R for toxicology

3. Data preprocessing 2

Objective

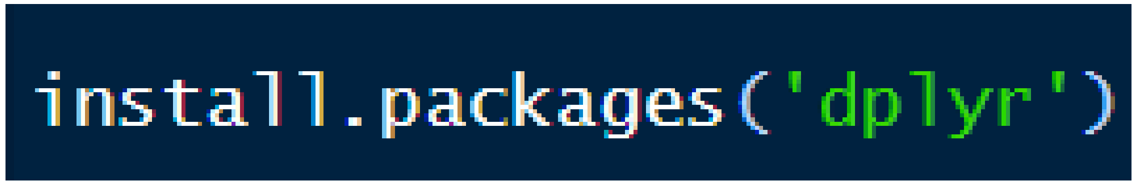
- Using dplyr package to process data

Why we use dplyr?

- Fast!
- Intuitive programming!
- Easier than basic R grammar

How to install dplyr package?

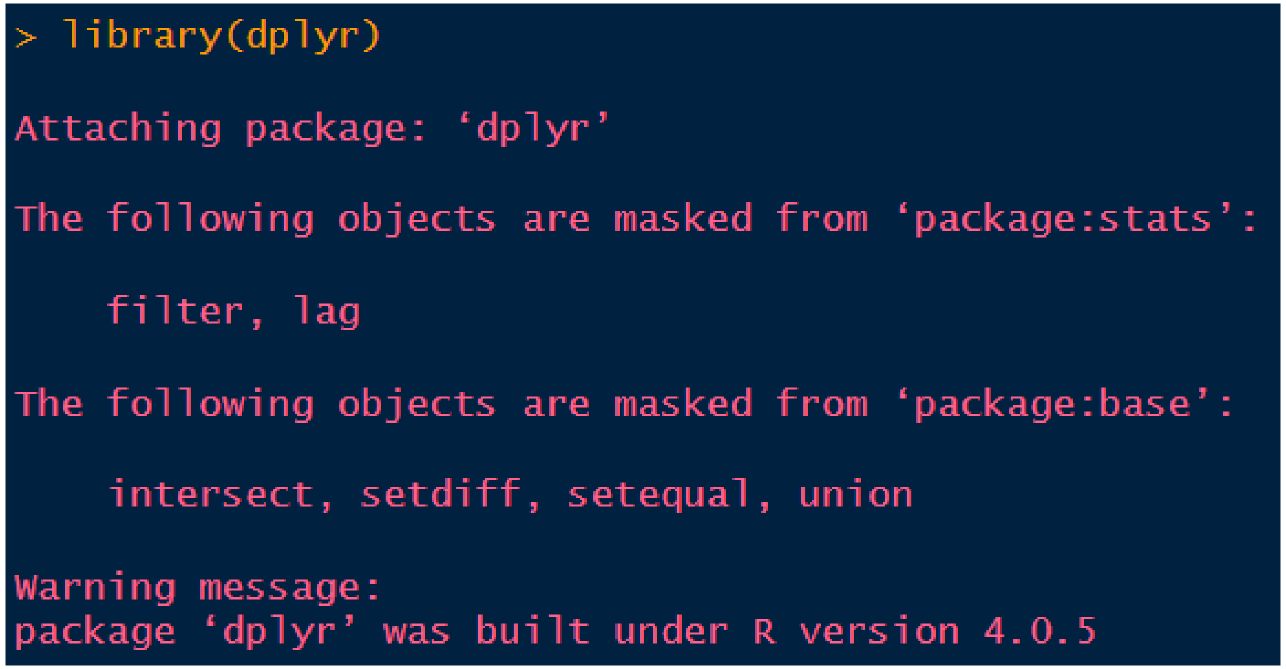
- Use install.packages(“~~”)



```
install.packages('dplyr')
```

How to install dplyr package?

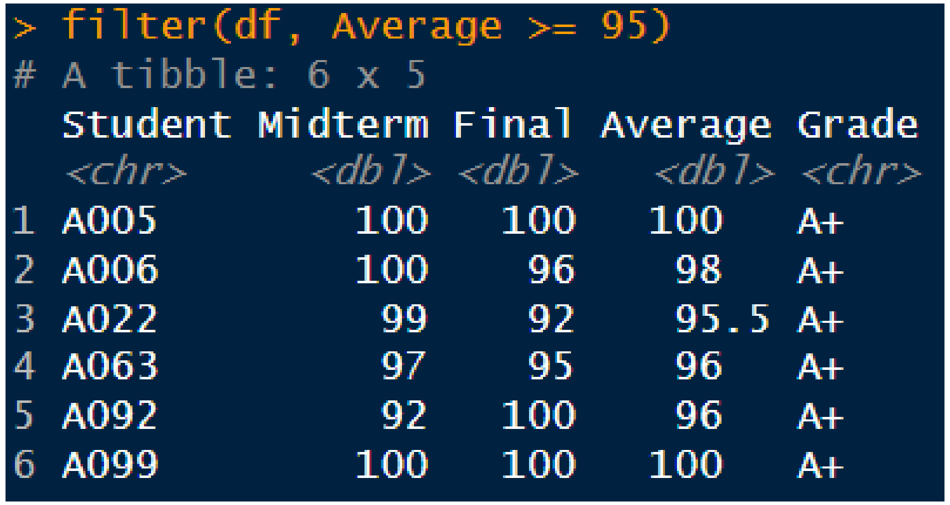
Load dplyr package using library(~~)



```
> library(dplyr)

Attaching package: ‘dplyr’

The following objects are masked from ‘package:stats’:

    filter, lag

The following objects are masked from ‘package:base’:

    intersect, setdiff, setequal, union

Warning message:
package ‘dplyr’ was built under R version 4.0.5
```

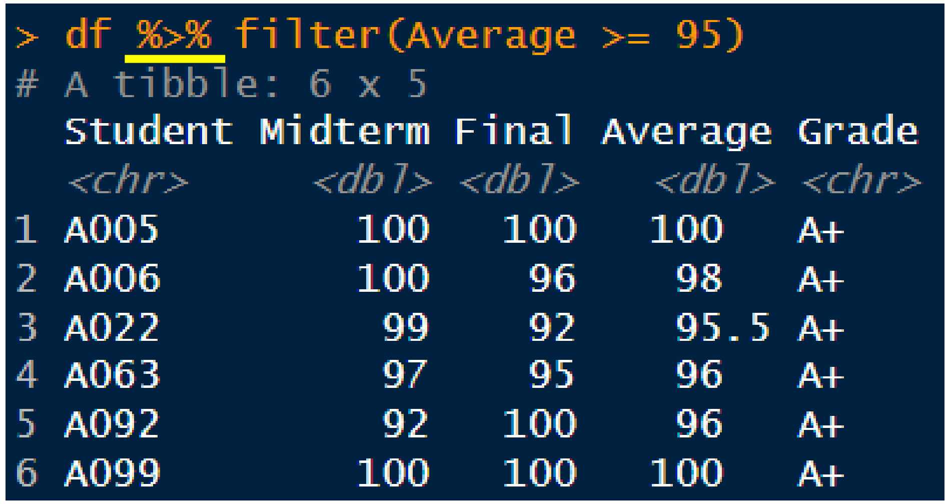
Two types of marking in dplyr

General method



```
> filter(df, Average >= 95)
# A tibble: 6 x 5
  Student Midterm Final Average Grade
  <chr>     <dbl> <dbl>   <dbl> <chr>
1 A005        100   100   100   A+
2 A006        100    96    98   A+
3 A022         99    92    95.5 A+
4 A063         97    95    96   A+
5 A092         92   100    96   A+
6 A099        100   100   100   A+
```

