Session 2: Data Manipulation

Dave Margraf

June 20, 2018

Contents

The dplyr package

The provides a grammar for data manipulation.

Load the dplyr package and data

```
library(dplyr)
```

Load the Theoph data set and save it as a data frame.

```
df <- data.frame(Theoph)
```

We can use the dim() and head() functions from base R to find the dimensions and take a look at the data.

```
dim(df)
```

```
## [1] 132   5
```

```
head(df)
```

```
##   Subject   Wt Dose Time  conc
## 1       1 79.6 4.02 0.00  0.74
## 2       1 79.6 4.02 0.25  2.84
## 3       1 79.6 4.02 0.57  6.57
## 4       1 79.6 4.02 1.12 10.50
## 5       1 79.6 4.02 2.02  9.66
## 6       1 79.6 4.02 3.82  8.58
```

Alternately, we can load the data as a tibble, which is a specialized data frame, with the `as_tibble()` function.

```
df <- as_tibble(Theoph)

df
```

```
## # A tibble: 132 x 5
##    Subject    Wt  Dose   Time  conc
##  * <ord>   <dbl> <dbl>  <dbl> <dbl>
##  1 1        79.6  4.02  0      0.74
##  2 1        79.6  4.02  0.25   2.84
##  3 1        79.6  4.02  0.570  6.57
##  4 1        79.6  4.02  1.12  10.5
##  5 1        79.6  4.02  2.02   9.66
##  6 1        79.6  4.02  3.82   8.58
##  7 1        79.6  4.02  5.1    8.36
##  8 1        79.6  4.02  7.03   7.47
##  9 1        79.6  4.02  9.05   6.89
## 10 1        79.6  4.02 12.1    5.94
## # ... with 122 more rows
```

Variable definitions for the `Theoph` data set:

- Wt - weight of the subject (kg)
- Dose - dose of theophylline administered orally to the subject (mg/kg)
- Time - time since drug administration when the sample was drawn (hr)
- conc - theophylline concentration in the sample (mg/L)

`filter()`

The filter verb subsets the data by rows (observations). That is, it extracts particular observations based their values.

Let's subset the theophylline data by weight of 70 kg or more.

```
filter(df, Wt >= 70)
```

```
## # A tibble: 77 x 5
##    Subject    Wt  Dose   Time  conc
##    <ord>   <dbl> <dbl>  <dbl> <dbl>
##  1 1        79.6  4.02  0      0.74
##  2 1        79.6  4.02  0.25   2.84
##  3 1        79.6  4.02  0.570  6.57
##  4 1        79.6  4.02  1.12  10.5
##  5 1        79.6  4.02  2.02   9.66
##  6 1        79.6  4.02  3.82   8.58
##  7 1        79.6  4.02  5.1    8.36
##  8 1        79.6  4.02  7.03   7.47
```

```
##  9 1        79.6  4.02  9.05   6.89
## 10 1        79.6  4.02 12.1    5.94
## # ... with 67 more rows
```

We can subset the data further with additional arguments.

```
filter(df, Wt >= 70, Dose >= 4)
```

```
## # A tibble: 66 x 5
##    Subject    Wt  Dose   Time  conc
##    <ord>   <dbl> <dbl>  <dbl> <dbl>
##  1 1        79.6  4.02  0      0.74
##  2 1        79.6  4.02  0.25   2.84
##  3 1        79.6  4.02  0.570  6.57
##  4 1        79.6  4.02  1.12  10.5
##  5 1        79.6  4.02  2.02   9.66
##  6 1        79.6  4.02  3.82   8.58
##  7 1        79.6  4.02  5.1    8.36
##  8 1        79.6  4.02  7.03   7.47
##  9 1        79.6  4.02  9.05   6.89
## 10 1        79.6  4.02 12.1    5.94
## # ... with 56 more rows
```

select()

The select verb subsets the data by columns (variables). That is, it extracts particular variables based on their names.

