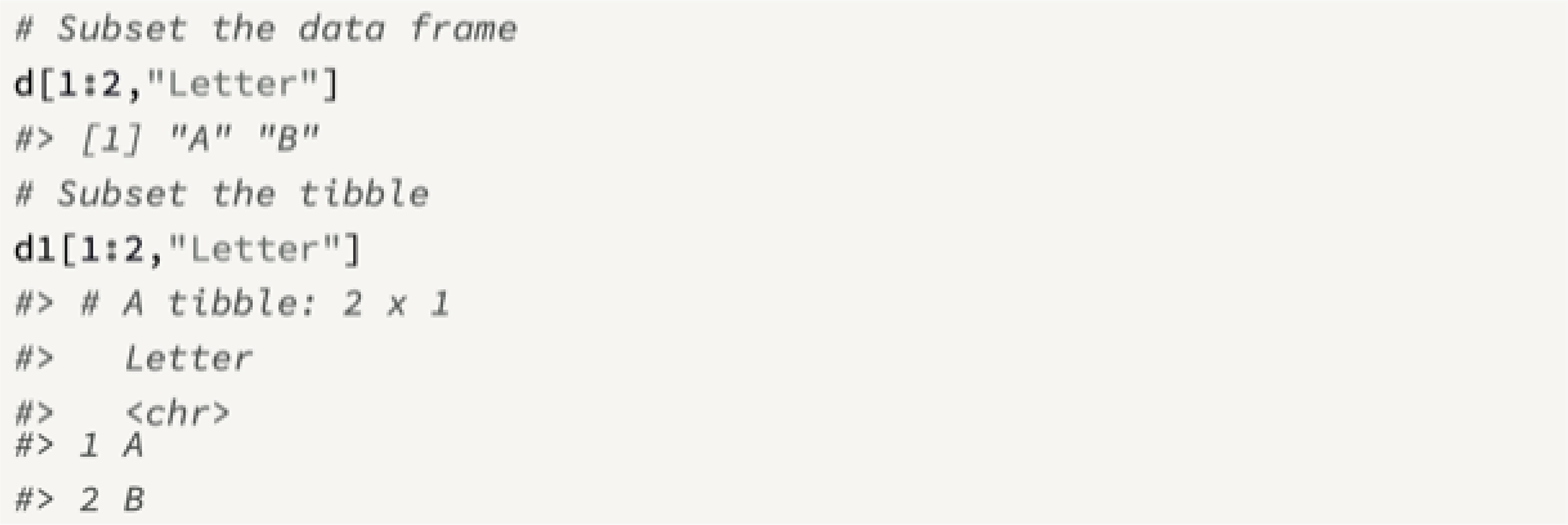
```
library(tibble)
d1 <- tibble(Number=1:5,
             Letter=LETTERS[1:5],
             Flag=c(T,F,T,F,T))
d1
#> # A tibble: 5 x 3
#>   Number Letter Flag
#>    <int> <chr>  <lgl>
#> 1      1 A      TRUE
#> 2      2 B      FALSE
#> 3      3 C      TRUE
#> 4      4 D      FALSE
#> 5      5 E      TRUE
```

Differences with data.frame



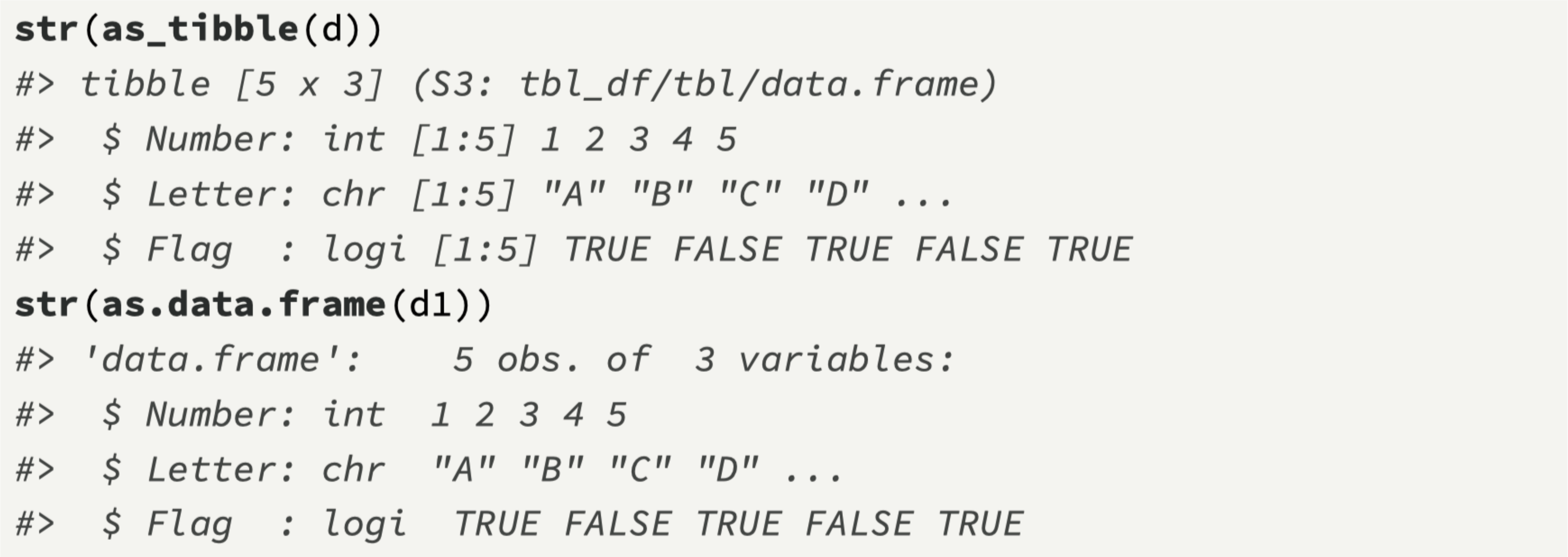
```
# Show the data frame
str(d)
#> 'data.frame':    5 obs. of  3 variables:
#>  $ Number: int  1 2 3 4 5
#>  $ Letter: chr  "A" "B" "C" "D" ...
#>  $ Flag  : logi  TRUE FALSE TRUE FALSE TRUE
# Show the tibble
str(d1)
#> tibble [5 x 3] (S3: tbl_df/tbl/data.frame)
#>  $ Number: int [1:5] 1 2 3 4 5
#>  $ Letter: chr [1:5] "A" "B" "C" "D" ...
#>  $ Flag  : logi [1:5] TRUE FALSE TRUE FALSE TRUE
```

Subsetting differences



```
# Subset the data frame
d[1:2,"Letter"]
#> [1] "A" "B"
# Subset the tibble
d1[1:2,"Letter"]
#> # A tibble: 2 x 1
#>   Letter
#>   <chr>
#> 1 A
#> 2 B
```