Chaining method



```
> df %>% filter(Average >= 95)
# A tibble: 6 x 5
  Student Midterm Final Average Grade
  <chr>     <dbl> <dbl>   <dbl> <chr>
1 A005        100   100   100   A+
2 A006        100    96    98   A+
3 A022         99    92    95.5 A+
4 A063         97    95    96   A+
5 A092         92   100    96   A+
6 A099        100   100   100   A+
```

I recommend this method because it can simplify your code

The basic 6 functions in dplyr

- filter
- select
- arrange
- rename
- distinct
- mutate

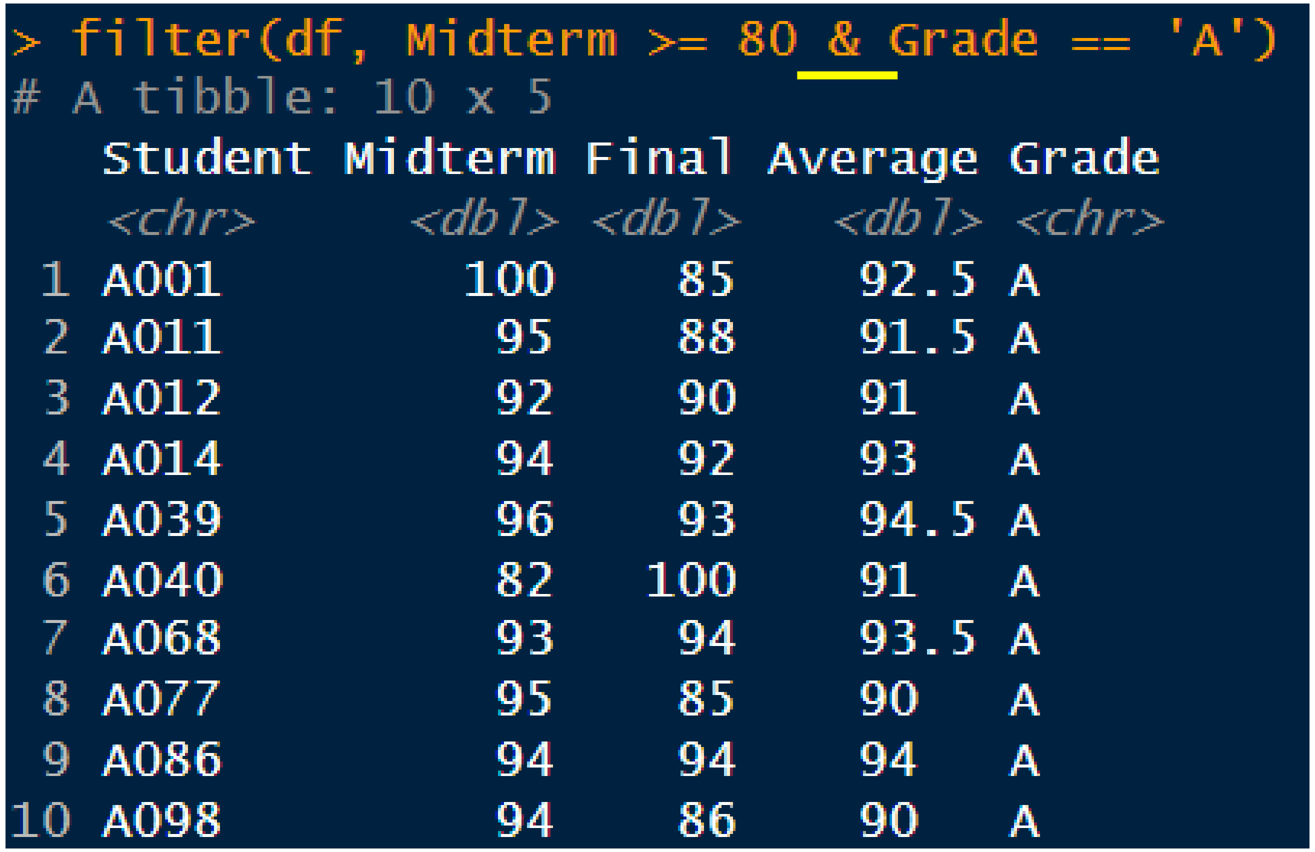
dplyr function 1: filter()

- Select specific rows that match with conditions

- Logical operators (>, <, ==, !=, >=, <=, &, |) are used to make conditions

dplyr function 1: filter()

Ex) Select data that has Midterm≥80 and Grade is A

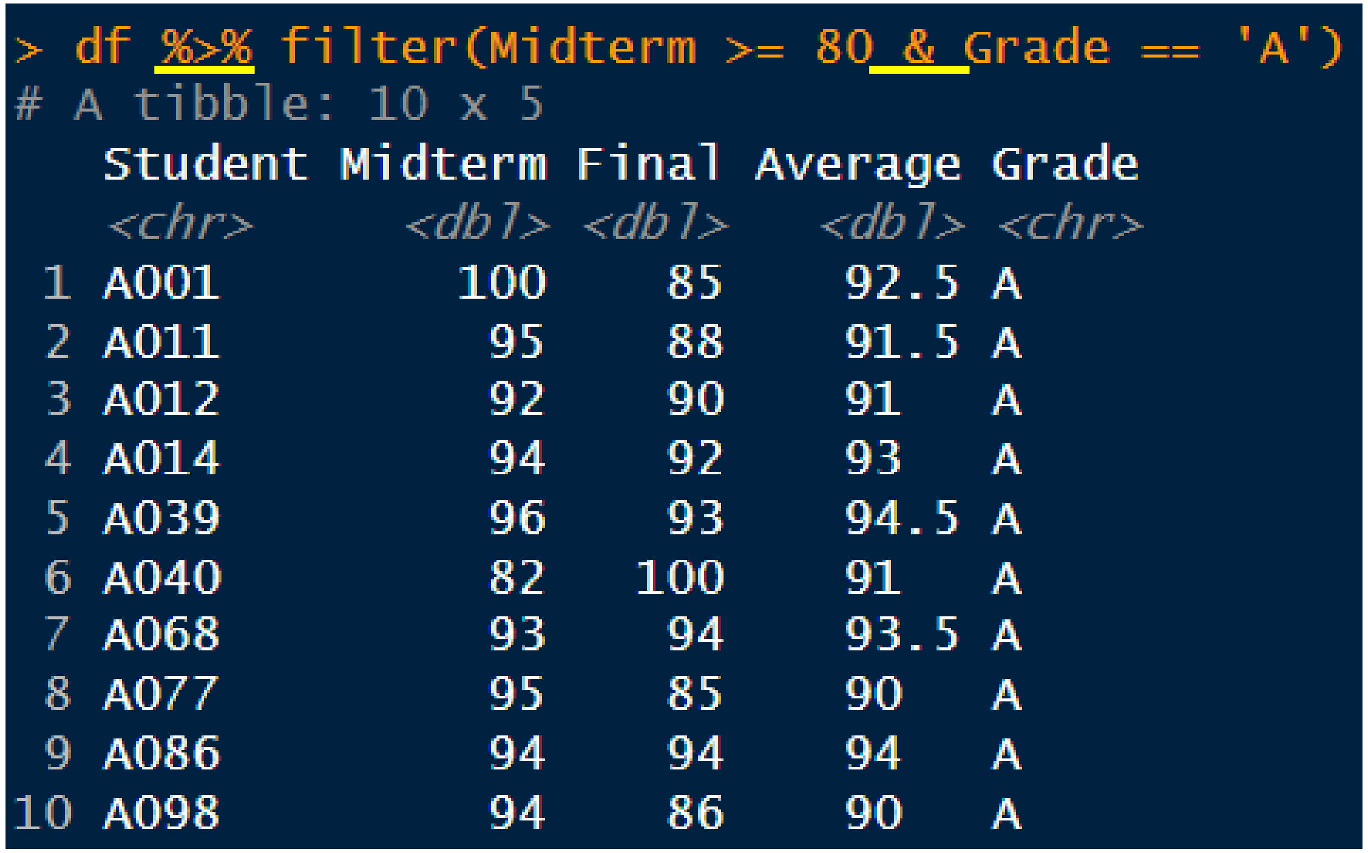


```
> filter(df, Midterm >= 80 & Grade == 'A')
# A tibble: 10 x 5
   Student Midterm Final Average Grade
   <chr>     <dbl> <dbl>   <dbl> <chr>
 1 A001        100    85    92.5 A
 2 A011         95    88    91.5 A
 3 A012         92    90    91   A
 4 A014         94    92    93   A
 5 A039         96    93    94.5 A
 6 A040         82   100    91   A
 7 A068         93    94    93.5 A
 8 A077         95    85    90   A
 9 A086         94    94    94   A
10 A098         94    86    90   A
```

