Lab 6: Processing data II: tibble and tidyverse

Introduction and Objective:

In Lab 5, we learned how to import, select, and output the data. We also learned how to process data using conventional data.frame(). Today, we are going to continue our journey of processing data. We will install a package, which is called tidyverse, and use many functions provided by this package to select and process data. We will also learn how to convert a conventional data frame to a tibble.

All the R command lines are in **bold**; all the notes are following a #; all the R results directly follow the R codes/command lines and are not in bold or following a #.

The first thing is setting your working directory.

I. # Last week, we used rank() to rank our data in MidtermGradeSheet.csv. Today, I am going to show you the second way to rank the data. And this time we don't use rank(), but use order() instead.

a <- read.csv("MidtermGradeSheet.csv")

head(a)

```
  Student.ID Midterm
1         AE    81.0
2         AN    78.5
3         AS    82.0
4         CA    94.0
5         CF    77.0
6         DN    87.0
```

The following command gives you the positions of the scores in ascending order:

order(a$Midterm)

If you rearrange the rows of object "a" according to the positions of the scores, you can type:

a[order(a$Midterm),]

By using head(), you can take a glance at the results:

head(a[order(a$Midterm),])

```
   Student.ID Midterm
34         XD    61.0
14         ET    65.5
33         VJ    68.0
25         OO    70.0
10         EE    70.5
36         YC    72.0
```

If you wish to rearrange the scores by their positions in descending order, then you can type a “minus” before the data applied by order():

```
b<-a[order(-a$Midterm),]
```

Check the first 6 rows of the results:

```
head(b<-a[order(-a$Midterm),])
```

```
  Student.ID Midterm
7         DR    99.5
12        EL    96.5
4         CA    94.0
27        RA    93.5
15        EU    93.0
29        SA    92.5
```

Next, we can generate a column with the placements:

```
new_rank <- c(1:length(b[,1]))
```

Then we combine the rearranged data with the new column by using data.frame():

```
c <- data.frame(b,new_rank)
```

At last, if we want, we can save this processed data in a csv file:

```
write.csv(c,"ranked_midterm_scores.csv",row.names=FALSE)
```

You can check to make sure the file is what you want:

```
read.csv("ranked_midterm_scores.csv")
```

```
  Student.ID Midterm new_rank
1         DR    99.5        1
2         EL    96.5        2
3         CA    94.0        3
4         RA    93.5        4
```

```
5        EU    93.0       5
6        SA    92.5       6
```

If you want to keep the original positions, then put row.names=TRUE or use the default setting when you save the processed data, like

write.csv(c,"ranked_midterm_scores.csv")

(a1 <- read.csv("ranked_midterm_scores.csv"))

```
   X Student.ID Midterm new_rank
1  7        DR    99.5       1
2 12        EL    96.5       2
3  4        CA    94.0       3
4 27        RA    93.5       4
5 15        EU    93.0       5
6 29        SA    92.5       6
```

As you can see from the above if I used parentheses surround my code "a1 <- read.csv("ranked_midterm_scores.csv")", I did not have to type "a1" to see the results (we will discuss this in detail a few minutes later).

We can change the column name of a particular column. For instance:

colnames(a1)[colnames(a1)=="X"]<-"original_position"

a1

```
  original_position Student.ID Midterm new_rank
1                 7        DR    99.5       1
2                12        EL    96.5       2
3                 4        CA    94.0       3
4                27        RA    93.5       4
5                15        EU    93.0       5
6                29        SA    92.5       6
```

The "==" in the above line of code is for a logical argument.

Once you have the original positions included in the data, you can arrange the data in the order as it appeared to you in the first place:

(a1[order(a1$original_position),])

```
  original_position Student.ID Midterm new_rank
22                 1        AE    81.0      22
```

```
25          2       AN   78.5      25
19          3       AS   82.0      19
3           4       CA   94.0       3
26          5       CF   77.0      26
14          6       DN   87.0      14
```

II. Use package "tidyverse"

We are going to install the package "tidyverse".

install.packages("tidyverse")

And then we activate it

library(tidyverse)

And then you can see, in fact, tidyverse contains eight different packages. These packages are for processing and visualizing data.

It also shows something like

```
── Conflicts ──────────────────────────────────────────────────────────────────────────────── tidyverse_conflicts() ──
✖ dplyr::filter() masks stats::filter()
✖ dplyr::lag()    masks stats::lag()
Warning messages:
…
```

This shows what functions in the packages conflict with other R functions from other packages.

To understand the information, you need to know the double colon :: shows the origin of a function, for instance, dplyr::filter() means function filter() from package dplyr.

1. # Now let's start with some simple data. When we do this, we are going to learn some new functions besides practicing the ones we have already learned.