We can extract a vector by naming one variable.

```
select(df, conc)
```

```
## # A tibble: 132 x 1
##     conc
##  * <dbl>
##  1  0.74
##  2  2.84
##  3  6.57
##  4 10.5
##  5  9.66
##  6  8.58
##  7  8.36
##  8  7.47
##  9  6.89
## 10  5.94
## # ... with 122 more rows
```

We can drop variables as well. Just place a minus sign in front of the variable you want to remove. The other variables will remain.

```
select(df, -Wt)
```

```
## # A tibble: 132 x 4
##    Subject  Dose   Time  conc
##  * <ord>   <dbl>  <dbl> <dbl>
##  1 1        4.02  0      0.74
##  2 1        4.02  0.25   2.84
```

```
##  3 1        4.02  0.570  6.57
##  4 1        4.02  1.12  10.5
##  5 1        4.02  2.02   9.66
##  6 1        4.02  3.82   8.58
##  7 1        4.02  5.1    8.36
##  8 1        4.02  7.03   7.47
##  9 1        4.02  9.05   6.89
## 10 1        4.02 12.1    5.94
## # ... with 122 more rows
```

Variables can be moved around if needed. Placing the `everything()` helper function will fill in the remaining variables you do not mention.

```
select(df, Time, Subject, everything())
```

```
## # A tibble: 132 x 5
##      Time Subject    Wt  Dose  conc
##  *  <dbl> <ord>   <dbl> <dbl> <dbl>
##  1  0     1        79.6  4.02  0.74
##  2  0.25  1        79.6  4.02  2.84
##  3  0.570 1        79.6  4.02  6.57
##  4  1.12  1        79.6  4.02 10.5
##  5  2.02  1        79.6  4.02  9.66
##  6  3.82  1        79.6  4.02  8.58
##  7  5.1   1        79.6  4.02  8.36
##  8  7.03  1        79.6  4.02  7.47
##  9  9.05  1        79.6  4.02  6.89
## 10 12.1   1        79.6  4.02  5.94
## # ... with 122 more rows
```

If you want to move a variable to the end of the data set, subtract then add it back. Also, you can rename variables within any `select()` function.

```
select(df, -Wt, weight=Wt)
```

```
## # A tibble: 132 x 5
##    Subject  Dose   Time  conc weight
##  * <ord>   <dbl>  <dbl> <dbl>  <dbl>
##  1 1        4.02  0      0.74   79.6
##  2 1        4.02  0.25   2.84   79.6
##  3 1        4.02  0.570  6.57   79.6
##  4 1        4.02  1.12  10.5    79.6
##  5 1        4.02  2.02   9.66   79.6
##  6 1        4.02  3.82   8.58   79.6
##  7 1        4.02  5.1    8.36   79.6
##  8 1        4.02  7.03   7.47   79.6
##  9 1        4.02  9.05   6.89   79.6
## 10 1        4.02 12.1    5.94   79.6
## # ... with 122 more rows
```

rename()

The rename verb keeps all variables unlike select, which keeps only the variables you mention.

```
rename(df, weight = Wt)
```

```
## # A tibble: 132 x 5
##    Subject weight  Dose   Time  conc
##  * <ord>    <dbl> <dbl>  <dbl> <dbl>
##  1 1         79.6  4.02  0      0.74
##  2 1         79.6  4.02  0.25   2.84
##  3 1         79.6  4.02  0.570  6.57
##  4 1         79.6  4.02  1.12  10.5
##  5 1         79.6  4.02  2.02   9.66
##  6 1         79.6  4.02  3.82   8.58
##  7 1         79.6  4.02  5.1    8.36
##  8 1         79.6  4.02  7.03   7.47
##  9 1         79.6  4.02  9.05   6.89
## 10 1         79.6  4.02 12.1    5.94
## # ... with 122 more rows
```

An aside, the pipe operator %>%

Takes the result from the left hand side and passes it into the function on the right hand side. This allows you to code in a more readable left-to-right fashion rather than nesting function within one another. For example,

