Manipulating Data Frames

```
# As usual, we load the mosaic and tidyverse packages.
# Loading tidyverse automatically loads the dplyr package, which is
# the package that provides many of the operations illustrated here.
library(mosaic)
library(tidyverse)

# We make use of the run09 data frame, which is included in the
# cherryblossom package.
library(cherryblossom)
head(run09)

## # A tibble: 6 x 14
##   place  time net_time  pace   age gender first  last  city  state country
div
##   <int> <dbl>    <dbl> <dbl> <int> <fct>  <fct>  <fct> <fct> <fct> <fct>
<int>
## 1     1  53.5     53.5  5.37    21 F      Lineth Chep~ Kenya NR    KEN
2
## 2     2  53.9     53.9  5.4     21 F      Belia~ Gebre Ethi~ NR    ETH
2
## 3     3  54.0     54.0  5.4     22 F      Teyba  Naser Ethi~ NR    ETH
2
## 4     4  54.4     54.4  5.45    19 F      Abebu  Gelan Ethi~ NR    ETH
1
## 5     5  54.4     54.4  5.45    36 F      Cathe~ Nder~ Kenya NR    KEN
5
## 6     6  54.5     54.5  5.47    28 F      Olga   Roma~ Russ~ NR    RUS
3
## # ... with 2 more variables: div_place <int>, div_tot <int>

# Selecting Columns.
# To select certain columns from a data frame, we use the select()
# operation.  Here, we select the net_time, age, gender, last, state,
# country, and div columns from run09, assigning the resulting data
# frame to data01.
data01 <- select(run09, net_time, age, gender, last,
                 state, country, div)
head(data01)

## # A tibble: 6 x 7
##   net_time   age gender last      state country   div
##      <dbl> <int> <fct>  <fct>     <fct> <fct>   <int>
## 1     53.5    21 F      Chepkurui NR    KEN         2
## 2     53.9    21 F      Gebre     NR    ETH         2
## 3     54.0    22 F      Naser     NR    ETH         2
## 4     54.4    19 F      Gelan     NR    ETH         1
```

```
## 5     54.4    36 F      Ndereba   NR    KEN         5
## 6     54.5    28 F      Romanova  NR    RUS         3

# The columns are ordered in the order they are listed in the
# select() command.  Let's do it again, putting the columns in
# a different order, and renaming the net_time column.
# Also, most data scientists prefer to use the "piping" operator,
# %>%.
data02 <- run09 %>%
  select(last, gender, age, time = net_time,
         state, country, div)
head(data02)

## # A tibble: 6 x 7
##   last      gender   age  time state country   div
##   <fct>     <fct>  <int> <dbl> <fct> <fct>   <int>
## 1 Chepkurui F         21  53.5 NR    KEN         2
## 2 Gebre     F         21  53.9 NR    ETH         2
## 3 Naser     F         22  54.0 NR    ETH         2
## 4 Gelan     F         19  54.4 NR    ETH         1
## 5 Ndereba   F         36  54.4 NR    KEN         5
## 6 Romanova  F         28  54.5 NR    RUS         3

# Read the above as code follows: Take the run09 data frame,
# and pipe it through the select operation; then assign the
# result to data02.
#
# Selecting Rows.
# To select the rows from a data frame that satisfy certain
# criteria, we use the filter() operation.  Let's "filter in"
# the rows of data02 corresponding to male runners from the USA.
# Again, we use piping.
data03 <- data02 %>%
  filter(gender == "M" & country == "USA")
head(data03)

## # A tibble: 6 x 7
##   last      gender   age  time state country   div
##   <fct>     <fct>  <int> <dbl> <fct> <fct>   <int>
## 1 McDonald  M         49  55.1 VA    USA         7
## 2 Kelly     M         53  77.9 NC    USA         8
## 3 Hartman   M         28  48.0 CO    USA         3
## 4 Lehmkuhle M         31  48.1 MN    USA         4
## 5 Morgan    M         29  48.1 NR    USA         3
## 6 Fasulo    M         27  50.7 PA    USA         3

# The View() command can be used to look at a data frame in a separate
# tab.  After executing the command, it is recommended that it be
# commented out, so that it doesn't "mess" with knitting.
# View(data03)
```

```r
# Note that there is at least one row where state = "NR", which
# I'm guessing means that runner failed to provide his state.  So,
# let's filter out any such rows; also, we no longer need country
# and gender, so let's select the other columns.
data04 <- data03 %>%
  filter(state != "NR") %>%
  select(!(country|gender))
head(data04)
```

```
## # A tibble: 6 x 5
##   last        age  time state   div
##   <fct>     <int> <dbl> <fct> <int>
## 1 McDonald     49  55.1 VA        7
## 2 Kelly        53  77.9 NC        8
## 3 Hartman      28  48.0 CO        3
## 4 Lehmkuhle    31  48.1 MN        4
## 5 Fasulo       27  50.7 PA        3
## 6 Rodriguez    29  51.1 VA        3
```