(a <- seq(1,10,length.out=3))

[1] 1.0 5.5 10.0

Previously, after we defined an object, for instance, a <- seq(1,10,length.out=3), to know what is defined in "a", we need to type "a" in a second command line, like

a <- seq(1,10,length.out=3)

a

However, if you use parenthesis to surround the command line you wrote, R will show the result of the code without asking you to type a second command line.

The seq() is a function, it generates a sequence of numbers. In the parenthesis, the first number shows the minimum of the sequence of the numbers, the second number tells R the maximum of the sequence of the numbers. The argument "length.out=" tells R, how many numbers in the sequence, then by default, R will evenly distribute the numbers.

(b <- seq(2,12,length.out=3))

[1] 2 7 12

Now let's make a small data frame:

(c<-data.frame(a,b))

```
    a  b
1  1.0  2
2  5.5  7
3 10.0 12
```

(d <- rep(NA,3))

[1] NA NA NA

The function rep() generates a string of 3 repeated "NA".

(c <- data.frame(a,b,d))

```
    a  b  d
1  1.0  2 NA
2  5.5  7 NA
3 10.0 12 NA
```

The function is.na() allows you to check if the values are missing data (NA):

is.na(a)

[1] FALSE FALSE FALSE

is.na(d)

[1] TRUE TRUE TRUE

is.na(c)

```
      a     b     d
[1,] FALSE FALSE TRUE
[2,] FALSE FALSE TRUE
[3,] FALSE FALSE TRUE
```

As you can see, is.na() gives logical results, although it is a function.

As we mentioned earlier, the "==" is for logical "be equal to". And "!" is for logical "does not equal". See examples below:

```
a==b
```

[1] FALSE FALSE FALSE

2. # The function filter() is used to select rows.

(filter(c,b %in% c(1)))

[1] a b d

<0 rows> (or 0-length row.names)

See another example:

(filter(c,b %in% c(2,12)))

```
   a  b  d
1  1  2 NA
2 10 12 NA
```

The "b %in% c(2,12)" means in variable b when it is 2 or 12.

Other examples of using filter and logical argument:

(filter(c,b==12))

```
   a  b  d
1 10 12 NA
```

(filter(c,b!=12))

```
   a  b   d
1 1.0 2 NA
2 5.5 7 NA
```

The following example shows how to use between() to select rows.

(filter(c,between(b,1,8)))

```
   a  b   d
1 1.0 2 NA
2 5.5 7 NA
```

You can see it chose the first 2 rows because the third one is false:

(between(b,1,8))

```
[1]  TRUE  TRUE FALSE
```

3. # The function "filter()" filtrates the data and selects the rows, while the function "arrange()" change the order of the rows (does this remind you the function "rank()" and "order()" in base R?).

a <- c(1, 3, 9)

b <- c(2, 12, 5)

d<-rep(NA,3)

(c<-data.frame(a,b,d))

```
  a  b   d
1 1  2  NA
2 3 12 NA
```

3 9 5 NA

(arrange(c,b))

```
  a  b   d
1 1  2  NA
2 9  5  NA
3 3 12 NA
```

As you can see, without specifying how to arrange the rows, by default, the function "arrange()" changes the order of the rows by arranging the values ascendingly in the variable (column) that has been chosen. If you wish to have the rows ordered by the values in descending order, you can use either "desc" or adding a "-" in front of the variable you want to arrange by.

(arrange(c,desc(b)))

or

(arrange(c,-b))

```
  a  b  d
1 3 12 NA
2 9  5 NA
3 1  2 NA
```

If there are ties in the values of the variable that is used to order by, no matter the rows are ordered by ascending or descending order of the values in the designated variable, the rows with tie values come in the order as they present in the original data.

The following is an example:

a<-c(2,3,1,0)

b<-c(3,3,8,7)

c<-c(4,10,11,8)

(d<-data.frame(a,c,b))

```
  a  c  b
1 2  4  3
2 3 10 3
3 1 11 8
4 0  8  7
```

You have two 3s in variable b. When you arrange the data according to b, you get:

(arrange(d,b))

```
  a  c  b
1 2  4  3
2 3 10 3
3 0  8  7
4 1 11 8
```

We can use arrange() to move the missing values (NA) to the top:

a<-c(1,0,NA,3)

b<-c(8,NA,3,3)

c<-c(11,8,4,NA)

(d <- data.frame(a,b,c))

```
   a    b   c
1  1    8  11
2  0   NA  8
3 NA  3    4
4  3    3  NA
```

(arrange(d,desc(is.na(a)),desc(is.na(b)),desc(is.na(c))))

```
    a   b   c
1 NA  3   4
2  0   NA  8
3  3   3   NA
4  1   8   11
```

4. # Use as.tibble(), we can convert a conventional data frame into a tibble:

as.tibble(d)

A tibble: 4 x 3

```
      a      b      c
  <dbl> <dbl> <dbl>
1     1      8     11
2     0     NA     8
3    NA     3      4
4     3      3     NA
```

As you can see, a tibble adopted the idea of the conventional data frame and use the variables as the columns. In addition, it shows the size of the tibble (number of rows x number of columns), and below the names of the variables, it shows the types of the variables. Besides, different from a conventional data frame, which will show all the rows of the data (of course, we can use the head() to view the first 6 rows only), in a large dataset has more than ten rows, a tibble will only show the first 10 rows of the data.