Moving between two types

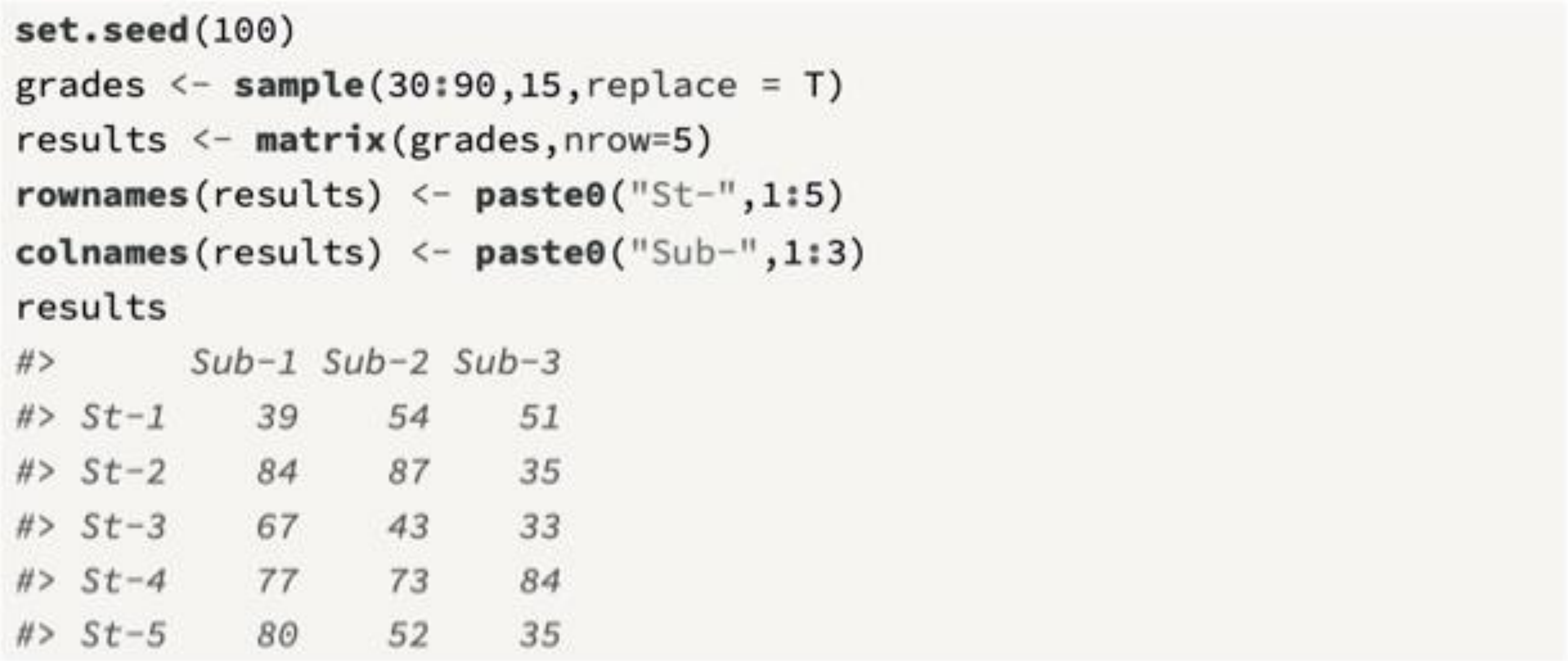


```
str(as_tibble(d))
#> tibble [5 x 3] (S3: tbl_df/tbl/data.frame)
#>  $ Number: int [1:5] 1 2 3 4 5
#>  $ Letter: chr [1:5] "A" "B" "C" "D" ...
#>  $ Flag  : logi [1:5] TRUE FALSE TRUE FALSE TRUE
str(as.data.frame(d1))
#> 'data.frame':    5 obs. of  3 variables:
#>  $ Number: int  1 2 3 4 5
#>  $ Letter: chr  "A" "B" "C" "D" ...
#>  $ Flag  : logi  TRUE FALSE TRUE FALSE TRUE
```

(4.5) Functionals on matrices and data frames

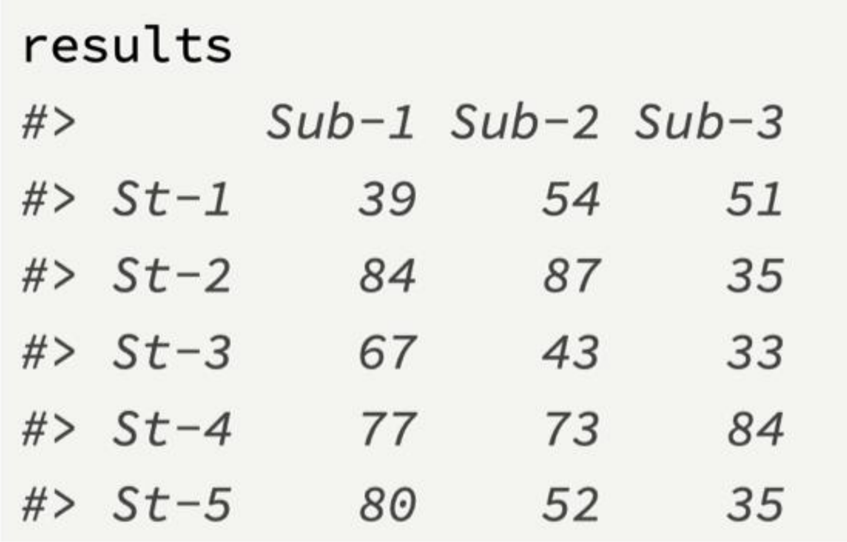
- The apply(x,margin,f) function (similar to lapply()) is a functional used to iterate over matrices and data frames, and it accepts the following arguments:
  - x, which can be a matrix or a data frame.
  - margin, a number that indicates whether the iteration is by row (margin=1), or by column (margin=2).
  - f, which is the function to be applied during each iteration.

Using apply() – analysing grades

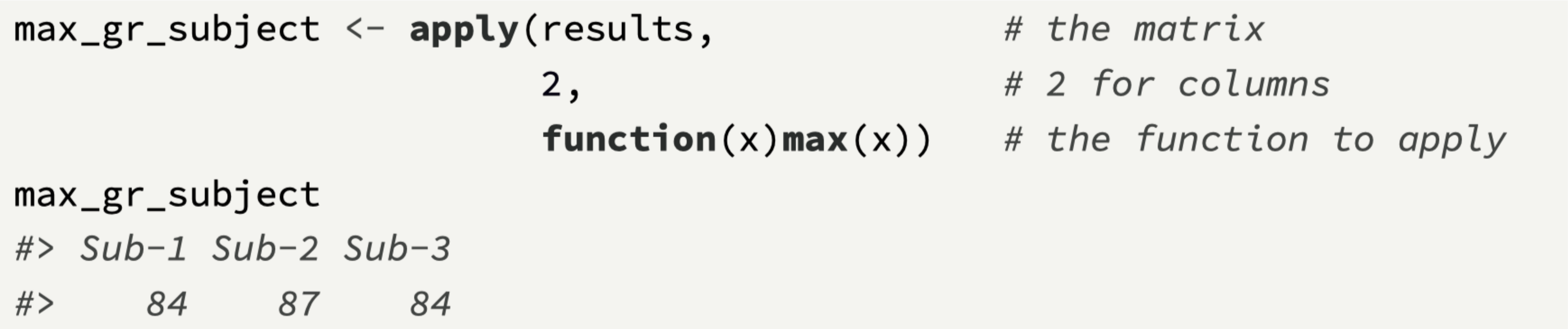


```
set.seed(100)
grades <- sample(30:90,15,replace = T)
results <- matrix(grades,nrow=5)
rownames(results) <- paste0("St-",1:5)
colnames(results) <- paste0("Sub-",1:3)
results
#>       Sub-1 Sub-2 Sub-3
#> St-1     39    54    51
#> St-2     84    87    35
#> St-3     67    43    33
#> St-4     77    73    84
#> St-5     80    52    35
```

Maximum Grade for each subject

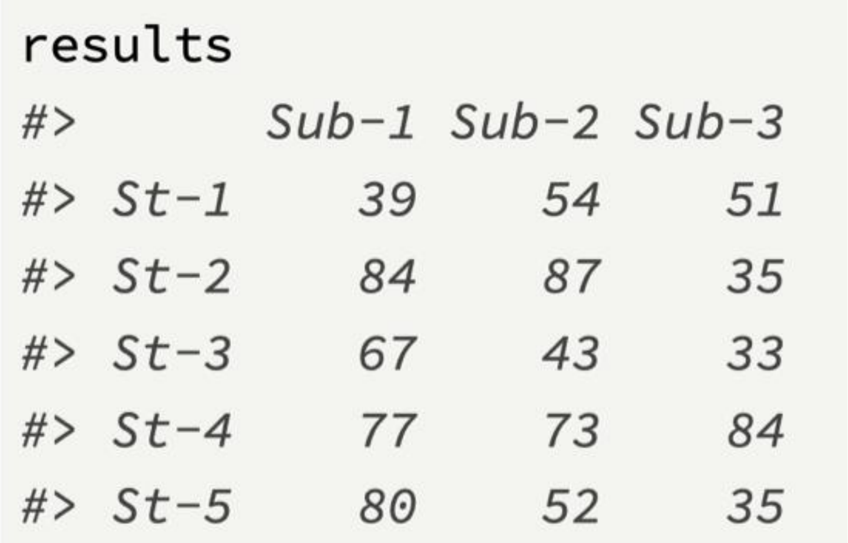


```
results
#>       Sub-1 Sub-2 Sub-3
#> St-1     39    54    51
#> St-2     84    87    35
#> St-3     67    43    33
#> St-4     77    73    84
#> St-5     80    52    35
```

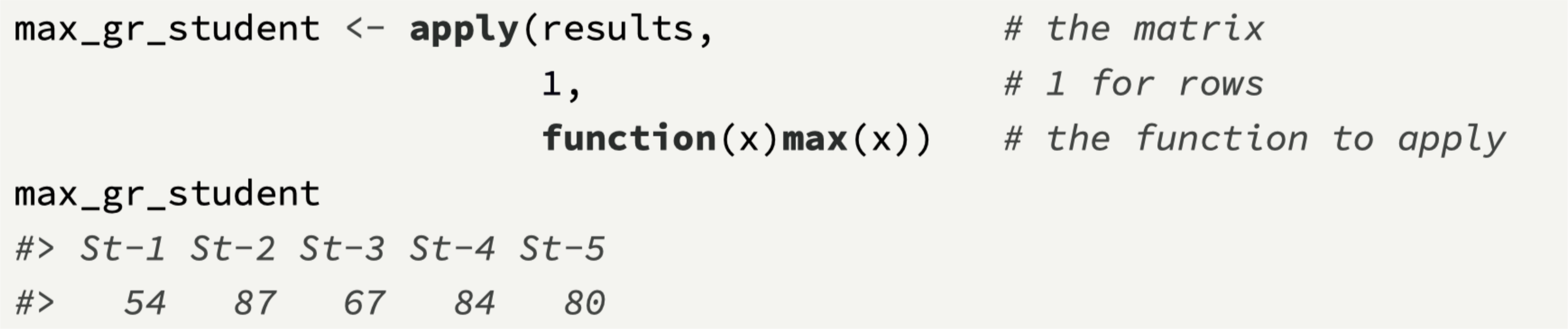


```
max_gr_subject <- apply(results,              # the matrix
                        2,                    # 2 for columns
                        function(x)max(x))    # the function to apply
max_gr_subject
#> Sub-1 Sub-2 Sub-3
#>    84    87    84
```