Let's practice using the using the filter verb to find the observations for the first subject.

```
df %>% filter(Subject == 3)
```

```
## # A tibble: 11 x 5
##    Subject    Wt  Dose   Time  conc
##    <ord>   <dbl> <dbl>  <dbl> <dbl>
##  1 3        70.5  4.53  0      0
##  2 3        70.5  4.53  0.27   4.4
##  3 3        70.5  4.53  0.580  6.9
##  4 3        70.5  4.53  1.02   8.2
##  5 3        70.5  4.53  2.02   7.8
##  6 3        70.5  4.53  3.62   7.5
##  7 3        70.5  4.53  5.08   6.2
##  8 3        70.5  4.53  7.07   5.3
##  9 3        70.5  4.53  9      4.9
## 10 3        70.5  4.53 12.2    3.7
## 11 3        70.5  4.53 24.2    1.05
```

We can chain pipes together to really benefit from its usefulness. Find the observed Cmax for subject three.

```
df %>%
  filter(Subject == 3) %>%
  select(conc) %>%
  max()
```

```
## [1] 8.2
```

mutate()

The mutate verb adds new variables.

New variables can be made that are functions of existing variables. For example, perhaps we want to express time in seconds rather than hours, or convert weight in kilograms to pounds.

Let's save this to `df` with the assignment operator `<-`.

```
df <- df %>%
  mutate(minutes = Time * 60,
         lbs = Wt * 2.2046)

df
```

```
## # A tibble: 132 x 7
##    Subject    Wt  Dose   Time  conc minutes   lbs
##    <ord>   <dbl> <dbl>  <dbl> <dbl>   <dbl> <dbl>
##  1 1        79.6  4.02  0      0.74     0    175.
##  2 1        79.6  4.02  0.25   2.84    15    175.
##  3 1        79.6  4.02  0.570  6.57    34.2  175.
##  4 1        79.6  4.02  1.12  10.5     67.2  175.
##  5 1        79.6  4.02  2.02   9.66   121.   175.
##  6 1        79.6  4.02  3.82   8.58   229.   175.
##  7 1        79.6  4.02  5.1    8.36   306    175.
##  8 1        79.6  4.02  7.03   7.47   422.   175.
##  9 1        79.6  4.02  9.05   6.89   543    175.
## 10 1        79.6  4.02 12.1    5.94   727.   175.
## # ... with 122 more rows
```

arrange()

The arrange verb changes the ordering of the rows.

Sort the data by increasing weight.

```
df %>% arrange(lbs)
```

```
## # A tibble: 132 x 7
##    Subject    Wt  Dose  Time  conc minutes   lbs
##    <ord>   <dbl> <dbl> <dbl> <dbl>   <dbl> <dbl>
##  1 5        54.6  5.86  0     0        0    120.
##  2 5        54.6  5.86  0.3   2.02    18    120.
##  3 5        54.6  5.86  0.52  5.63    31.2  120.
##  4 5        54.6  5.86  1    11.4     60    120.
##  5 5        54.6  5.86  2.02  9.33   121.   120.
##  6 5        54.6  5.86  3.5   8.74   210    120.
##  7 5        54.6  5.86  5.02  7.56   301.   120.
##  8 5        54.6  5.86  7.02  7.09   421.   120.
##  9 5        54.6  5.86  9.1   5.9    546    120.
## 10 5        54.6  5.86 12     4.37   720    120.
## # ... with 122 more rows
```

Use `desc()` to sort a variable in descending order.

```
df %>% arrange(desc(lbs))
```

```
## # A tibble: 132 x 7
##    Subject    Wt  Dose  Time  conc minutes   lbs
##    <ord>   <dbl> <dbl> <dbl> <dbl>   <dbl> <dbl>
##  1 9        86.4   3.1  0     0        0    190.
##  2 9        86.4   3.1  0.3   7.37    18    190.
##  3 9        86.4   3.1  0.63  9.03    37.8  190.
##  4 9        86.4   3.1  1.05  7.14    63    190.
```

```
##  5 9        86.4   3.1  2.02  6.33   121.   190.
##  6 9        86.4   3.1  3.53  5.66   212.   190.
##  7 9        86.4   3.1  5.02  5.67   301.   190.
##  8 9        86.4   3.1  7.17  4.24   430.   190.
##  9 9        86.4   3.1  8.8   4.11   528    190.
## 10 9        86.4   3.1 11.6   3.16   696    190.
## # ... with 122 more rows
```