dplyr function 1: filter()

Ex) Select data that has Midterm≥80 and Grade is A

Pipe operator (%>%) can also be applied



```
> df %>% filter(Midterm >= 80 & Grade == 'A')
# A tibble: 10 x 5
   Student Midterm Final Average Grade
   <chr>     <dbl> <dbl>   <dbl> <chr>
 1 A001        100    85    92.5 A
 2 A011         95    88    91.5 A
 3 A012         92    90    91   A
 4 A014         94    92    93   A
 5 A039         96    93    94.5 A
 6 A040         82   100    91   A
 7 A068         93    94    93.5 A
 8 A077         95    85    90   A
 9 A086         94    94    94   A
10 A098         94    86    90   A
```

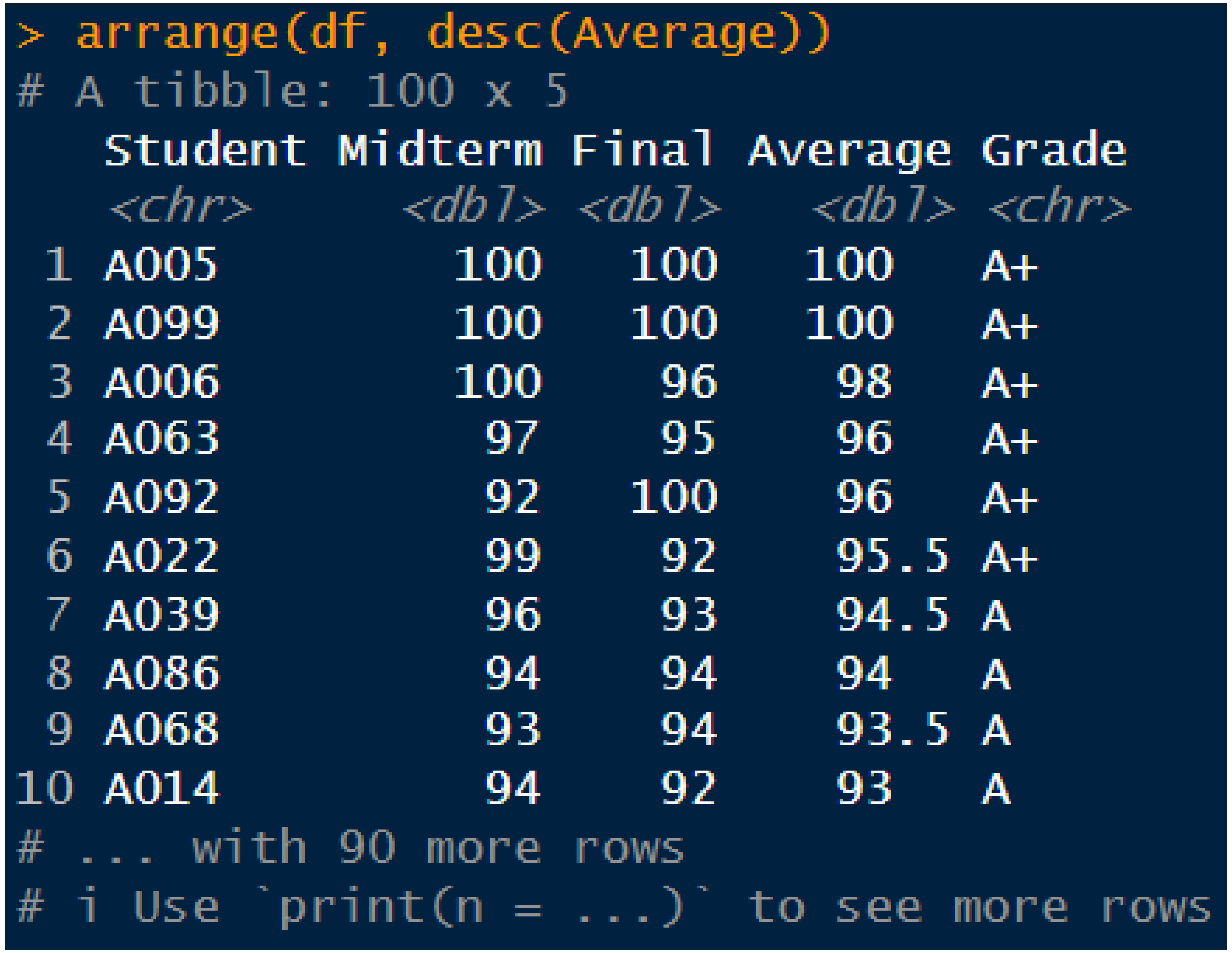
dplyr function 2: arrange()

- Arrange data by ascending or descending orders

- If you want to use descending order, use desc(～)

dplyr function 2: arrange()