Maximum Grade for each student

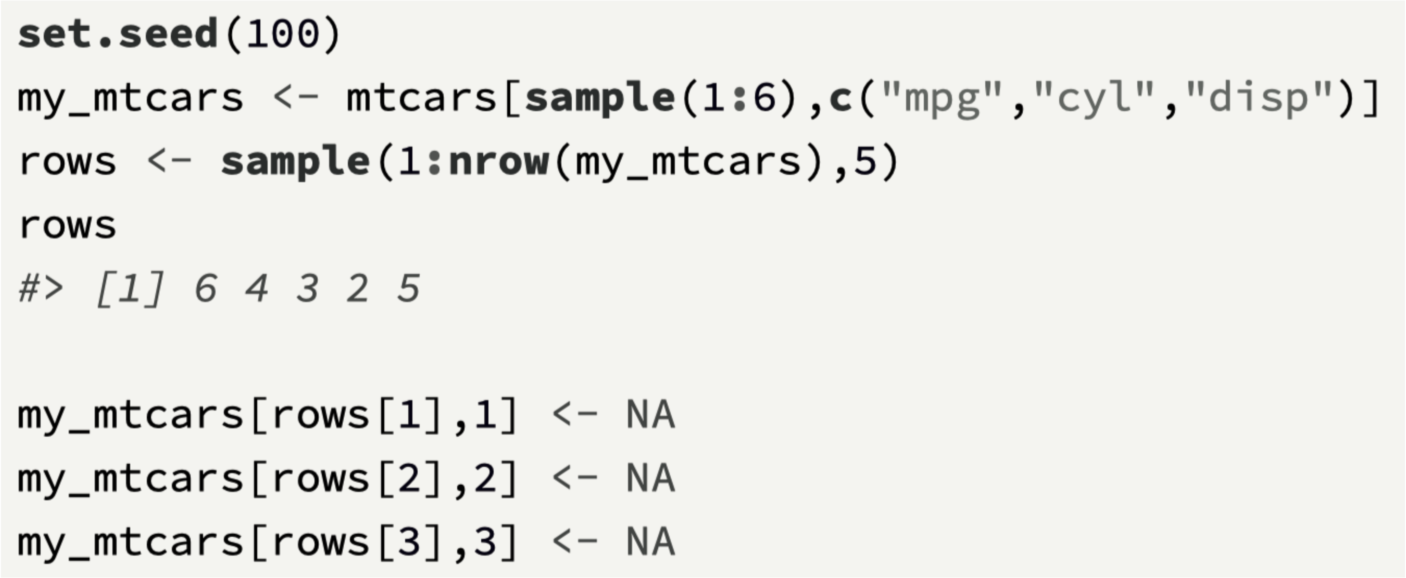


```
results
#>       Sub-1 Sub-2 Sub-3
#> St-1     39    54    51
#> St-2     84    87    35
#> St-3     67    43    33
#> St-4     77    73    84
#> St-5     80    52    35
```

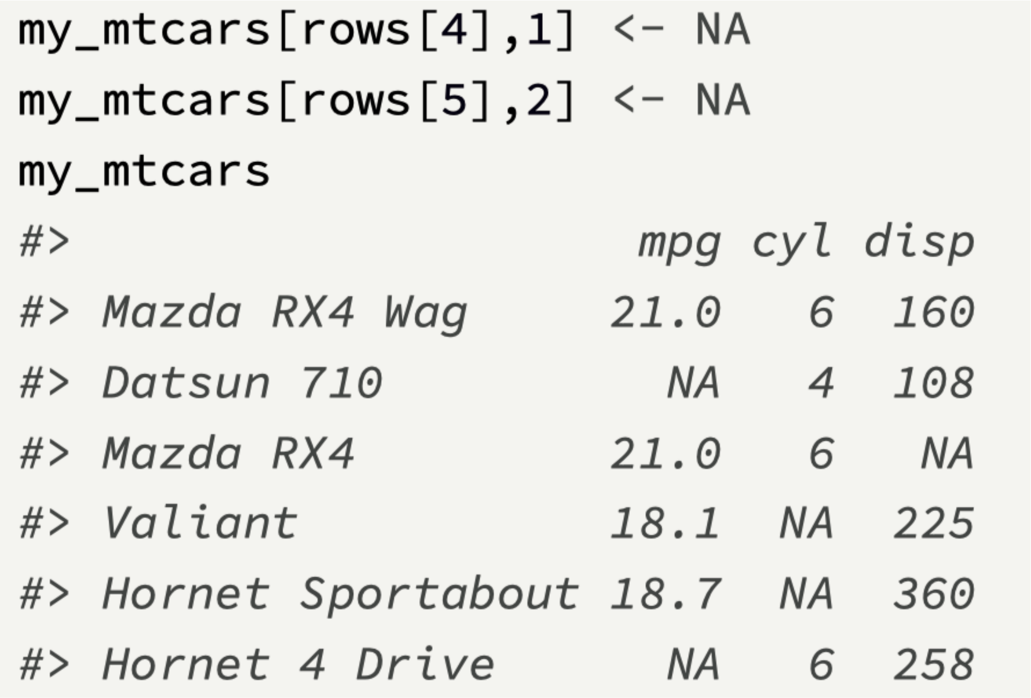


```
max_gr_student <- apply(results,              # the matrix
                        1,                    # 1 for rows
                        function(x)max(x))    # the function to apply
max_gr_student
#> St-1 St-2 St-3 St-4 St-5
#>   54   87   67   84   80
```

Using apply() on data frames

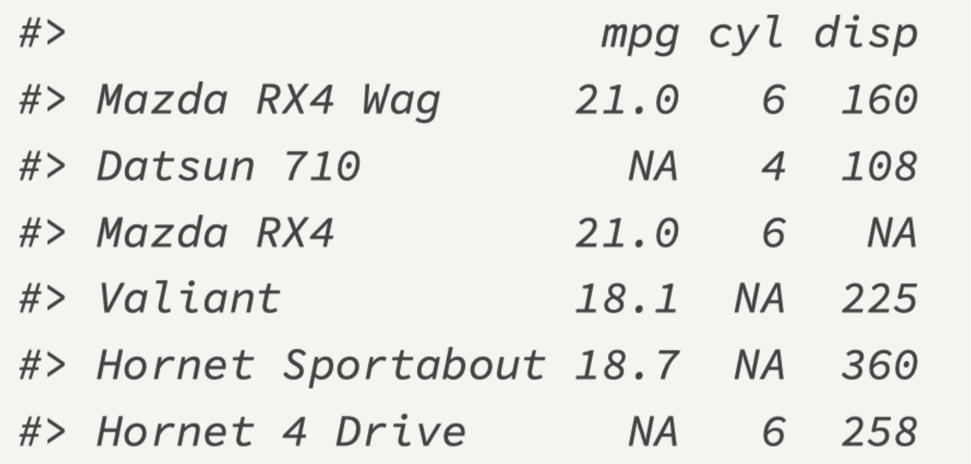


```
set.seed(100)
my_mtcars <- mtcars[sample(1:6),c("mpg","cyl","disp")]
rows <- sample(1:nrow(my_mtcars),5)
rows
#> [1] 6 4 3 2 5

my_mtcars[rows[1],1] <- NA
my_mtcars[rows[2],2] <- NA
my_mtcars[rows[3],3] <- NA
```

