

# Solutions to Extra dplyr Practice

AW

8/11/2020

## Exercise 1

Create a new data frame with just the participant id, age, grade, and CTOPP Elision data. Try reordering the columns, like putting grade ahead of everything else.

*# 1. Solution*

```
b2_elision <- b2_clean %>%  
  select(grade, pid, age, elision_raw)  
  
b2_elision
```

```
## # A tibble: 53 x 4  
##   grade pid      age elision_raw  
##   <fct> <chr>   <dbl>      <dbl>  
## 1 1     BLC_200  7.63         32  
## 2 1     BLC_202  7.89         27  
## 3 K     BLC_203  6.39         18  
## 4 1     BLC_204  7.70         25  
## 5 2     BLC_205  7.97         34  
## 6 K     BLC_206  6.52         11  
## 7 K     BLC_207  6.12         19  
## 8 2     BLC_208  8.16         32  
## 9 2     BLC_210  8.02         29  
## 10 K    BLC_213  6.32         14  
## # ... with 43 more rows
```

## Exercise 2a

Create a new data frame with just 2nd grade W-J data. (Hint: Use `filter()`). You should end up with 19 subjects.

*# 2a. Solution*

```
b2_wj_grd2 <- b2_clean %>%  
  filter(grade == '2') %>%  
  select(pid, grade, wj_raw)  
  
b2_wj_grd2
```

```
## # A tibble: 19 x 3
##   pid      grade wj_raw
##   <chr>    <fct>  <dbl>
## 1 BLC_205 2         44
## 2 BLC_208 2         63
## 3 BLC_210 2         63
## 4 BLC_214 2         65
## 5 BLC_219 2         64
## 6 BLC_224 2         59
## 7 BLC_226 2         43
## 8 BLC_229 2         50
## 9 BLC_230 2         69
## 10 BLC_231 2         50
## 11 BLC_232 2         50
## 12 BLC_236 2         63
## 13 BLC_242 2         58
## 14 BLC_245 2         55
## 15 BLC_251 2         52
## 16 BLC_255 2         66
## 17 BLC_256 2         58
## 18 BLC_265 2         59
## 19 BLC_266 2         60
```

We end up with 19 second grade subjects.

## Exercise 2b

Still working with the same data frame in 2a (Hint: Use piping!), pick out subjects who scored 60 or above. You should end up with 8 subjects.

```
# 2b. Solution

# copied the first part from 2a...
b2_wj_grd2 <- b2_clean %>%
  filter(grade == '2') %>%
  select(pid, grade, wj_raw) %>%
  filter(wj_raw >= 60) # adding this filter line

b2_wj_grd2
```

```
## # A tibble: 8 x 3
##   pid      grade wj_raw
##   <chr>    <fct>  <dbl>
## 1 BLC_208 2         63
## 2 BLC_210 2         63
## 3 BLC_214 2         65
## 4 BLC_219 2         64
## 5 BLC_230 2         69
## 6 BLC_236 2         63
## 7 BLC_255 2         66
## 8 BLC_266 2         60
```

We are down to 8 second grade subjects with W-J scores  $\geq 60$ .

### Exercise 3

Create a new dataframe with 1st and 2nd grade data. (Hint: Use logical operators to get both grade levels)

#### *# 3. Solution*

```
b2_grd1and2 <- b2_clean %>%  
  filter(grade == '1' | grade == '2') # grade is '1' OR '2'  
  
b2_grd1and2
```

```
## # A tibble: 35 x 14  
##   pid   grade   age age_year ctopp_c_raw ctopp_l_raw elision_raw swe_raw  
##   <chr> <fct> <dbl>   <dbl>     <dbl>     <dbl>     <dbl>     <dbl>  
## 1 BLC_~ 1     7.63     7      35      24      32      64  
## 2 BLC_~ 1     7.89     7      24      17      27      65  
## 3 BLC_~ 1     7.70     7      26      20      25      51  
## 4 BLC_~ 2     7.97     7      29      23      34      36  
## 5 BLC_~ 2     8.16     8      32      23      32      71  
## 6 BLC_~ 2     8.02     8      24      23      29      65  
## 7 BLC_~ 2     8.62     8      32      20      27      71  
## 8 BLC_~ 1     7.61     7      32      18      33      70  
## 9 BLC_~ 2     8.22     8      68      24      26      56  
## 10 BLC_~ 1    7.16     7      38      29      23      39  
## # ... with 25 more rows, and 6 more variables: pde_raw <dbl>, swe_age <dbl>,  
## #   pde_age <dbl>, swe_grade <dbl>, pde_grade <dbl>, wj_raw <dbl>
```

#### *# another solution is to take out the Kindergarteners*

```
b2_grd1and2_alt <- b2_clean %>%  
  filter(grade != 'K') # grade is NOT 'K'  
  
b2_grd1and2_alt
```

```
## # A tibble: 35 x 14  
##   pid   grade   age age_year ctopp_c_raw ctopp_l_raw elision_raw swe_raw  
##   <chr> <fct> <dbl>   <dbl>     <dbl>     <dbl>     <dbl>     <dbl>  
## 1 BLC_~ 1     7.63     7      35      24      32      64  
## 2 BLC_~ 1     7.89     7      24      17      27      65  
## 3 BLC_~ 1     7.70     7      26      20      25      51  
## 4 BLC_~ 2     7.97     7      29      23      34      36  
## 5 BLC_~ 2     8.16     8      32      23      32      71  
## 6 BLC_~ 2     8.02     8      24      23      29      65  
## 7 BLC_~ 2     8.62     8      32      20      27      71  
## 8 BLC_~ 1     7.61     7      32      18      33      70  
## 9 BLC_~ 2     8.22     8      68      24      26      56  
## 10 BLC_~ 1    7.16     7      38      29      23      39  
## # ... with 25 more rows, and 6 more variables: pde_raw <dbl>, swe_age <dbl>,  
## #   pde_age <dbl>, swe_grade <dbl>, pde_grade <dbl>, wj_raw <dbl>
```

We get the same result!