5. # Use select() to select columns

select(d,c("a","b"))

```
   a    b
1  1    8
2  0   NA
3 NA  3
4  3    3
```

Add "-" in front of the name of the variables to exclude the corresponding columns.

select(d,-c("a","b"))

```
  c
1 11
2  8
3  4
4 NA
```

Examples of using starts_with(), ends_with(), contains(), matches().

(select(d,starts_with("a")))

```
   a
1  1
2  0
3 NA
4  3
```

(select(d,ends_with("a")))

```
   a
1  1
2  0
3 NA
4  3
```

(select(d,contains("a")))

```
   a
1  1
2  0
3 NA
4  3
```

select(d,matches("c"))

```
   c
1 11
2  8
3  4
4 NA
```

6. # Tricks for column and row names

Use sprintf() to create row and column names using a constant string with variable numbers. For instance, create a table with the column names as a1, a2, a3, a4, a5, a6, a7, a8, a9, and a10; and the row names as b1, b2, b3, b4, b5, b6, b7, b8, b9, and b10.

a <- matrix(1:100,ncol=10,byrow=TRUE)

colnames(a)=c(sprintf("a%s",1:10))

rownames(a)=c(sprintf("b%s",1:10))

a

```
    a1 a2 a3 a4 a5 a6 a7 a8 a9 a10
b1   1  2  3   4  5  6  7  8  9  10
b2  11 12 13 14 15 16 17 18 19  20
b3  21 22 23 24 25 26 27 28 29  30
b4  31 32 33 34 35 36 37 38 39  40
b5  41 42 43 44 45 46 47 48 49  50
b6  51 52 53 54 55 56 57 58 59  60
b7  61 62 63 64 65 66 67 68 69  70
b8  71 72 73 74 75 76 77 78 79  80
b9  81 82 83 84 85 86 87 88 89  90
b10 91 92 93 94 95 96 97 98 99 100
```

When we create the table, if we tell R byrow=FALSE, or play with it and write byrow=(“a”==”A”), as in the code below:

(b<-matrix(1:100,ncol=10,byrow=("a"=="A")))

then, the values in the table being generated are arranged by columns.

```
      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]
 [1,]    1   11   21   31   41   51   61   71   81    91
 [2,]    2   12   22   32   42   52   62   72   82    92
 [3,]    3   13   23   33   43   53   63   73   83    93
 [4,]    4   14   24   34   44   54   64   74   84    94
 [5,]    5   15   25   35   45   55   65   75   85    95
 [6,]    6   16   26   36   46   56   66   76   86    96
 [7,]    7   17   27   37   47   57   67   77   87    97
 [8,]    8   18   28   38   48   58   68   78   88    98
 [9,]    9   19   29   39   49   59   69   79   89    99
[10,]   10   20   30   40   50   60   70   80   90   100
```

Besides what we have already talked about, there are some other tricks to change the column and row names.

Here, we show how we can achieve this by using an example to replace the column name "a6" with "A6":

(colnames(a)=c(sprintf("a%s",1:5),"A6",sprintf("a%s",7:10)))

```
 [1] "a1"  "a2"  "a3"  "a4"  "a5"  "A6"  "a7"  "a8"  "a9"  "a10"
```

a

```
    a1 a2 a3 a4 a5 A6 a7 a8 a9 a10
b1   1  2  3  4  5  6  7  8  9  10
b2  11 12 13 14 15 16 17 18 19  20
b3  21 22 23 24 25 26 27 28 29  30
b4  31 32 33 34 35 36 37 38 39  40
b5  41 42 43 44 45 46 47 48 49  50
b6  51 52 53 54 55 56 57 58 59  60
b7  61 62 63 64 65 66 67 68 69  70
b8  71 72 73 74 75 76 77 78 79  80
b9  81 82 83 84 85 86 87 88 89  90
b10 91 92 93 94 95 96 97 98 99 100
```

We can do similar things using rownames() to change the row names.

Make it as a data frame before we can select the columns.

A <- data.frame(a)

As you can see below, when we are using select(), R does not differentiate the case of the column names.