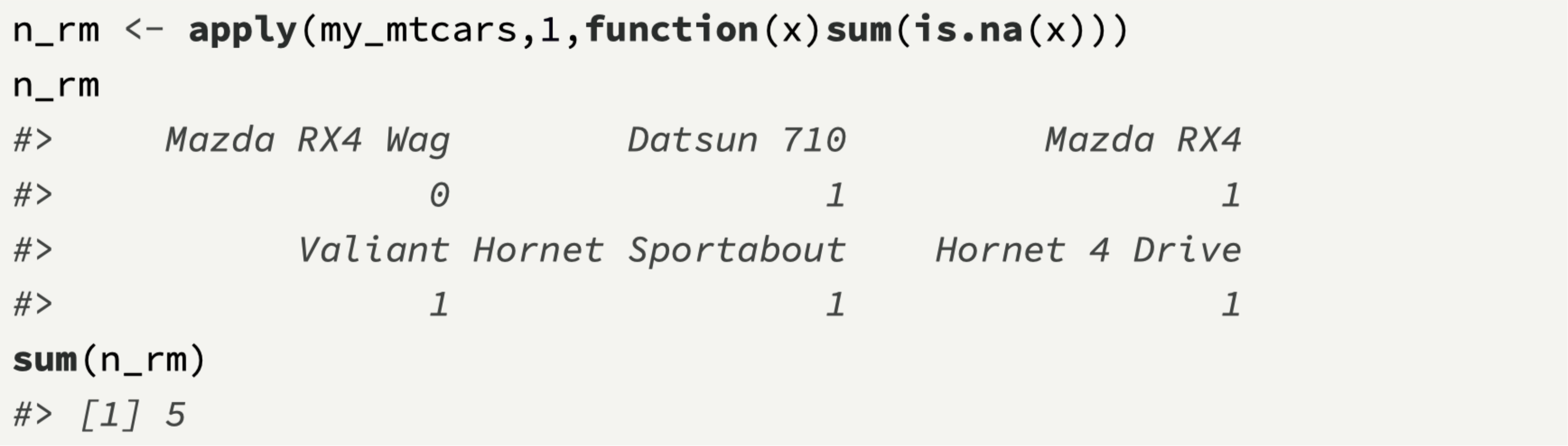


```
my_mtcars[rows[4],1] <- NA
my_mtcars[rows[5],2] <- NA
my_mtcars
#>                    mpg cyl disp
#> Mazda RX4 Wag     21.0   6  160
#> Datsun 710          NA   4  108
#> Mazda RX4         21.0   6   NA
#> Valiant           18.1  NA  225
#> Hornet Sportabout 18.7  NA  360
#> Hornet 4 Drive      NA   6  258
```

Count number of missing values by row

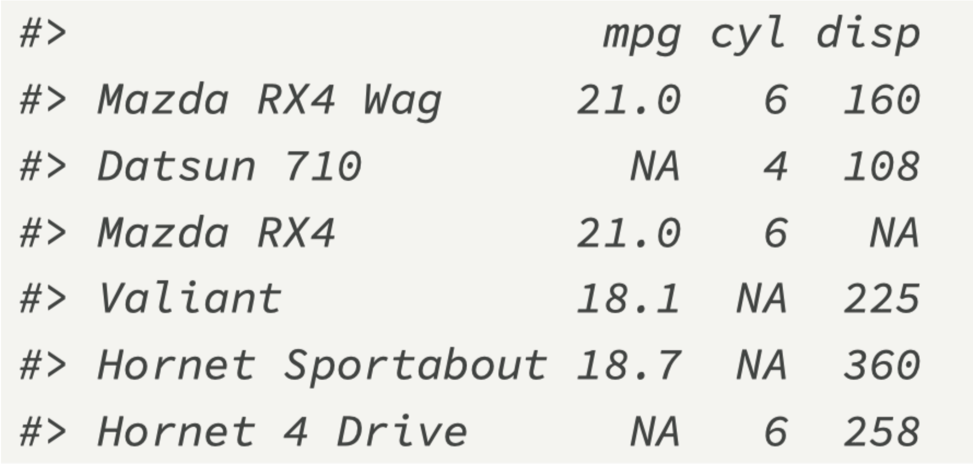


```
#>                    mpg cyl disp
#> Mazda RX4 Wag     21.0   6  160
#> Datsun 710          NA   4  108
#> Mazda RX4         21.0   6   NA
#> Valiant           18.1  NA  225
#> Hornet Sportabout 18.7  NA  360
#> Hornet 4 Drive      NA   6  258
```

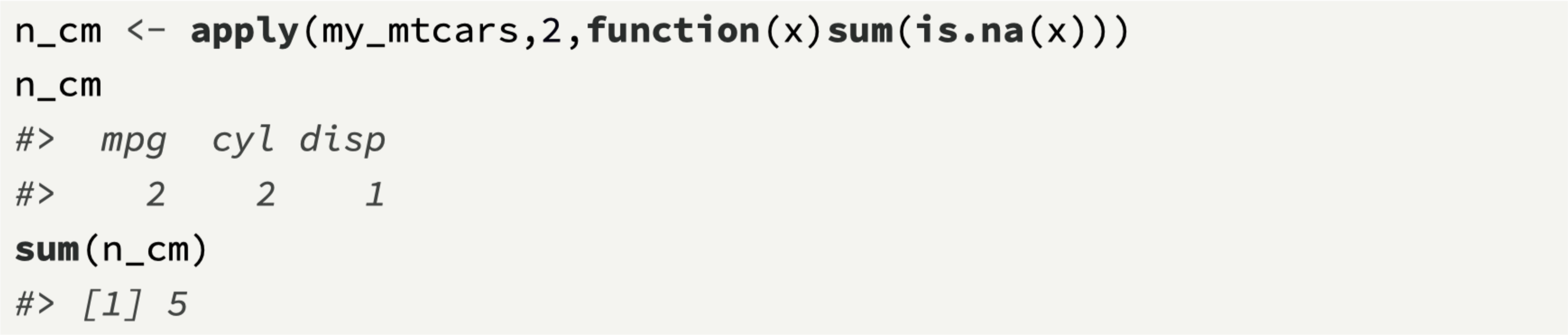


```
n_rm <- apply(my_mtcars,1,function(x)sum(is.na(x)))
n_rm
#>     Mazda RX4 Wag        Datsun 710         Mazda RX4
#>                 0                 1                 1
#>           Valiant Hornet Sportabout    Hornet 4 Drive
#>                 1                 1                 1
sum(n_rm)
#> [1] 5
```

Count number of missing values by column



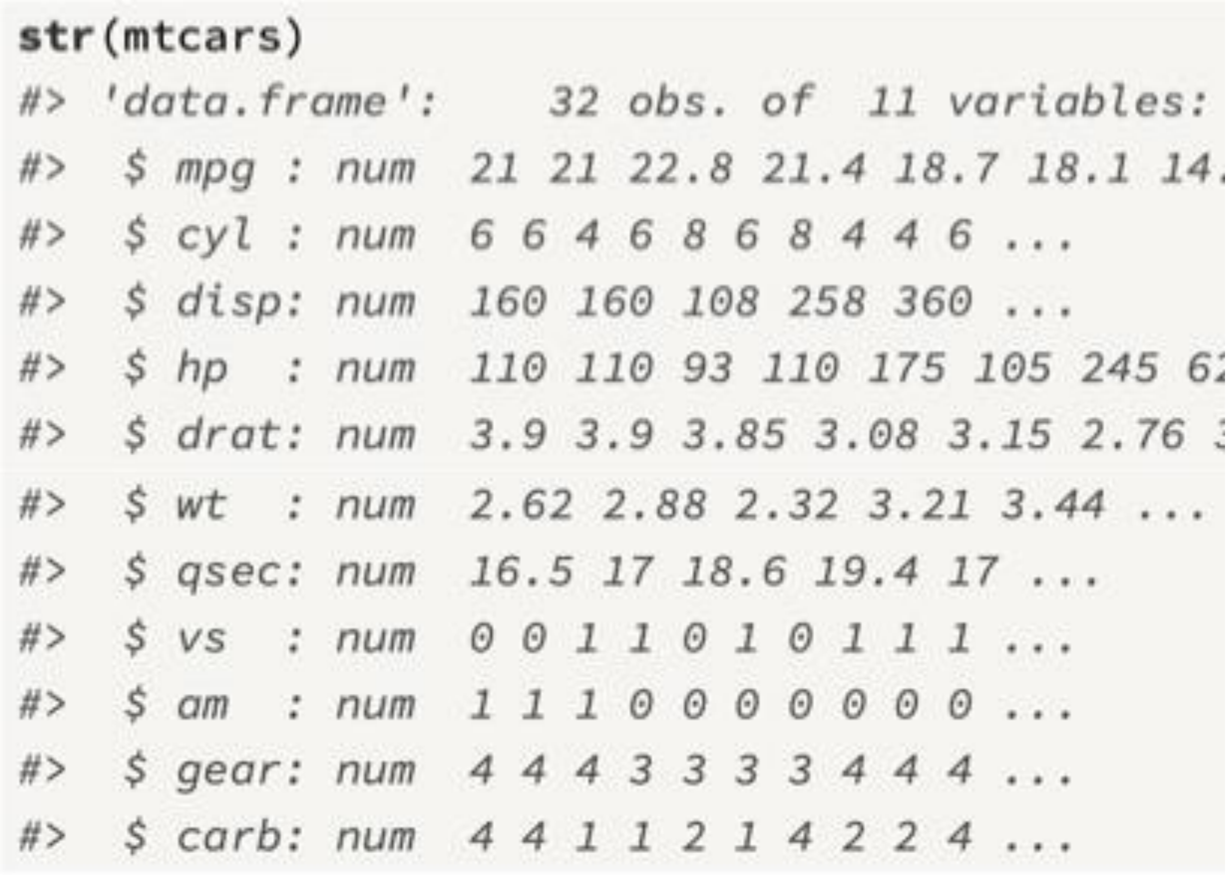
```
#>                    mpg cyl disp
#> Mazda RX4 Wag     21.0   6  160
#> Datsun 710          NA   4  108
#> Mazda RX4         21.0   6   NA
#> Valiant           18.1  NA  225
#> Hornet Sportabout 18.7  NA  360
#> Hornet 4 Drive      NA   6  258
```