Adding verbs together.

```
df %>%
  filter(Time ==0) %>%
  arrange(desc(lbs))
```

```
## # A tibble: 12 x 7
##    Subject    Wt  Dose  Time  conc minutes   lbs
##    <ord>   <dbl> <dbl> <dbl> <dbl>   <dbl> <dbl>
##  1 9        86.4  3.1      0  0           0  190.
##  2 6        80    4        0  0           0  176.
##  3 1        79.6  4.02     0  0.74        0  175.
##  4 4        72.7  4.4      0  0           0  160.
##  5 2        72.4  4.4      0  0           0  160.
##  6 3        70.5  4.53     0  0           0  155.
##  7 8        70.5  4.53     0  0           0  155.
##  8 11       65    4.92     0  0           0  143.
##  9 7        64.6  4.95     0  0.15        0  142.
## 10 12       60.5  5.3      0  0           0  133.
## 11 10       58.2  5.5      0  0.24        0  128.
## 12 5        54.6  5.86     0  0           0  120.
```

By subsetting and sorting the data we can see that three subjects have positive drug concentrations at time zero, and dose appears to be inversely proportional to weight.

group_by()

You will usually want to group data by some variable.

Grouping doesn't change how the data looks (apart from listing how it's grouped), but it does change how it acts with the other `dplyr` verbs.

```
df %>%
  group_by(Subject)
```

```
## # A tibble: 132 x 7
## # Groups:   Subject [12]
##    Subject    Wt  Dose   Time  conc minutes   lbs
##    <ord>   <dbl> <dbl>  <dbl> <dbl>   <dbl> <dbl>
##  1 1        79.6  4.02  0      0.74     0    175.
##  2 1        79.6  4.02  0.25   2.84    15    175.
##  3 1        79.6  4.02  0.570  6.57    34.2  175.
##  4 1        79.6  4.02  1.12  10.5     67.2  175.
##  5 1        79.6  4.02  2.02   9.66   121.   175.
##  6 1        79.6  4.02  3.82   8.58   229.   175.
##  7 1        79.6  4.02  5.1    8.36   306    175.
##  8 1        79.6  4.02  7.03   7.47   422.   175.
##  9 1        79.6  4.02  9.05   6.89   543    175.
```

```
## 10 1        79.6  4.02 12.1    5.94   727.   175.
## # ... with 122 more rows
```

Now we can create a new columns specific to each subject with `group_by()` and `mutate()`. Let's find the Cmax and Tmax for each concentration-time profile. Since Tmax is related to the pharmacokinetic parameter Cmax, we can use the `case_when()` function to identify the time when Cmax is observed. This observation is saved in a temporary variable, `temp`, then dropped with the `select()` verb.

```
df %>%
  group_by(Subject) %>%
  mutate(Cmax = max(conc),
         temp = case_when(conc == Cmax ~ Time),
         Tmax = max(temp, na.rm = T)) %>%
  select(-temp)
```

```
## # A tibble: 132 x 9
## # Groups:   Subject [12]
##    Subject    Wt  Dose   Time  conc minutes   lbs  Cmax  Tmax
##    <ord>   <dbl> <dbl>  <dbl> <dbl>   <dbl> <dbl> <dbl> <dbl>
##  1 1        79.6  4.02  0      0.74     0    175.  10.5  1.12
##  2 1        79.6  4.02  0.25   2.84    15    175.  10.5  1.12
##  3 1        79.6  4.02  0.570  6.57    34.2  175.  10.5  1.12
##  4 1        79.6  4.02  1.12  10.5     67.2  175.  10.5  1.12
##  5 1        79.6  4.02  2.02   9.66   121.   175.  10.5  1.12
##  6 1        79.6  4.02  3.82   8.58   229.   175.  10.5  1.12
##  7 1        79.6  4.02  5.1    8.36   306    175.  10.5  1.12
##  8 1        79.6  4.02  7.03   7.47   422.   175.  10.5  1.12
##  9 1        79.6  4.02  9.05   6.89   543    175.  10.5  1.12
## 10 1        79.6  4.02 12.1    5.94   727.   175.  10.5  1.12
## # ... with 122 more rows
```

summarise()