```r
# Modifying/Adding Columns.
# The mutate() operation can be used to modify an existing column,
# or to add columns.  The type of the variable last should not (in
# my opinion) be "factor" - think of factor as indicating that the
# value could be chosen from a drop-down menu.  For example, it
# makes sense for the gender and state variables to be of type
# factor.  Rather, the type of last should be "character".  Let's
# apply mutate() and the as.character() operation to fix this.
data05 <- data04 %>%
  mutate(last = as.character(last))
head(data05)
```

```
## # A tibble: 6 x 5
##   last        age  time state   div
##   <chr>     <int> <dbl> <fct> <int>
## 1 McDonald     49  55.1 VA        7
## 2 Kelly        53  77.9 NC        8
## 3 Hartman      28  48.0 CO        3
## 4 Lehmkuhle    31  48.1 MN        4
## 5 Fasulo       27  50.7 PA        3
## 6 Rodriguez    29  51.1 VA        3
```

```r
# Let's add a new column, adj_time (adjusted time), defined as
# follows: adj_time = time - age.
data06 <- data05 %>%
  mutate(adj_time = time - age)
head(data06)
```

```
## # A tibble: 6 x 6
##   last        age  time state   div adj_time
##   <chr>     <int> <dbl> <fct> <int>    <dbl>
## 1 McDonald     49  55.1 VA        7     6.07
```

```
## 2 Kelly        53  77.9 NC        8    24.9
## 3 Hartman      28  48.0 CO        3    20.0
## 4 Lehmkuhle    31  48.1 MN        4    17.1
## 5 Fasulo       27  50.7 PA        3    23.7
## 6 Rodriguez    29  51.1 VA        3    22.1

# Renaming Columns.
# The rename() operation can be used to rename a column.  Let's
# rename "last" as "last_name".  Note: This could have been done
# above as part of selection/mutation.
data07 <- data06 %>%
  rename(last_name = last)
head(data07)

## # A tibble: 6 x 6
##   last_name   age  time state   div adj_time
##   <chr>     <int> <dbl> <fct> <int>    <dbl>
## 1 McDonald     49  55.1 VA        7     6.07
## 2 Kelly        53  77.9 NC        8    24.9
## 3 Hartman      28  48.0 CO        3    20.0
## 4 Lehmkuhle    31  48.1 MN        4    17.1
## 5 Fasulo       27  50.7 PA        3    23.7
## 6 Rodriguez    29  51.1 VA        3    22.1

# Ordering Rows.
# The arrange() operation can be used to order the rows according to
# the values in one (or more) columns.  Here, we arrange the rows in
# order of increasing time, and then we arrange the rows in order of
# decreasing age.
data08 <- data07 %>%
  arrange(time)
head(data08)

## # A tibble: 6 x 6
##   last_name   age  time state   div adj_time
##   <chr>     <int> <dbl> <fct> <int>    <dbl>
## 1 Hartman      28  48.0 CO        3     20.0
## 2 Lehmkuhle    31  48.1 MN        4     17.1
## 3 Fasulo       27  50.7 PA        3     23.7
## 4 Rodriguez    29  51.1 VA        3     22.1
## 5 Smits        30  51.2 MD        4     21.2
## 6 Zins         29  51.2 PA        3     22.2

data09 <- data07 %>%
  arrange(desc(age))
head(data09)

## # A tibble: 6 x 6
##   last_name    age  time state   div adj_time
##   <chr>      <int> <dbl> <fct> <int>    <dbl>
## 1 Lodovico      85  95.2 PA       14    10.2
```

```
## 2 Xie           81 100.  MD       14    19.0
## 3 Yannakakis    77  85.5 MD       13     8.52
## 4 Momiyama      77 102.  MD       13    25.1
## 5 Green         77 109.  DC       13    31.8
## 6 Lewis         75  85.4 VA       13    10.4