- Ex) Descending order by Average

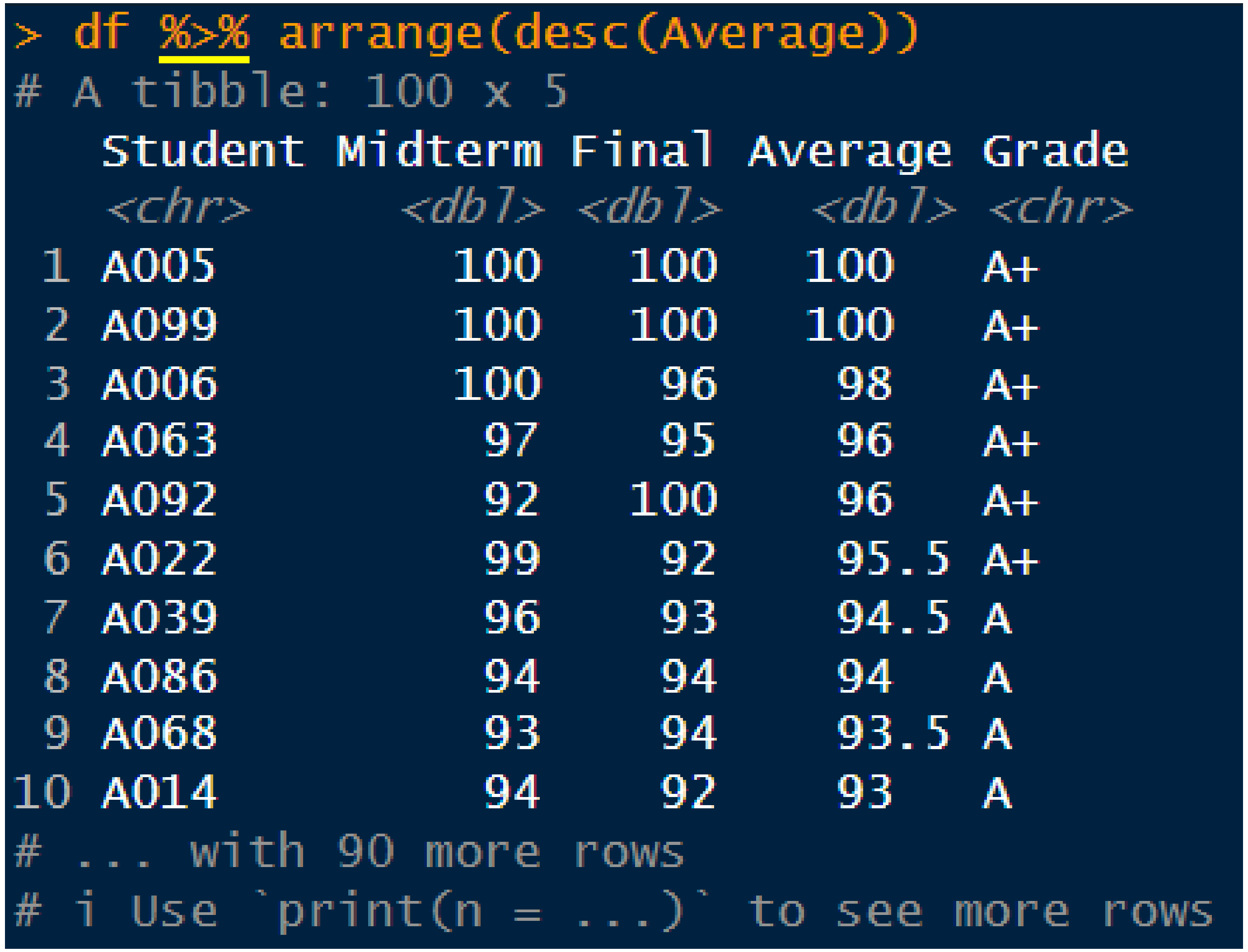


```
> arrange(df, desc(Average))
# A tibble: 100 x 5
   Student Midterm Final Average Grade
   <chr>     <dbl> <dbl>   <dbl> <chr>
 1 A005        100   100   100   A+
 2 A099        100   100   100   A+
 3 A006        100    96    98   A+
 4 A063         97    95    96   A+
 5 A092         92   100    96   A+
 6 A022         99    92    95.5 A+
 7 A039         96    93    94.5 A
 8 A086         94    94    94   A
 9 A068         93    94    93.5 A
10 A014         94    92    93   A
# ... with 90 more rows
# i Use `print(n = ...)` to see more rows
```

dplyr function 2: arrange()

- Ex) Descending order by Average

Pipe operator (%>%) can also be applied



```
> df %>% arrange(desc(Average))
# A tibble: 100 x 5
   Student Midterm Final Average Grade
   <chr>     <dbl> <dbl>   <dbl> <chr>
 1 A005        100   100   100   A+
 2 A099        100   100   100   A+
 3 A006        100    96    98   A+
 4 A063         97    95    96   A+
 5 A092         92   100    96   A+
 6 A022         99    92    95.5 A+
 7 A039         96    93    94.5 A
 8 A086         94    94    94   A
 9 A068         93    94    93.5 A
10 A014         94    92    93   A
# ... with 90 more rows
# i Use `print(n = ...)` to see more rows
```

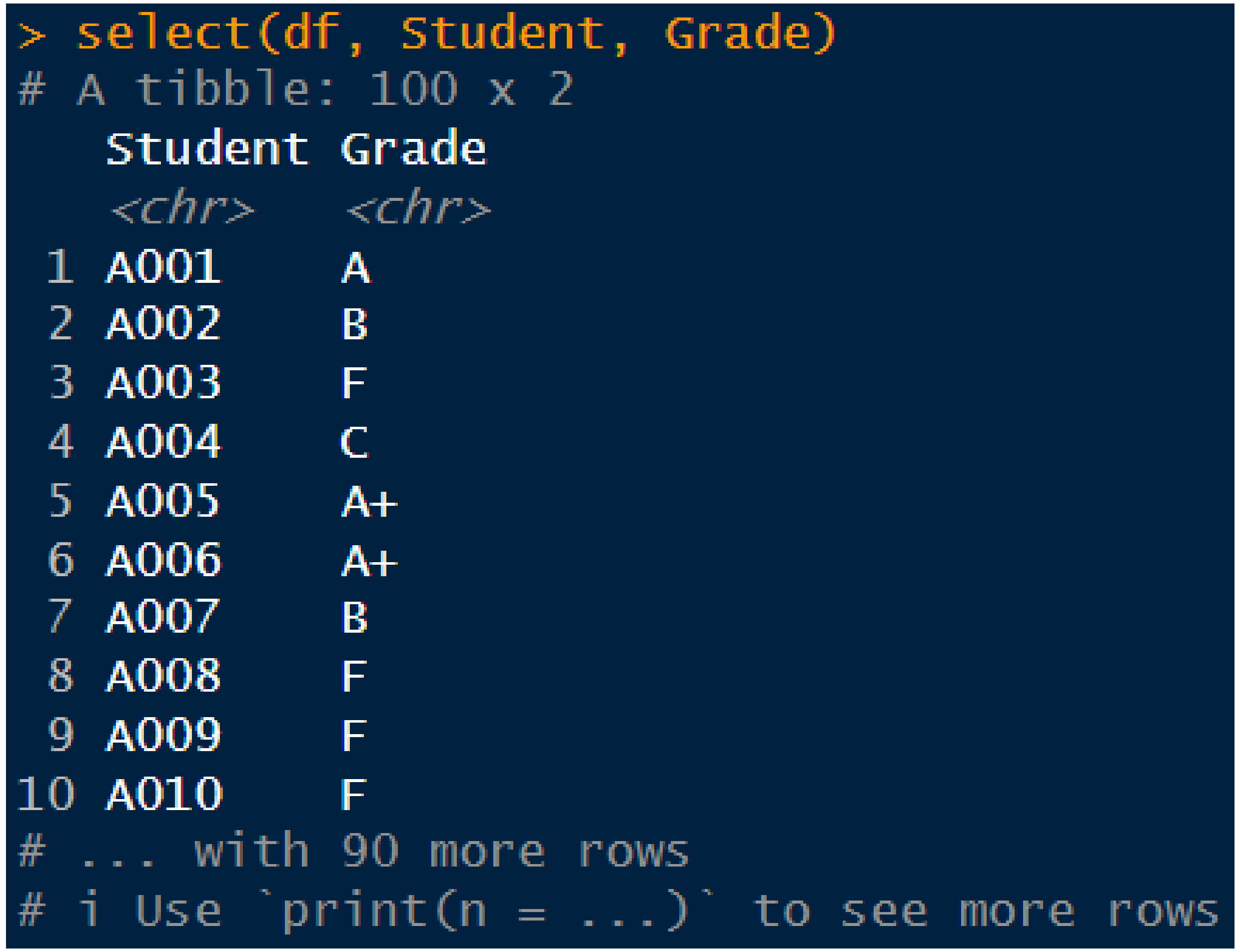
dplyr function 3: select()