(select(A,starts_with("a")))

```
    a1 a2 a3 a4 a5 A6 a7 a8 a9 a10
b1   1  2  3  4  5  6  7  8  9  10
b2  11 12 13 14 15 16 17 18 19  20
b3  21 22 23 24 25 26 27 28 29  30
b4  31 32 33 34 35 36 37 38 39  40
b5  41 42 43 44 45 46 47 48 49  50
b6  51 52 53 54 55 56 57 58 59  60
b7  61 62 63 64 65 66 67 68 69  70
b8  71 72 73 74 75 76 77 78 79  80
b9  81 82 83 84 85 86 87 88 89  90
b10 91 92 93 94 95 96 97 98 99 100
```

We learned how to change all the column names or row names. But what if we only want to change one or some of the column/row names? How can we not write all the column/row names we don't want to change, but only type the ones we'd like to (You may remember we mentioned this earlier today)?

Here is an example of changing a specific column's name:

colnames(A)[colnames(A)=="A6"] <- "C2"

A

```
    a1 a2 a3 a4 a5 C2 a7 a8 a9 a10
b1   1  2  3  4  5  6  7  8  9  10
b2  11 12 13 14 15 16 17 18 19  20
b3  21 22 23 24 25 26 27 28 29  30
b4  31 32 33 34 35 36 37 38 39  40
b5  41 42 43 44 45 46 47 48 49  50
b6  51 52 53 54 55 56 57 58 59  60
b7  61 62 63 64 65 66 67 68 69  70
b8  71 72 73 74 75 76 77 78 79  80
b9  81 82 83 84 85 86 87 88 89  90
b10 91 92 93 94 95 96 97 98 99 100
```

Similarly, we can change the name of a specific row.

rownames(A)[rownames(A)=="b6"] <- "B6"

A

```
     a1 a2 a3 a4 a5 C2 a7 a8 a9 a10
b1    1  2  3  4  5  6  7  8  9  10
b2   11 12 13 14 15 16 17 18 19  20
b3   21 22 23 24 25 26 27 28 29  30
b4   31 32 33 34 35 36 37 38 39  40
b5   41 42 43 44 45 46 47 48 49  50
B6   51 52 53 54 55 56 57 58 59  60
b7   61 62 63 64 65 66 67 68 69  70
b8   71 72 73 74 75 76 77 78 79  80
b9   81 82 83 84 85 86 87 88 89  90
b10 91 92 93 94 95 96 97 98 99 100
```

Move some chosen columns/variables to the front by using everything()

select(A,"a10","a9","a7","a8",everything())

```
     a10 a9 a7 a8 a1 a2 a3 a4 a5 C2

b1   10  9  7  8  1  2  3  4  5  6
b2   20 19 17 18 11 12 13 14 15 16
b3   30 29 27 28 21 22 23 24 25 26
b4   40 39 37 38 31 32 33 34 35 36
b5   50 49 47 48 41 42 43 44 45 46
B6   60 59 57 58 51 52 53 54 55 56
b7   70 69 67 68 61 62 63 64 65 66
b8   80 79 77 78 71 72 73 74 75 76
b9   90 89 87 88 81 82 83 84 85 86
b10 100 99 97 98 91 92 93 94 95 96
```

7. # Adding a variable/column using mutate():

(A1 <- mutate(A,a11=a2+a3))

```
   a1 a2 a3 a4 a5 C2 a7 a8 a9 a10 a11

1   1  2  3  4  5  6  7  8  9  10   5
2  11 12 13 14 15 16 17 18 19  20  25
3  21 22 23 24 25 26 27 28 29  30  45
4  31 32 33 34 35 36 37 38 39  40  65
5  41 42 43 44 45 46 47 48 49  50  85
6  51 52 53 54 55 56 57 58 59  60 105
7  61 62 63 64 65 66 67 68 69  70 125
```

```
8  71 72 73 74 75 76 77 78 79  80 145
9  81 82 83 84 85 86 87 88 89  90 165
10 91 92 93 94 95 96 97 98 99 100 185
```

Use transmute() to show the new variable/column only:

(transmute(A,a11=a2+a3))

```
   a11
1    5
2   25
3   45
4   65
5   85
6  105
7  125
8  145
9  165
10 185
```

Assignment 4 (groupwork):

Do **NOT** use the function "rank()" or "order()" from base R,

1) rank the scores in "**MidtermGradeSheet.csv**" by using a function from package "tidyverse" and make sure who got the highest score would receive rank "1" (5pts);
2) and then add a new column to the original table with the name "new_rank" showing the ranks (3pts).
3) Save it as a csv file with the name as "ranked_midterm_scores_again" (1.5pt). Tell me where it was saved to (in which folder I can find it). (0.5pt)

Type down all the R codes and results.