The summarise verb reduces multiple values down to a single summary.

```
df %>%
  summarise(meanWt = mean(Wt),
            medWt = median(Wt),
            n = n_distinct(Subject))
```

```
## # A tibble: 1 x 3
##   meanWt medWt     n
##    <dbl> <dbl> <int>
## 1   69.6  70.5    12
```

You may want to group data before summarizing.

```
df %>%
  group_by(Wt < 70) %>%
  summarise(medDose = median(Dose),
            meanDose = mean(Dose),
            sdDose = sd(Dose))
```

```
## # A tibble: 2 x 4
##   `Wt < 70` medDose meanDose sdDose
##   <lgl>       <dbl>    <dbl>  <dbl>
```

```
## 1 FALSE         4.4     4.14  0.474
## 2 TRUE          5.3     5.31  0.355
```

Let's build a data set

Some useful base R functions:

`seq()` generates regular sequences.
`rep()` replicates values.
`length()` gets or sets the length of vectors (including lists) and factors.
`unique()` returns a vector, data frame or array like `x` but with duplicate elements/rows removed.
`sample()` takes a random sample of the specified size from the elements of `x` either with or without replacement.
`round()` rounds the values to the specified number of decimal places (default 0).

Statistical functions in the `stats` package.

`rnorm()` random generation for the normal distribution with mean equal to `mean` and standard deviation equal to `sd`. `runif()` generates random deviates about the uniform distribution on the interval from `min` to `max`.

Subjects

To create a vector for 20 subjects we can start with the `seq()` function.

```
seq(1:20)
```

```
##  [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20
```

If we want longitudinal (repeated measures) data we can pipe this into the `rep()` function.

```
seq(1:20) %>% rep(10)
```

```
##   [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20  1  2  3
##  [24]  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20  1  2  3  4  5  6
##  [47]  7  8  9 10 11 12 13 14 15 16 17 18 19 20  1  2  3  4  5  6  7  8  9
##  [70] 10 11 12 13 14 15 16 17 18 19 20  1  2  3  4  5  6  7  8  9 10 11 12
##  [93] 13 14 15 16 17 18 19 20  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15
## [116] 16 17 18 19 20  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18
## [139] 19 20  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20  1
## [162]  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20  1  2  3  4
## [185]  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20
```

This isn't quite right. We could use `arrange()` to fix this, but an easier way is to use the `each =` argument in `rep()`. Note: using `rep(10)` is equivalent to `rep(times=10)`.

```
  seq(1:20) %>% rep(each=10)
```

```
##   [1]  1  1  1  1  1  1  1  1  1  1  2  2  2  2  2  2  2  2  2  2  3  3  3
##  [24]  3  3  3  3  3  3  3  4  4  4  4  4  4  4  4  4  4  5  5  5  5  5  5
##  [47]  5  5  5  5  6  6  6  6  6  6  6  6  6  6  7  7  7  7  7  7  7  7  7
##  [70]  7  8  8  8  8  8  8  8  8  8  8  9  9  9  9  9  9  9  9  9 10 10 10
##  [93] 10 10 10 10 10 10 10 11 11 11 11 11 11 11 11 11 11 12 12 12 12 12 12
## [116] 12 12 12 12 12 13 13 13 13 13 13 13 13 13 13 14 14 14 14 14 14 14 14
## [139] 14 14 15 15 15 15 15 15 15 15 15 15 16 16 16 16 16 16 16 16 16 16 17
```

```
## [162] 17 17 17 17 17 17 17 17 17 18 18 18 18 18 18 18 18 18 19 19 19 19
## [185] 19 19 19 19 19 19 20 20 20 20 20 20 20 20 20 20
```

That looks better. Let's store this in a tibble named `new`.

```
new <- seq(1:20) %>%
        rep(each=10) %>%
        as_tibble()
```

Sampling times

Next, we'll create a vector of sampling times.

```
c(0,1,2,3,4,6,9,12,18,24)
```

```
##  [1]  0  1  2  3  4  6  9 12 18 24
```