- Select columns you want to take

- Logical operators can also be applied

dplyr function 3: select()

Ex) Select Student and Grade columns from df

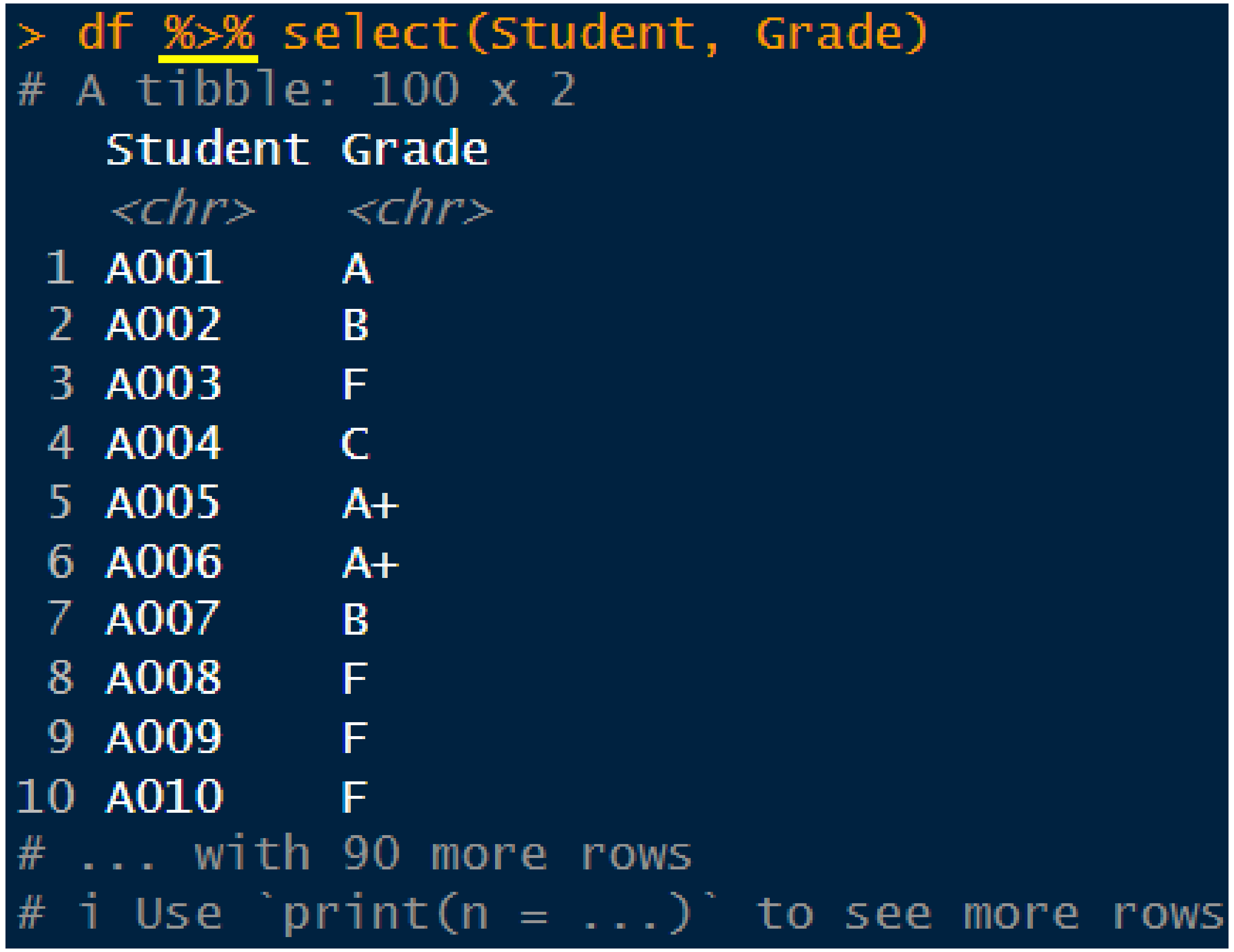


```
> select(df, Student, Grade)
# A tibble: 100 x 2
   Student Grade
   <chr>   <chr>
 1 A001    A
 2 A002    B
 3 A003    F
 4 A004    C
 5 A005    A+
 6 A006    A+
 7 A007    B
 8 A008    F
 9 A009    F
10 A010    F
# ... with 90 more rows
# i Use `print(n = ...)` to see more rows
```

dplyr function 3: select()

Ex) Select Student and Grade columns from df

Pipe operator (%>%) can also be applied



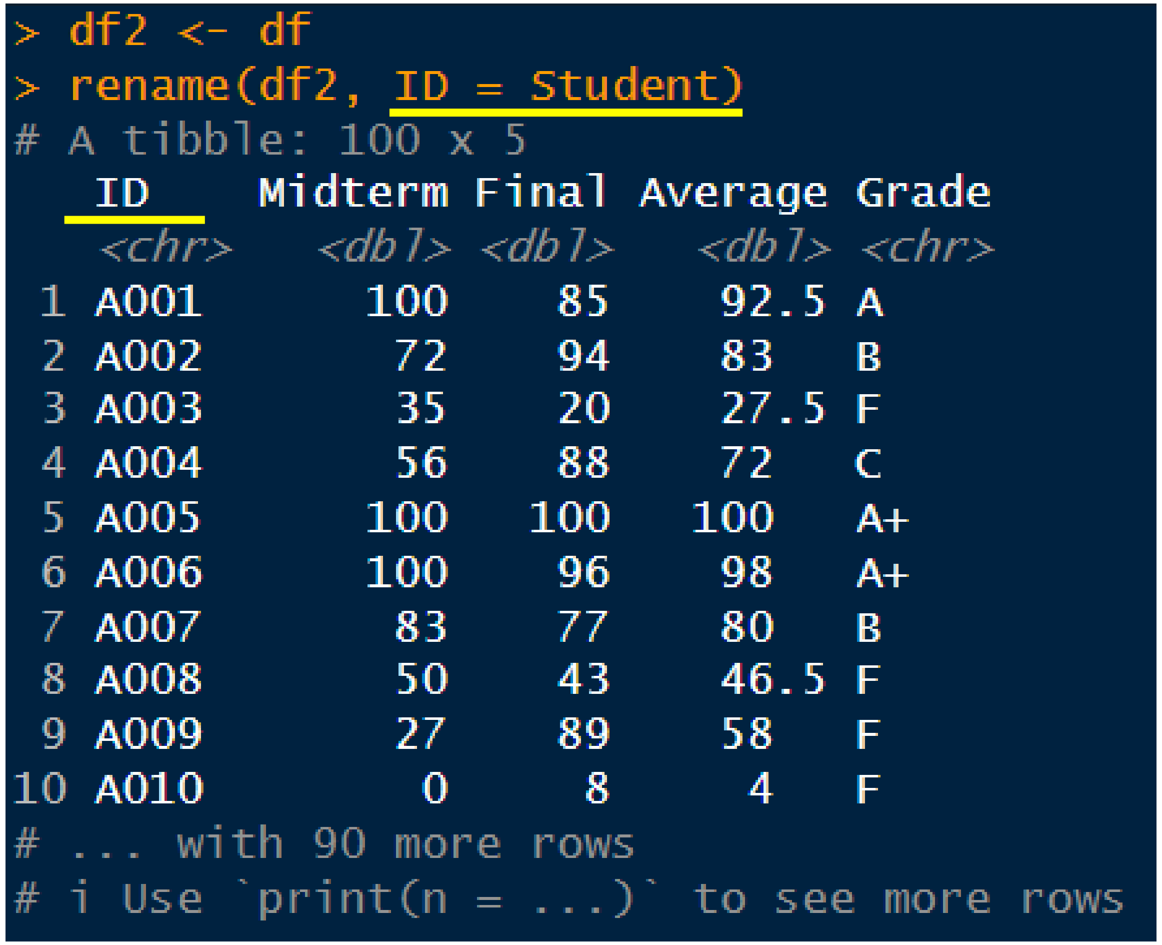
```
> df %>% select(Student, Grade)
# A tibble: 100 x 2
   Student Grade
   <chr>   <chr>
 1 A001    A
 2 A002    B
 3 A003    F
 4 A004    C
 5 A005    A+
 6 A006    A+
 7 A007    B
 8 A008    F
 9 A009    F
10 A010    F
# ... with 90 more rows
# i Use `print(n = ...)` to see more rows
```