# Aggregating Rows: group_by() and summarize().
# Example: Produce a data frame that gives the number of runners
# in, and the minimum time, for each division.
data10 <- data09 %>%
  group_by(div) %>%
  summarize(number = n(), min_time = min(time))
data10

## # A tibble: 14 x 3
##      div number min_time
##    <int>  <int>    <dbl>
##  1     1     61     56.2
##  2     2    395     52.8
##  3     3   1410     48.0
##  4     4   1263     48.1
##  5     5   1034     54.3
##  6     6    864     NA
##  7     7    582     54.9
##  8     8    474     55.7
##  9     9    285     59.7
## 10    10    160     65
## 11    11     65     66.5
## 12    12     19     74.0
## 13    13      8     85.4
## 14    14      2     95.2

# Apparently, the time is missing for at least one runner in
# division 6.  Let's filter out any such rows from data09, and
# then try again.
data09alt <- data09 %>%
  filter(time != "NA")
data10alt <- data09alt %>%
  group_by(div) %>%
  summarize(number = n(), min_time = min(time))
data10alt

## # A tibble: 14 x 3
##      div number min_time
##    <int>  <int>    <dbl>
##  1     1     61     56.2
##  2     2    395     52.8
##  3     3   1410     48.0
##  4     4   1263     48.1
##  5     5   1034     54.3
##  6     6    863     52.1
```

```
##  7     7    582     54.9
##  8     8    474     55.7
##  9     9    285     59.7
## 10    10    160     65
## 11    11     65     66.5
## 12    12     19     74.0
## 13    13      8     85.4
## 14    14      2     95.2

# Bingo!
# Another Example: The state with the best average adjusted time
# receives a trophy.  To qualify, the state must have at least 10
# runners.
data11 <- data09alt %>%
  group_by(state) %>%
  summarize(number = n(), mean_adj_time = mean(adj_time)) %>%
  filter(number > 9) %>%
  arrange(mean_adj_time)
head(data11,10)

## # A tibble: 10 x 3
##    state number mean_adj_time
##    <fct>  <int>         <dbl>
##  1 NH        10          34.6
##  2 GA        20          36.9
##  3 IL        27          42.0
##  4 TX        15          42.4
##  5 WV        16          44.7
##  6 OH        34          46.5
##  7 PA       217          46.7
##  8 DE        27          46.9
##  9 NY       190          47.5
## 10 NJ        81          47.7

# And the winner is ... New Hampshire!
# Note that they had just enough runners to qualify.
#
# Joining Data Frames Over a Common Column
# There are several versions of the join operation. To illustrate
# them, let's build a couple of small data frames.
init <- c("AA","BB","CC","DD","EE")
id <- c("BB","CC","DD","EE","FF")
var1 <- seq(11,19,2)
var2 <- seq(21,33,3)
df1 <- tibble(init,var1)
df1

## # A tibble: 5 x 2
##   init   var1
##   <chr> <dbl>
## 1 AA       11
```

```
## 2 BB       13
## 3 CC       15
## 4 DD       17
## 5 EE       19

df2 <- tibble(id,var2)
df2

## # A tibble: 5 x 2
##   id     var2
##   <chr> <dbl>
## 1 BB       21
## 2 CC       24
## 3 DD       27
## 4 EE       30
## 5 FF       33

# Using inner_join() to join df1 with df2.
join1 <- df1 %>%
  inner_join(df2, by = c("init" = "id"))
join1

## # A tibble: 4 x 3
##   init   var1  var2
##   <chr> <dbl> <dbl>
## 1 BB       13    21
## 2 CC       15    24
## 3 DD       17    27
## 4 EE       19    30

# Using inner_join() to join df2 with df1.
join2 <- df2 %>%
  inner_join(df1, by = c("id" = "init"))
join2

## # A tibble: 4 x 3
##   id     var2  var1
##   <chr> <dbl> <dbl>
## 1 BB       21    13
## 2 CC       24    15
## 3 DD       27    17
## 4 EE       30    19

# Note that the inner_join() operation is essentially
# "commutative" - except for the order of the columns in the
# result.  That is, we get essentially the same result whether
# we inner_join df1 with df2 or df2 with df1.
# Also note that we get only those ids/inits that are common to
# both data frames.
# Contrast this with the results produced by left_join().
join3 <- df1 %>%
```

```
  left_join(df2, by = c("init" = "id"))
join3
```

```
## # A tibble: 5 x 3
##   init   var1  var2
##   <chr> <dbl> <dbl>
## 1 AA       11    NA
## 2 BB       13    21
## 3 CC       15    24
## 4 DD       17    27
## 5 EE       19    30
```

```
# Note that we get all the information from df1, and the
# values for var2 are added where defined.
# Let's try it the other way.
join4 <- df2 %>%
  left_join(df1, by = c("id" = "init"))
join4
```

```
## # A tibble: 5 x 3
##   id     var2  var1
##   <chr> <dbl> <dbl>
## 1 BB       21    13
## 2 CC       24    15
## 3 DD       27    17
## 4 EE       30    19
## 5 FF       33    NA
```

```
# Here, we get all the information from df2, with the values
# for var1 added where defined.
# In short, the left_join operation is not commutative.
```