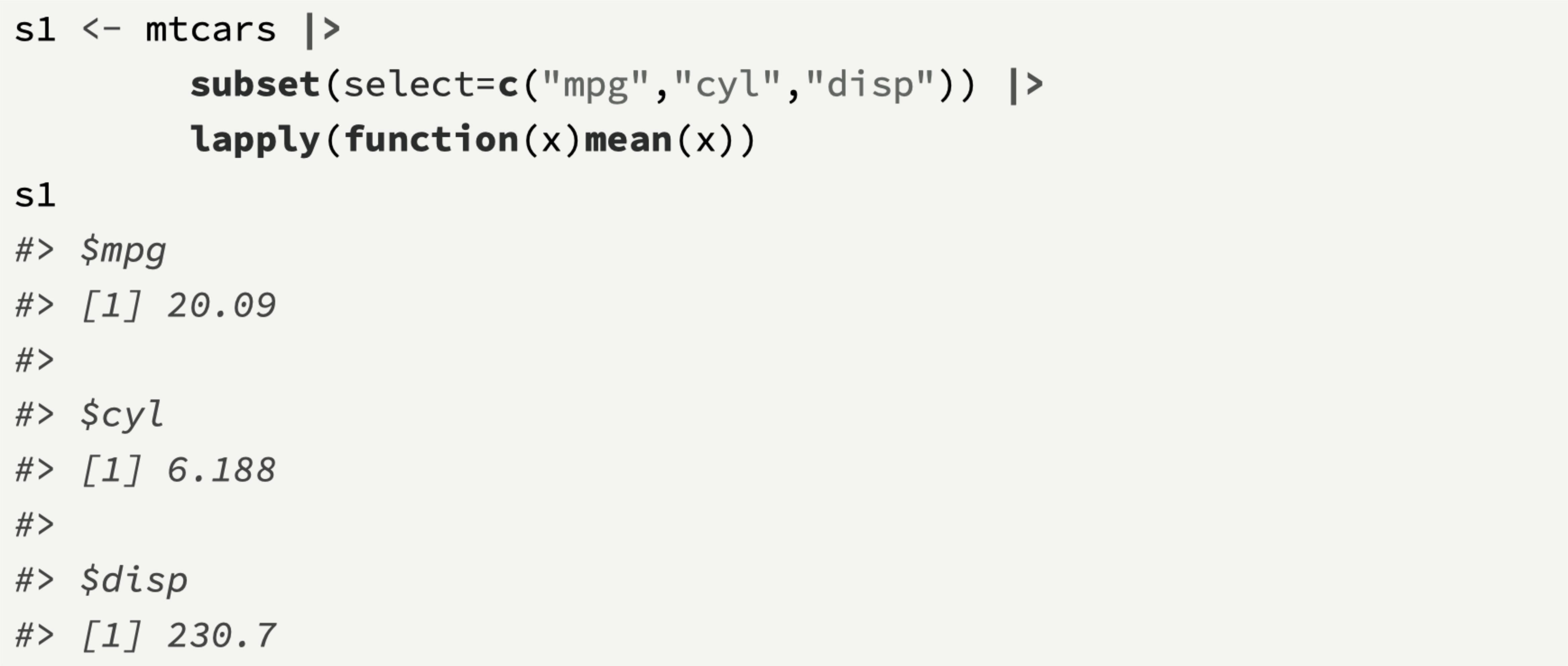
```
n_cm <- apply(my_mtcars,2,function(x)sum(is.na(x)))
n_cm
#>  mpg  cyl disp
#>    2    2    1
sum(n_cm)
#> [1] 5
```

Using lapply() on data frames

- Given that a data frame is also a list, and that lapply() processes lists, it also means that the lapply() functional can be used to process a data frame.
- When processing data frames with lapply(), the most important thing to remember is that the data frame will be processed ***column-by-column*** (e.g. variable)



```
str(mtcars)
#> 'data.frame':    32 obs. of  11 variables:
#>  $ mpg : num  21 21 22.8 21.4 18.7 18.1 14.
#>  $ cyl : num  6 6 4 6 8 6 8 4 4 6 ...
#>  $ disp: num  160 160 108 258 360 ...
#>  $ hp  : num  110 110 93 110 175 105 245 6
#>  $ drat: num  3.9 3.9 3.85 3.08 3.15 2.76 3
#>  $ wt  : num  2.62 2.88 2.32 3.21 3.44 ...
#>  $ qsec: num  16.5 17 18.6 19.4 17 ...
#>  $ vs  : num  0 0 1 1 0 1 0 1 1 1 ...
#>  $ am  : num  1 1 1 0 0 0 0 0 0 0 ...
#>  $ gear: num  4 4 4 3 3 3 3 4 4 4 ...
#>  $ carb: num  4 4 1 1 2 1 4 2 2 4 ...
```

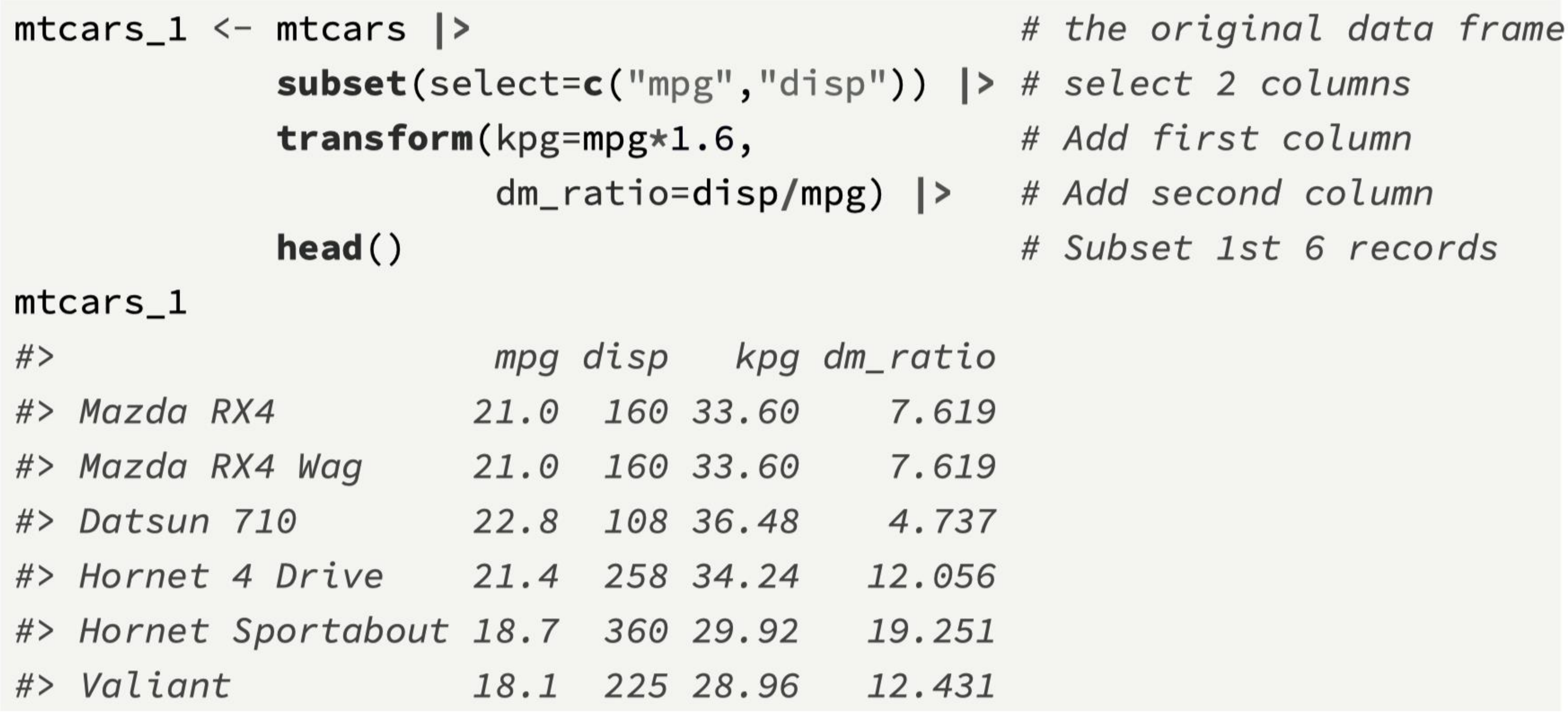


```r
s1 <- mtcars |>
        subset(select=c("mpg","cyl","disp")) |>
        lapply(function(x)mean(x))
s1
#> $mpg
#> [1] 20.09
#>
#> $cyl
#> [1] 6.188
#>
#> $disp
#> [1] 230.7
```