dplyr function 4: rename()

- Change column names

dplyr function 4: rename()

Ex) Change ‘Student’ column to ‘ID’ column

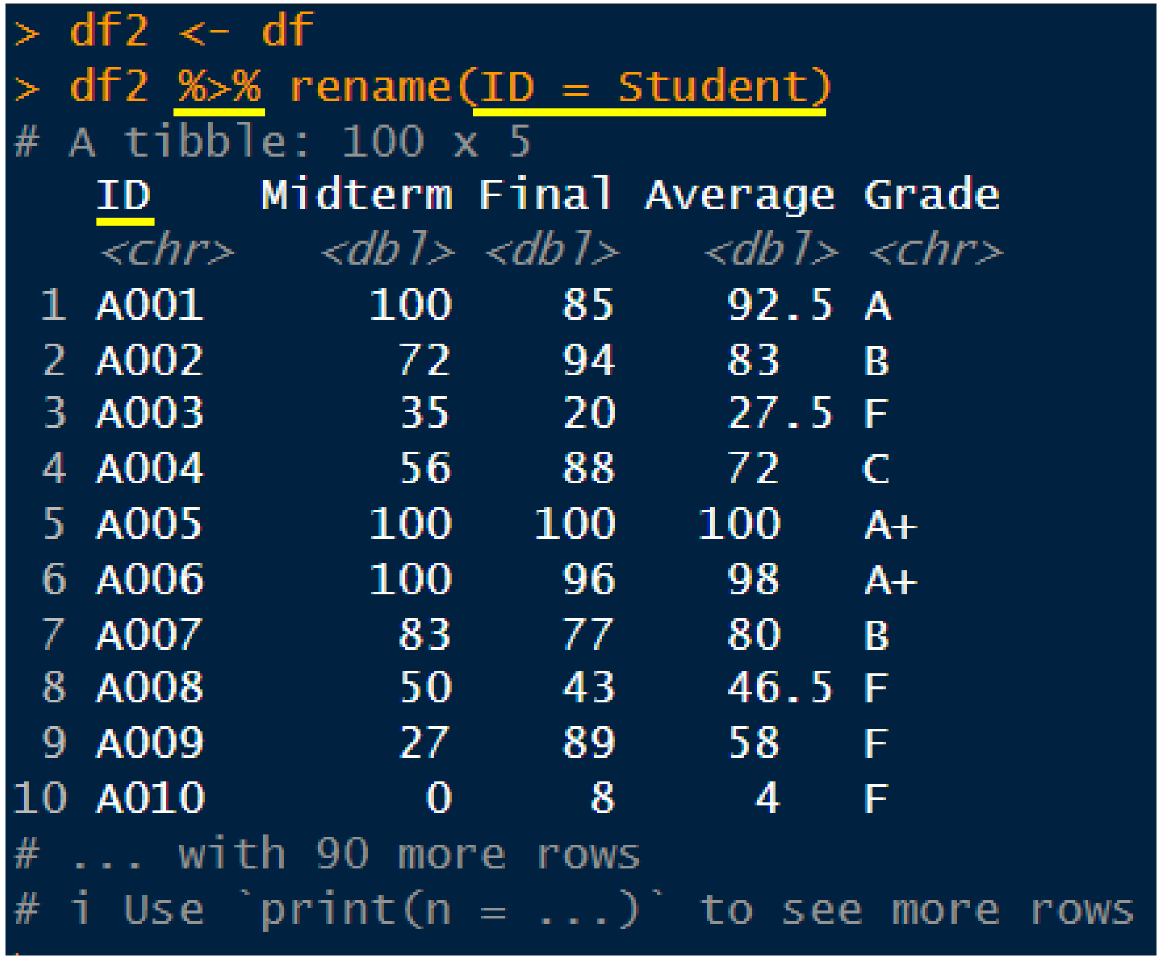


```
> df2 <- df
> rename(df2, ID = Student)
# A tibble: 100 x 5
   ID    Midterm Final Average Grade
   <chr>   <dbl> <dbl>   <dbl> <chr>
 1 A001      100    85    92.5 A
 2 A002       72    94    83   B
 3 A003       35    20    27.5 F
 4 A004       56    88    72   C
 5 A005      100   100   100   A+
 6 A006      100    96    98   A+
 7 A007       83    77    80   B
 8 A008       50    43    46.5 F
 9 A009       27    89    58   F
10 A010        0     8     4   F
# ... with 90 more rows
# i Use `print(n = ...)` to see more rows
```

dplyr function 4: rename()