Use the `rep()` function to match `id` and save it as the variable `time`.

```
time <-
  c(0,1,2,3,4,6,9,12,18,24) %>%
  rep(20)
```

We can add this to the data set with `mutate()` and change the name of `value` to `id` with `rename()`.

```
new <- new %>%
  rename(id = value) %>%
  mutate(time = time)

new
```

```
## # A tibble: 200 x 2
##       id  time
##    <int> <dbl>
##  1     1     0
##  2     1     1
##  3     1     2
##  4     1     3
##  5     1     4
##  6     1     6
##  7     1     9
##  8     1    12
##  9     1    18
## 10     1    24
## # ... with 190 more rows
```

This is a good start but how often are sampling times this precise? We can add some variability and create a new variable. Sample from the the normal distribution with a mean of 1 and a small standard deviation, multiply by nominal time, then round the result.

```
timeR <- time %>%
  '*'(rnorm(200,1,0.05)) %>%
  round(2)

new <- new %>%
      rename(nomTime = time) %>%
      mutate(time = timeR)
```

```
new
```

```
## # A tibble: 200 x 3
##       id nomTime  time
##    <int>   <dbl> <dbl>
##  1     1       0  0
##  2     1       1  1.05
##  3     1       2  1.93
##  4     1       3  2.91
##  5     1       4  4.31
##  6     1       6  6.1
##  7     1       9  8.77
##  8     1      12 12.3
##  9     1      18 19.0
## 10     1      24 26
## # ... with 190 more rows
```

Simulating binary or categorical variables with equal probability of being chosen

```
sample(c(0,1), length(unique(new$id)), replace = T) %>% rep(each=10)
```

```
##   [1] 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1
##  [36] 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1
##  [71] 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
## [106] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
## [141] 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0
## [176] 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

```
sample(c(1,2,3,4), length(unique(new$id)), replace = T) %>% rep(each=10)
```

```
##   [1] 1 1 1 1 1 1 1 1 1 1 3 3 3 3 3 3 3 3 3 3 2 2 2 2 2 2 2 2 2 2 3 3 3 3 3
##  [36] 3 3 3 3 3 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 3 3 3 3 3 3 3 3 3
##  [71] 1 1 1 1 1 1 1 1 1 1 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 4 4 4 4 4
## [106] 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 1 1 1 1 1 1 1 1 1 4 4 4 4 4 4 4 4 4 4 4
## [141] 2 2 2 2 2 2 2 2 2 2 1 1 1 1 1 1 1 1 1 1 4 4 4 4 4 4 4 4 4 4 3 3 3 3 3
## [176] 3 3 3 3 3 2 2 2 2 2 2 2 2 2 2 3 3 3 3 3 3 3 3 3 3
```

Use `set.seed()` for reproducible results.

```
set.seed(1907)
sex <- sample(c(0,1), length(unique(new$id)), replace = T) %>% rep(each=10)
```

```
set.seed(1907)
race <- sample(c(1,2,3,4), length(unique(new$id)), replace = T) %>% rep(each=10)

new <- new %>%
    mutate(sex = sex,
           race)

new
```

```
## # A tibble: 200 x 5
```

```
##       id nomTime  time   sex  race
##    <int>   <dbl> <dbl> <dbl> <dbl>
##  1     1       0  0        0     1
##  2     1       1  1.05     0     1
##  3     1       2  1.93     0     1
##  4     1       3  2.91     0     1
##  5     1       4  4.31     0     1
##  6     1       6  6.1      0     1
##  7     1       9  8.77     0     1
##  8     1      12 12.3      0     1
##  9     1      18 19.0      0     1
## 10     1      24 26        0     1
## # ... with 190 more rows
```

Note the argument in 'mutate()' to keep the same variable name.

Simulate a uniform distribution of ages