(4.6) Mini-case: Creating a pipeline for processing data frames

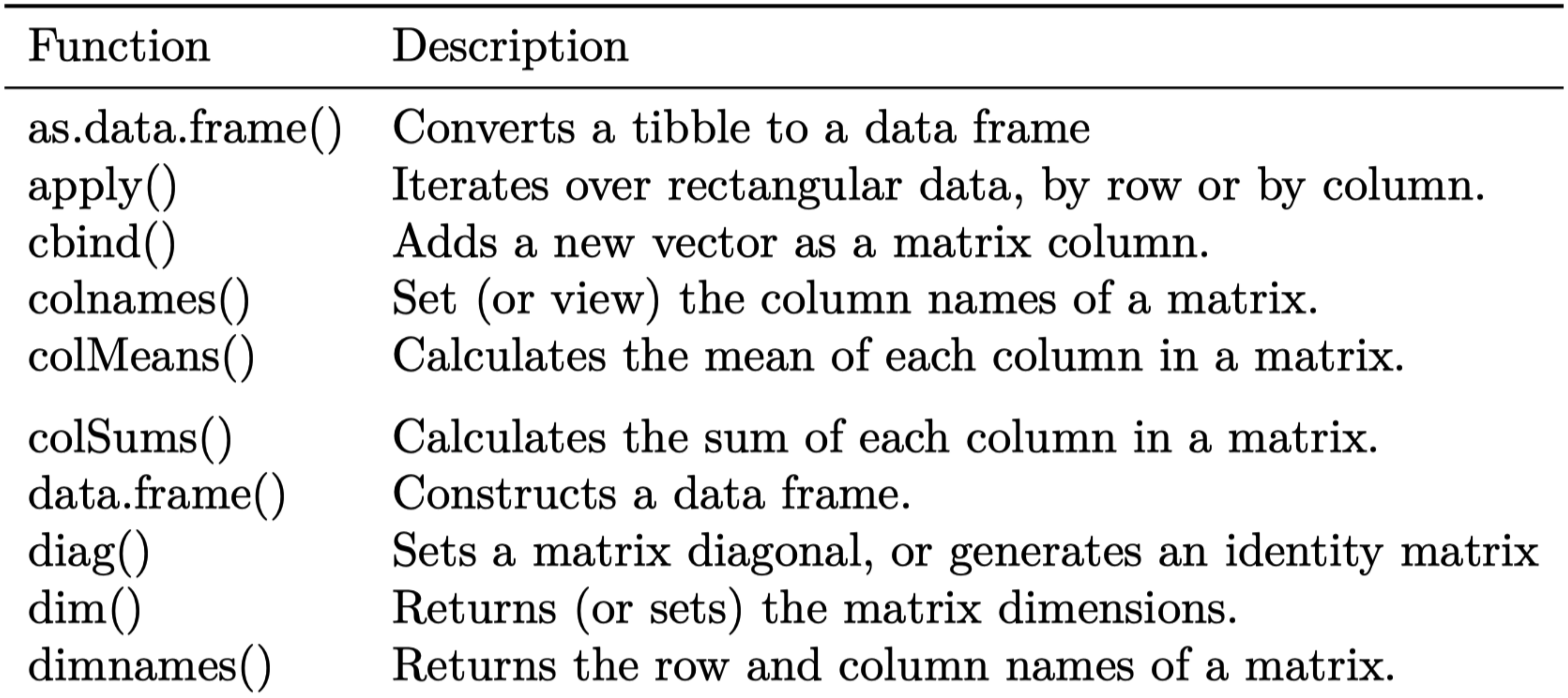
- For the data frame mtcars, the following processing actions will be taken:
  - Two columns from mtcars will be selected, mpg and disp.
  - A new column kpg will be added, which converts mpg to kilometers per gallon, using the multiplier 1.6.
  - A new column dm_ratio will be added, which is the ratio of disp and mpg.
  - The first six observations will then be shown.

Solution pipeline

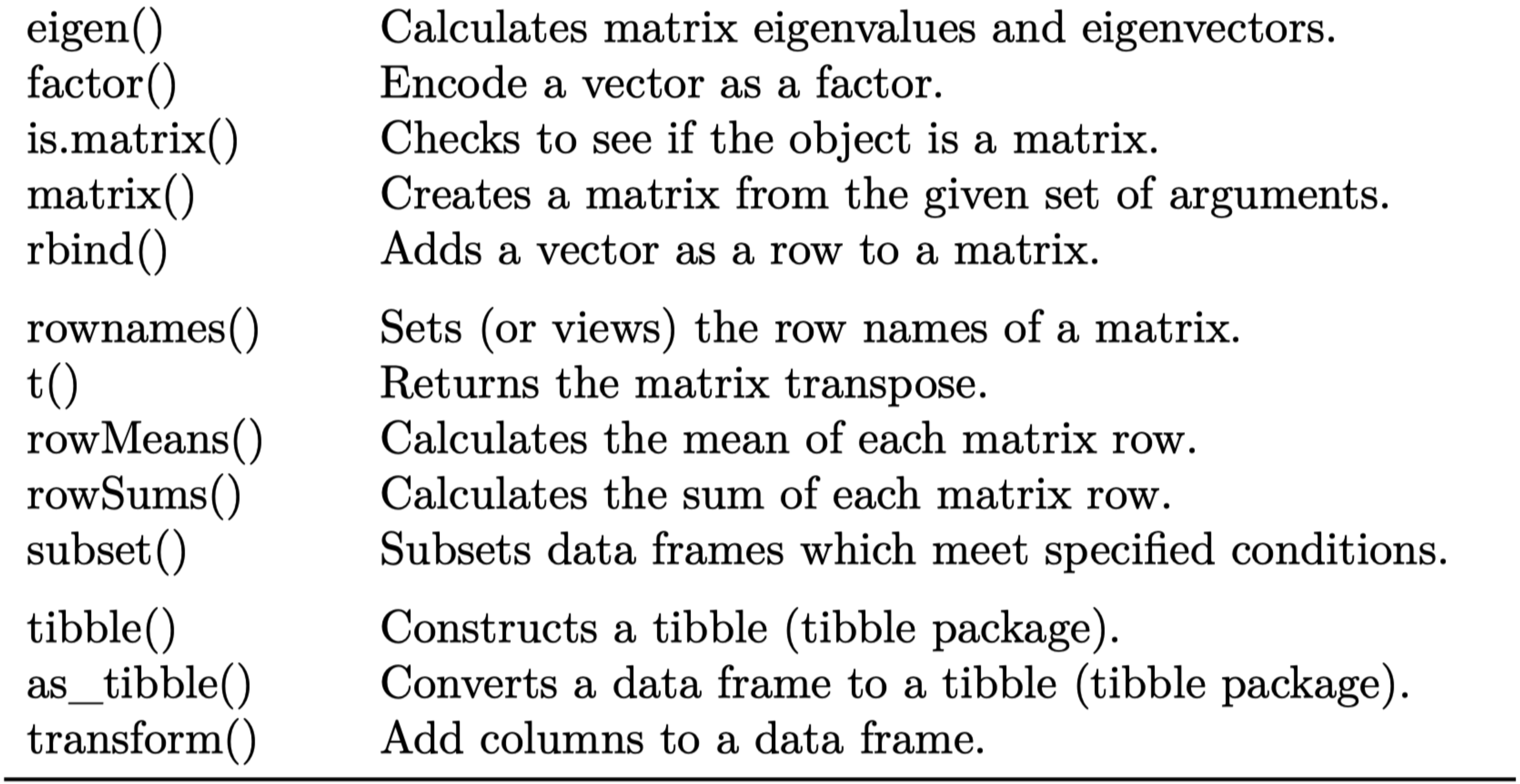


```
mtcars_1 <- mtcars |>                         # the original data frame
            subset(select=c("mpg","disp")) |> # select 2 columns
            transform(kpg=mpg*1.6,            # Add first column
                      dm_ratio=disp/mpg) |>   # Add second column
            head()                            # Subset 1st 6 records
mtcars_1
#>                    mpg disp   kpg dm_ratio
#> Mazda RX4         21.0  160 33.60    7.619
#> Mazda RX4 Wag     21.0  160 33.60    7.619
#> Datsun 710        22.8  108 36.48    4.737
#> Hornet 4 Drive    21.4  258 34.24   12.056
#> Hornet Sportabout 18.7  360 29.92   19.251
#> Valiant           18.1  225 28.96   12.431
```