Ex) Change 'Student' column to 'ID' column

Pipe operator (%>%)
can also be applied



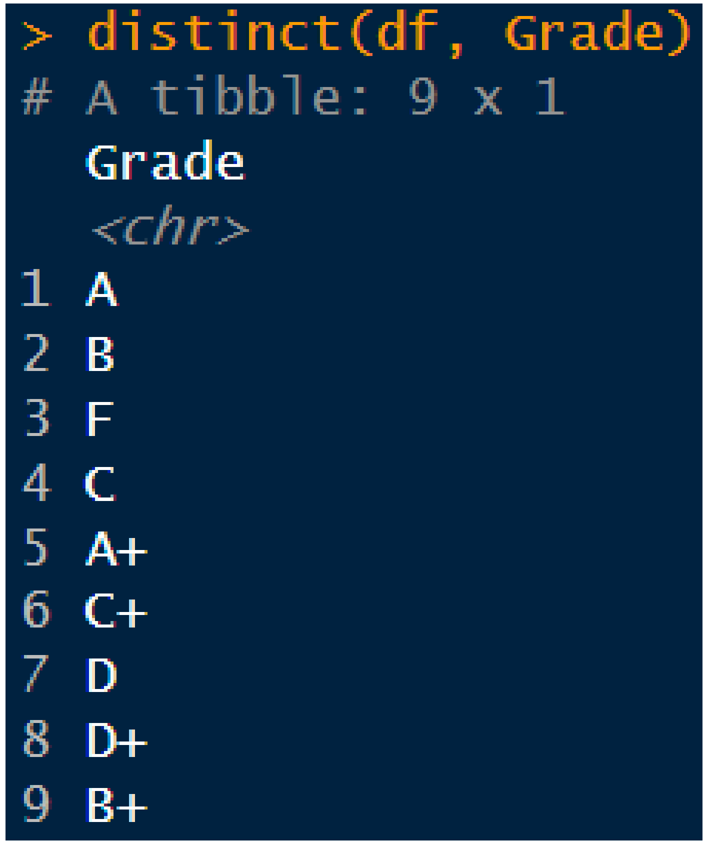
```
> df2 <- df
> df2 %>% rename(ID = Student)
# A tibble: 100 x 5
   ID    Midterm Final Average Grade
   <chr>   <dbl> <dbl>   <dbl> <chr>
 1 A001      100    85    92.5 A
 2 A002       72    94    83   B
 3 A003       35    20    27.5 F
 4 A004       56    88    72   C
 5 A005      100   100   100   A+
 6 A006      100    96    98   A+
 7 A007       83    77    80   B
 8 A008       50    43    46.5 F
 9 A009       27    89    58   F
10 A010        0     8     4   F
# ... with 90 more rows
# i Use `print(n = ...)` to see more rows
```

dplyr function 5: distinct()

- Show unique rows

dplyr function 5: distinct()

Ex) Show unique data in ‘Grade’ column

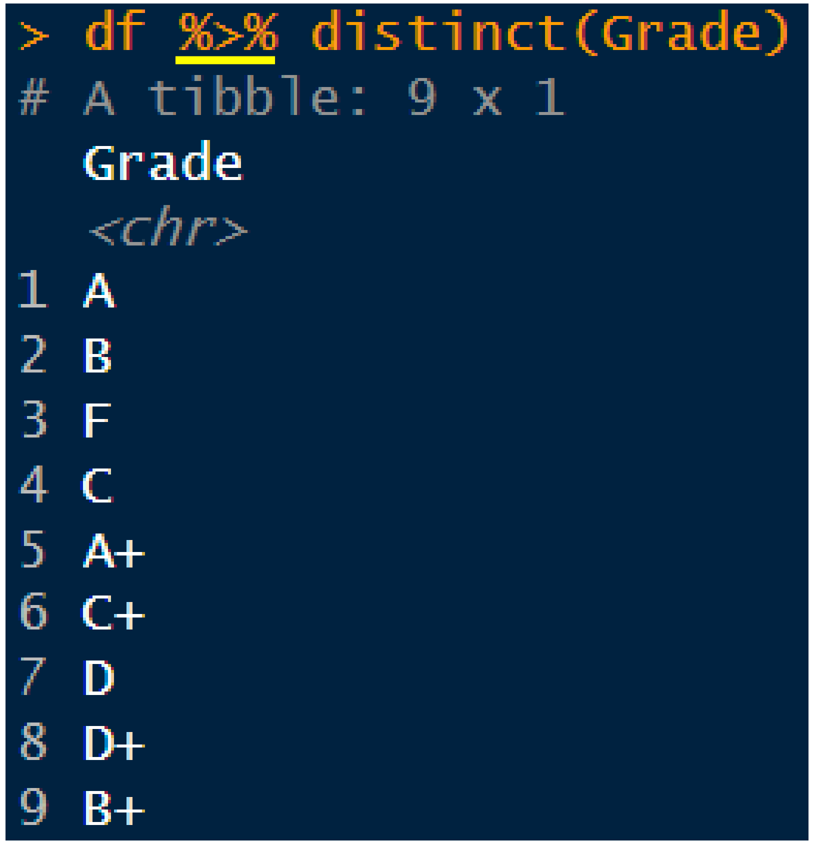


```
> distinct(df, Grade)
# A tibble: 9 x 1
  Grade
  <chr>
1 A
2 B
3 F
4 C
5 A+
6 C+
7 D
8 D+
9 B+
```

dplyr function 5: distinct()

Ex) Show unique data in 'Grade' column

Pipe operator (%>%)
can also be applied



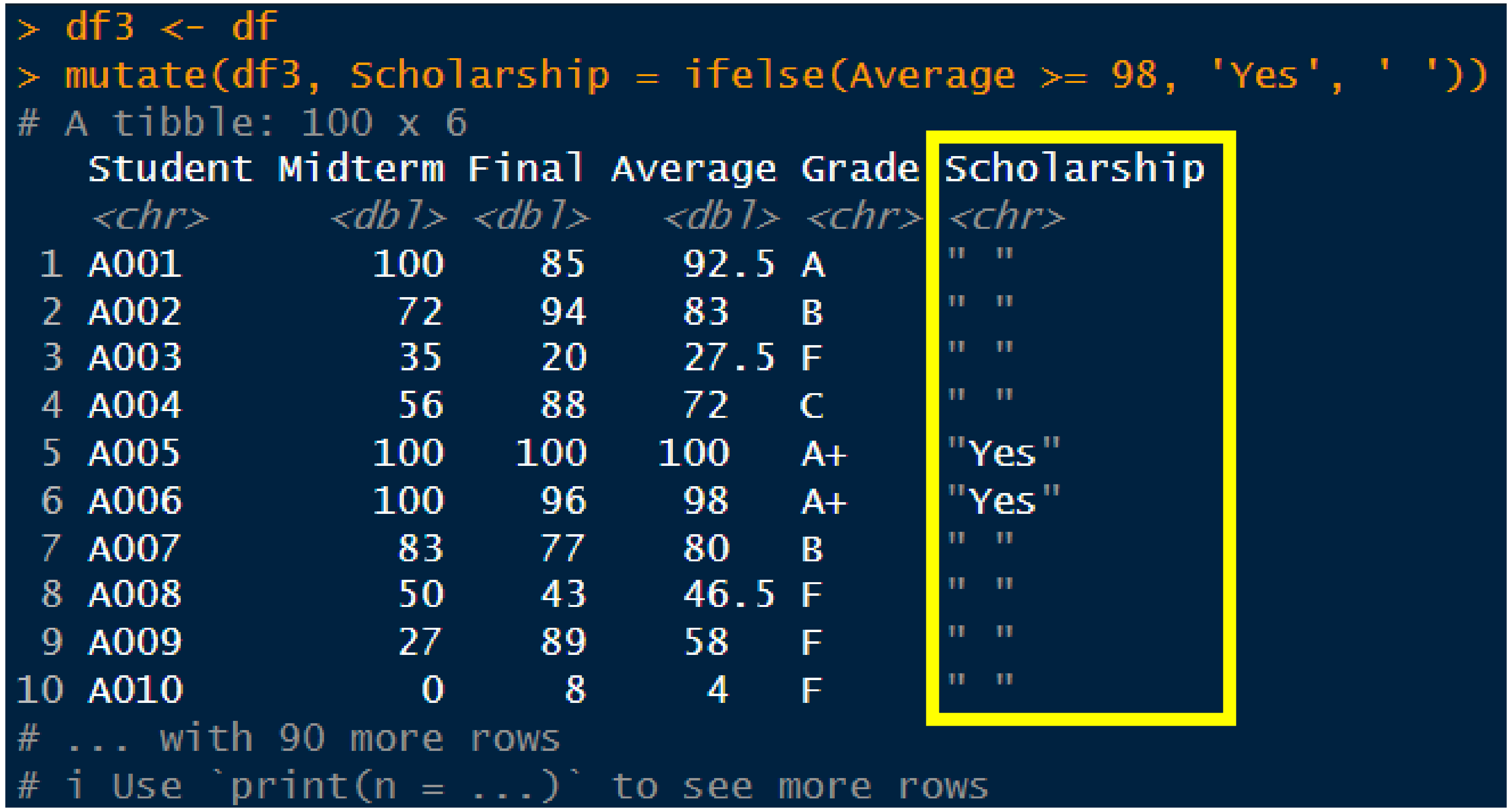
```
> df %>% distinct(Grade)
# A tibble: 9 x 1
  Grade
  <chr>
1 A
2 B
3 F
4 C
5 A+
6 C+
7 D
8 D+
9 B+
```