```
set.seed(1907)
age <- runif(length(unique(new$id)), 18, 65) %>% rep(each=10) %>% floor()

age
```

```
##   [1] 18 18 18 18 18 18 18 18 18 18 55 55 55 55 55 55 55 55 55 55 26 26 26
##  [24] 26 26 26 26 26 26 26 45 45 45 45 45 45 45 45 45 45 45 45 45 45 45 45
##  [47] 45 45 45 45 43 43 43 43 43 43 43 43 43 43 45 45 45 45 45 45 45 45 45
##  [70] 45 26 26 26 26 26 26 26 26 26 26 41 41 41 41 41 41 41 41 41 41 44 44
##  [93] 44 44 44 44 44 44 44 44 20 20 20 20 20 20 20 20 20 20 28 28 28 28 28
## [116] 28 28 28 28 28 37 37 37 37 37 37 37 37 37 37 51 51 51 51 51 51 51 51
## [139] 51 51 45 45 45 45 45 45 45 45 45 45 29 29 29 29 29 29 29 29 29 29 18
## [162] 18 18 18 18 18 18 18 18 18 22 22 22 22 22 22 22 22 22 22 43 43 43 43
## [185] 43 43 43 43 43 43 54 54 54 54 54 54 54 54 54 54
```

```
new <- new %>% mutate(age)

new
```

```
## # A tibble: 200 x 6
##       id nomTime  time   sex  race   age
##    <int>   <dbl> <dbl> <dbl> <dbl> <dbl>
##  1     1       0  0        0     1    18
##  2     1       1  1.05     0     1    18
##  3     1       2  1.93     0     1    18
##  4     1       3  2.91     0     1    18
##  5     1       4  4.31     0     1    18
##  6     1       6  6.1      0     1    18
##  7     1       9  8.77     0     1    18
##  8     1      12 12.3      0     1    18
##  9     1      18 19.0      0     1    18
## 10     1      24 26        0     1    18
## # ... with 190 more rows
```

Check the documentation for `round()` to look at the `floor()` function and others related to it.

Finding first and last observations for a subject in longitudial data

```
new <- new %>%
  mutate(fid = as.numeric(!duplicated(new$id)),
         lid = as.numeric(!duplicated(new$id, fromLast = T)))

new
```

```
## # A tibble: 200 x 8
##       id nomTime  time   sex  race   age   fid   lid
##    <int>   <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>
##  1     1       0  0        0     1    18     1     0
##  2     1       1  1.05     0     1    18     0     0
##  3     1       2  1.93     0     1    18     0     0
##  4     1       3  2.91     0     1    18     0     0
##  5     1       4  4.31     0     1    18     0     0
##  6     1       6  6.1      0     1    18     0     0
##  7     1       9  8.77     0     1    18     0     0
##  8     1      12 12.3      0     1    18     0     0
##  9     1      18 19.0      0     1    18     0     0
## 10     1      24 26        0     1    18     0     1
## # ... with 190 more rows
```

Exercise: Summarize the new dataset.

Session information

```
sessionInfo()
```

```
## R version 3.5.1 (2018-07-02)
## Platform: x86_64-w64-mingw32/x64 (64-bit)
## Running under: Windows 10 x64 (build 17134)
##
## Matrix products: default
##
## locale:
## [1] LC_COLLATE=English_United States.1252
## [2] LC_CTYPE=English_United States.1252
## [3] LC_MONETARY=English_United States.1252
## [4] LC_NUMERIC=C
## [5] LC_TIME=English_United States.1252
##
## attached base packages:
## [1] stats     graphics  grDevices utils     datasets  methods   base
##
## other attached packages:
## [1] bindrcpp_0.2.2 dplyr_0.7.6
##
## loaded via a namespace (and not attached):
##  [1] Rcpp_0.12.17     knitr_1.20       bindr_0.1.1      magrittr_1.5
##  [5] tidyselect_0.2.4 R6_2.2.2         rlang_0.2.1      stringr_1.3.1
##  [9] tools_3.5.1      utf8_1.1.4       cli_1.0.0        htmltools_0.3.6
```

```
## [13] yaml_2.2.0       assertthat_0.2.0 rprojroot_1.3-2  digest_0.6.15
## [17] tibble_1.4.2     crayon_1.3.4     purrr_0.2.5      glue_1.2.0
## [21] evaluate_0.10.1  rmarkdown_1.10   stringi_1.2.3    compiler_3.5.1
## [25] pillar_1.2.3     backports_1.1.2  pkgconfig_2.0.1
```