(4.7) Summary Functions



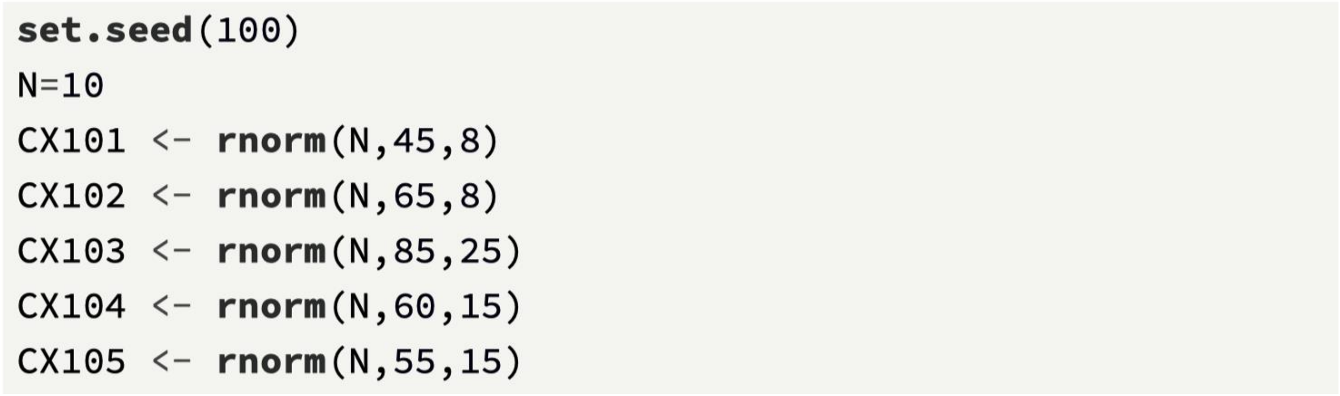
Function	Description
as.data.frame()	Converts a tibble to a data frame
apply()	Iterates over rectangular data, by row or by column.
cbind()	Adds a new vector as a matrix column.
colnames()	Set (or view) the column names of a matrix.
colMeans()	Calculates the mean of each column in a matrix.
colSums()	Calculates the sum of each column in a matrix.
data.frame()	Constructs a data frame.
diag()	Sets a matrix diagonal, or generates an identity matrix
dim()	Returns (or sets) the matrix dimensions.
dimnames()	Returns the row and column names of a matrix.



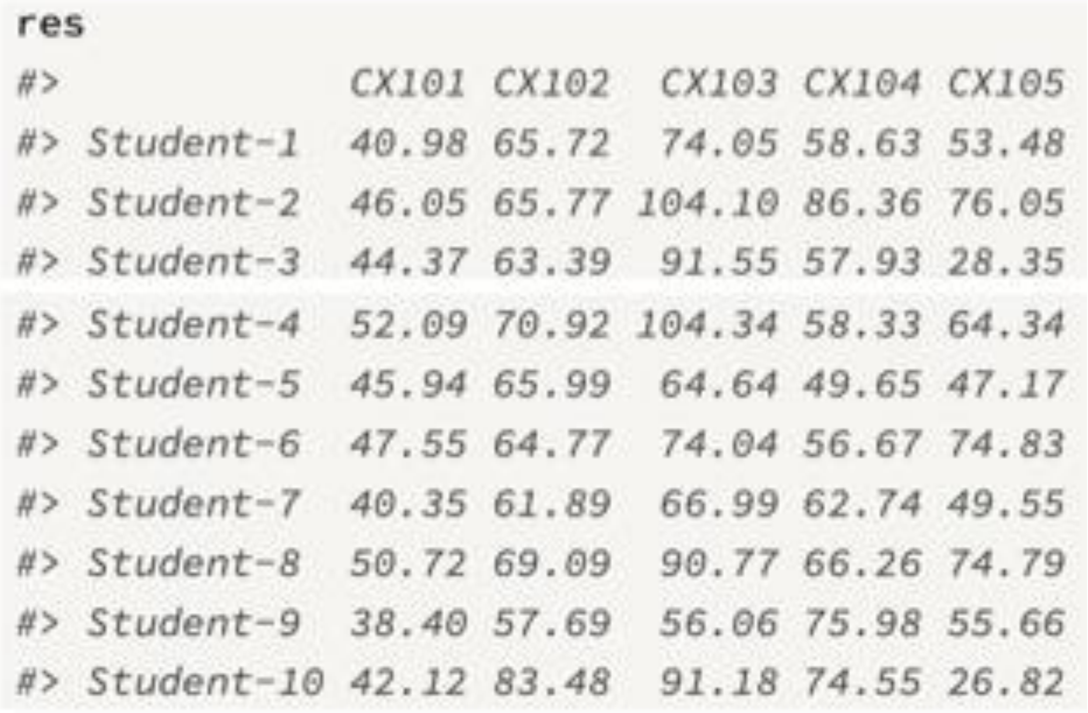
eigen()	Calculates matrix eigenvalues and eigenvectors.
factor()	Encode a vector as a factor.
is.matrix()	Checks to see if the object is a matrix.
matrix()	Creates a matrix from the given set of arguments.
rbind()	Adds a vector as a row to a matrix.
rownames()	Sets (or views) the row names of a matrix.
t()	Returns the matrix transpose.
rowMeans()	Calculates the mean of each matrix row.
rowSums()	Calculates the sum of each matrix row.
subset()	Subsets data frames which meet specified conditions.
tibble()	Constructs a tibble (tibble package).
as_tibble()	Converts a data frame to a tibble (tibble package).
transform()	Add columns to a data frame.

(4.8) Exercises

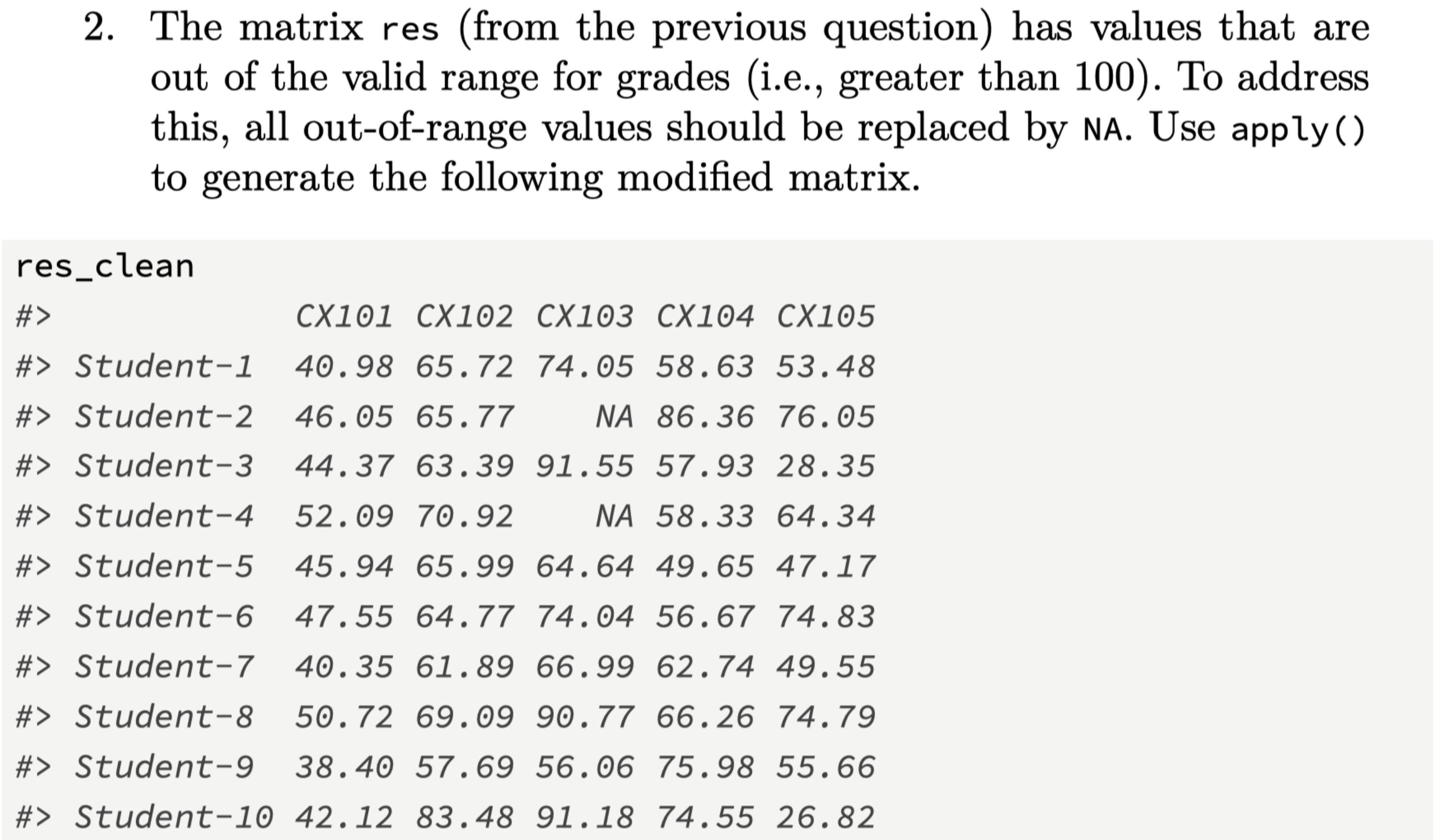
1. Use the following initial code to generate the matrix `res`.



```
set.seed(100)
N=10
CX101 <- rnorm(N,45,8)
CX102 <- rnorm(N,65,8)
CX103 <- rnorm(N,85,25)
CX104 <- rnorm(N,60,15)
CX105 <- rnorm(N,55,15)
```



```
res
#>            CX101 CX102  CX103 CX104 CX105
#> Student-1  40.98 65.72  74.05 58.63 53.48
#> Student-2  46.05 65.77 104.10 86.36 76.05
#> Student-3  44.37 63.39  91.55 57.93 28.35
#> Student-4  52.09 70.92 104.34 58.33 64.34
#> Student-5  45.94 65.99  64.64 49.65 47.17
#> Student-6  47.55 64.77  74.04 56.67 74.83
#> Student-7  40.35 61.89  66.99 62.74 49.55
#> Student-8  50.72 69.09  90.77 66.26 74.79
#> Student-9  38.40 57.69  56.06 75.98 55.66
#> Student-10 42.12 83.48  91.18 74.55 26.82
```