dplyr function 6: mutate()

- Make a new column

dplyr function 6: mutate()

- Ex) Make a ‘Scholarship’ column and show ‘Yes’ if ‘Average’ is same or bigger than 98

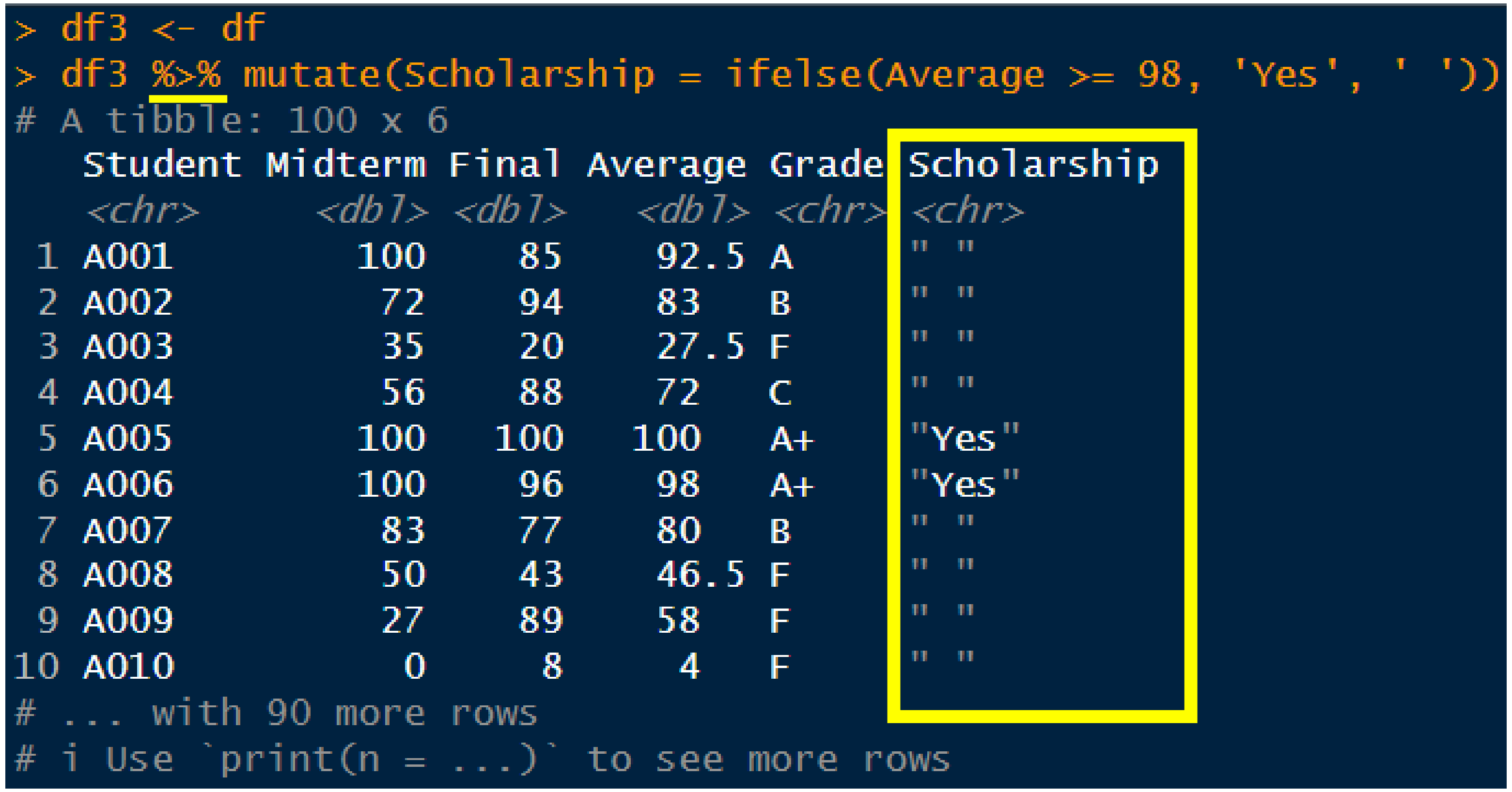


```
> df3 <- df
> mutate(df3, Scholarship = ifelse(Average >= 98, 'Yes', ' '))
# A tibble: 100 x 6
   Student Midterm Final Average Grade Scholarship
   <chr>     <dbl> <dbl>   <dbl> <chr> <chr>
 1 A001        100    85    92.5 A     " "
 2 A002         72    94    83   B     " "
 3 A003         35    20    27.5 F     " "
 4 A004         56    88    72   C     " "
 5 A005        100   100   100   A+    "Yes"
 6 A006        100    96    98   A+    "Yes"
 7 A007         83    77    80   B     " "
 8 A008         50    43    46.5 F     " "
 9 A009         27    89    58   F     " "
10 A010          0     8     4   F     " "
# ... with 90 more rows
# i Use `print(n = ...)` to see more rows
```

dplyr function 6: mutate()

- Ex) Make a ‘Scholarship’ column and show ‘Yes’ if ‘Average’ is same or bigger than 98

Pipe operator (%>%) can also be applied



```
> df3 <- df
> df3 %>% mutate(Scholarship = ifelse(Average >= 98, 'Yes', ' '))
# A tibble: 100 x 6
   Student Midterm Final Average Grade Scholarship
   <chr>     <dbl> <dbl>   <dbl> <chr> <chr>
 1 A001        100    85    92.5 A     " "
 2 A002         72    94    83   B     " "
 3 A003         35    20    27.5 F     " "
 4 A004         56    88    72   C     " "
 5 A005        100   100   100   A+    "Yes"
 6 A006        100    96    98   A+    "Yes"
 7 A007         83    77    80   B     " "
 8 A008         50    43    46.5 F     " "
 9 A009         27    89    58   F     " "
10 A010          0     8     4   F     " "
# ... with 90 more rows
# i Use `print(n = ...)` to see more rows
```

Practice

1. Arrange ‘Average’ by descending order and make a new column ‘Scholarship’ and mark ‘Yes’ if ‘Average’ is 98 or higher. Change the name of ‘Student’ column to ‘ID’ and show student lists who get scholarship.