## Exercise 4a

The `arrange()` function orders rows by values of a column or columns (low to high). Create a new dataframe where you arrange the `b2_clean` data by age (low to high).

*# 4a. Solution*

```
b2_arrange <- b2_clean %>%  
  arrange(age)
```

```
b2_arrange
```

```
## # A tibble: 53 x 14  
##   pid   grade   age age_year ctopp_c_raw ctopp_l_raw elision_raw swe_raw  
##   <chr> <fct> <dbl>   <dbl>     <dbl>     <dbl>     <dbl>     <dbl>  
## 1 BLC_~ K     5.84     5       39       36       13       15  
## 2 BLC_~ K     5.86     5       50       25        9       49  
## 3 BLC_~ K     5.88     5       30       31       27       32  
## 4 BLC_~ K     6.09     6       30       30       14        6  
## 5 BLC_~ K     6.10     6       49       40       10       11  
## 6 BLC_~ K     6.12     6       36       49       19        2  
## 7 BLC_~ K     6.17     6       46       52       23        6  
## 8 BLC_~ K     6.22     6       45       30       23       11  
## 9 BLC_~ K     6.32     6       57       24       17       56  
## 10 BLC_~ K     6.32     6       52       31       14       27  
## # ... with 43 more rows, and 6 more variables: pde_raw <dbl>, swe_age <dbl>,  
## #   pde_age <dbl>, swe_grade <dbl>, pde_grade <dbl>, wj_raw <dbl>
```

*# if you want to arrange by high to low, use desc()*

```
b2_arrange_hightolow <- b2_clean %>%  
  arrange(desc(age))
```

```
b2_arrange_hightolow
```

```
## # A tibble: 53 x 14  
##   pid   grade   age age_year ctopp_c_raw ctopp_l_raw elision_raw swe_raw  
##   <chr> <fct> <dbl>   <dbl>     <dbl>     <dbl>     <dbl>     <dbl>  
## 1 BLC_~ 2     8.64     8       26       13       30       75  
## 2 BLC_~ 2     8.62     8       32       20       27       71  
## 3 BLC_~ 2     8.53     8       29       22       27       71  
## 4 BLC_~ 2     8.42     8       25       19       25       63  
## 5 BLC_~ 2     8.32     8       30       23       22       61  
## 6 BLC_~ 2     8.32     8       53       22       26       49  
## 7 BLC_~ 2     8.31     8       24       19       30       65  
## 8 BLC_~ 2     8.24     8       29       23       27       69  
## 9 BLC_~ 2     8.22     8       68       24       26       56  
## 10 BLC_~ 2     8.16     8       32       23       32       71  
## # ... with 43 more rows, and 6 more variables: pde_raw <dbl>, swe_age <dbl>,  
## #   pde_age <dbl>, swe_grade <dbl>, pde_grade <dbl>, wj_raw <dbl>
```

## Exercise 4b

Use `drop_na()` to drop rows (subjects) that have any NA values in their scores.

*# 4b. Solution*

```
b2_arrange_dropna <- b2_clean %>%  
  arrange(age) %>%  
  drop_na() # this added line drops entire subjects with NAs (i.e. drops entire rows)  
  
b2_arrange_dropna
```

```
## # A tibble: 34 x 14  
##   pid   grade  age age_year ctopp_c_raw ctopp_l_raw elision_raw swe_raw  
##   <chr> <fct> <dbl>   <dbl>     <dbl>     <dbl>     <dbl>   <dbl>  
## 1 BLC_~ 1     6.70     6      47      31      26     58  
## 2 BLC_~ 1     6.85     6      37      19      17     66  
## 3 BLC_~ 1     6.87     6      42      19      30     61  
## 4 BLC_~ 1     7.11     7      48      32      22     42  
## 5 BLC_~ 1     7.15     7      48      40      22     14  
## 6 BLC_~ 1     7.16     7      38      29      23     39  
## 7 BLC_~ 1     7.26     7      29      17      25     64  
## 8 BLC_~ 1     7.38     7      32      23      21     63  
## 9 BLC_~ 1     7.49     7      41      20      27     41  
## 10 BLC_~ 1     7.51     7      36      21      23     49  
## # ... with 24 more rows, and 6 more variables: pde_raw <dbl>, swe_age <dbl>,  
## #   pde_age <dbl>, swe_grade <dbl>, pde_grade <dbl>, wj_raw <dbl>
```

We get 34 remaining subjects, and NAs shouldn't exist in the data anymore. To double check, we can use the `is.na()` function and compare our data before and after we applied `drop_na()`.

Returns:

- TRUE = we have NAs in the data
- FALSE = no NAs in the data

*# First look at the data before we dropped NAs.*

```
any(is.na(b2_arrange))
```

```
## [1] TRUE
```

*# Now check the data after we applied drop\_na().*

```
any(is.na(b2_arrange_dropna))
```

```
## [1] FALSE
```