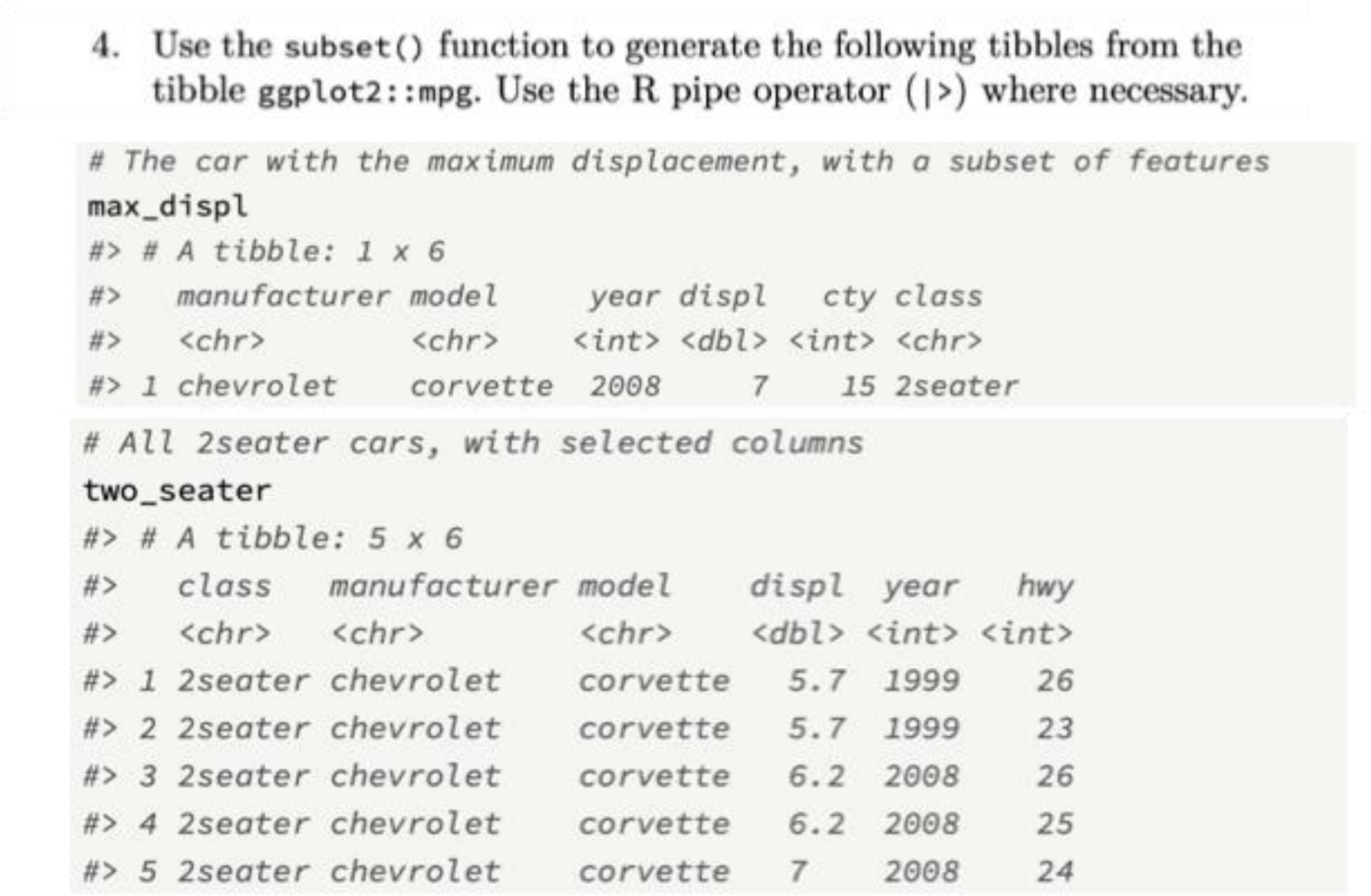


2. The matrix `res` (from the previous question) has values that are out of the valid range for grades (i.e., greater than 100). To address this, all out-of-range values should be replaced by `NA`. Use `apply()` to generate the following modified matrix.

```
res_clean
#>             CX101 CX102 CX103 CX104 CX105
#> Student-1   40.98 65.72 74.05 58.63 53.48
#> Student-2   46.05 65.77    NA 86.36 76.05
#> Student-3   44.37 63.39 91.55 57.93 28.35
#> Student-4   52.09 70.92    NA 58.33 64.34
#> Student-5   45.94 65.99 64.64 49.65 47.17
#> Student-6   47.55 64.77 74.04 56.67 74.83
#> Student-7   40.35 61.89 66.99 62.74 49.55
#> Student-8   50.72 69.09 90.77 66.26 74.79
#> Student-9   38.40 57.69 56.06 75.98 55.66
#> Student-10 42.12 83.48 91.18 74.55 26.82
```

3. The matrix `res_clean` (from the previous question) has NA values, and as a work around, it has been decided to replace these values with the average subject mark. Write the code (using `apply()`) to generate the matrix `res_update`.

```
res_update
#>             CX101 CX102 CX103 CX104 CX105
#> Student-1   40.98 65.72 74.05 58.63 53.48
#> Student-2   46.05 65.77 76.16 86.36 76.05
#> Student-3   44.37 63.39 91.55 57.93 28.35
#> Student-4   52.09 70.92 76.16 58.33 64.34
#> Student-5   45.94 65.99 64.64 49.65 47.17
#> Student-6   47.55 64.77 74.04 56.67 74.83
#> Student-7   40.35 61.89 66.99 62.74 49.55
#> Student-8   50.72 69.09 90.77 66.26 74.79
#> Student-9   38.40 57.69 56.06 75.98 55.66
#> Student-10 42.12 83.48 91.18 74.55 26.82
```



4. Use the `subset()` function to generate the following tibbles from the tibble `ggplot2::mpg`. Use the R pipe operator (`|>`) where necessary.

```
# The car with the maximum displacement, with a subset of features
max_displ
#> # A tibble: 1 x 6
#>   manufacturer model     year displ   cty class
#>   <chr>        <chr>    <int> <dbl> <int> <chr>
#> 1 chevrolet    corvette  2008     7    15 2seater
```

```
# All 2seater cars, with selected columns
two_seater
#> # A tibble: 5 x 6
#>   class   manufacturer model    displ  year   hwy
#>   <chr>   <chr>        <chr>    <dbl> <int> <int>
#> 1 2seater chevrolet    corvette   5.7  1999    26
#> 2 2seater chevrolet    corvette   5.7  1999    23
#> 3 2seater chevrolet    corvette   6.2  2008    26
#> 4 2seater chevrolet    corvette   6.2  2008    25
#> 5 2seater chevrolet    corvette   7    2008    24
```