TidyVerse - mutate_at(), mutate_if(), mutate_all()

Ryan Gordon

May 5, 2019

Contents

Description

The "mutate" function is found within the dplyr package. In this class, we have explored this function at length, but we did not go into too much depth with variants of this function: mutate_at(), mutate_if() and mutate_all(). The functionality of these variable can be seen below:

(1) mutate_all() - Affects every column variable in a particular table.

(2) mutate_if() - Affects onlt the columns that satisfy the if statement. For example, if it was necessary to only edit the numeric columns, the first argument in the function would be "is.numeric". This will be explored in the examples section.

(3) mutate_at() - Affects only the columns that were explicitly defined in the function.

Example

To show an example of how these mutate variants can be used, I decided to use a dataset provided by Kaggle that describes Chicago's Beach Water Quality (https://www.kaggle.com/chicago/chicago-beach-swim,-weather,-lab-data/downloads/chicago-beach-swim,-weather,-lab-data.zip/21#beach-water-quality-automated-sensors.csv). The only library that I needed to load was the tidyverse package, as shown below.

```
library(tidyverse)
```

Dataset and Initial Data Handling

Initially, I imported the csv file into my Github, and uploaded the csv from there. I then changed it to a tbl table, which is required by the mutate function.

```
water.quality.df <- read.csv(url("https://raw.githubusercontent.com/rg563/SPRING2019TIDYVERSE/master/bea
water.quality.tbl <- as_tibble(water.quality.df)
head(water.quality.tbl)
```

```
# A tibble: 6 x 10
  Beach.Name Measurement.Tim~ Water.Temperatu~ Turbidity Transducer.Depth
  <fct>      <fct>                       <dbl>     <dbl>            <dbl>
1 Montrose ~ 2013-08-30T08:0~             20.3      1.18            0.891
2 Osterman ~ 2013-08-31T23:0~             21.5      3.51            1.54
3 Ohio Stre~ 2013-09-03T03:0~             21.9      4.97            1.04
4 Calumet B~ 2013-09-03T16:0~             23.2      3.63            1.20
5 63rd Stre~ 2013-09-18T10:0~             18.9      7.56            1.52
6 Rainbow B~ 2014-05-21T13:0~             27.1      0.74            0.104
# ... with 5 more variables: Wave.Height <dbl>, Wave.Period <dbl>,
#   Battery.Life <dbl>, Measurement.Timestamp.Label <fct>,
#   Measurement.ID <fct>
```

The next section provides examples of how to use each of the mutate functions.

Function Examples

mutate_all()

In this example, let's simply divide all of the columns of type double by 100, which happens to be columns 3-8. First, I created a simple function, which divides all x by 100. Next, I used the mutate_all function to call columns 3-8 from the table and called the function to manipulate these columns

```
divide.by.100 <- function(x, na.rm=FALSE) (x/100)
water.quality.tbl.div.100 <- mutate_all(water.quality.tbl[,3:8], divide.by.100)
head(water.quality.tbl)
```

```
# A tibble: 6 x 10
  Beach.Name Measurement.Tim~ Water.Temperatu~ Turbidity Transducer.Depth
  <fct>      <fct>                       <dbl>     <dbl>            <dbl>
1 Montrose ~ 2013-08-30T08:0~             20.3      1.18            0.891
2 Osterman ~ 2013-08-31T23:0~             21.5      3.51            1.54
3 Ohio Stre~ 2013-09-03T03:0~             21.9      4.97            1.04
4 Calumet B~ 2013-09-03T16:0~             23.2      3.63            1.20
5 63rd Stre~ 2013-09-18T10:0~             18.9      7.56            1.52
6 Rainbow B~ 2014-05-21T13:0~             27.1      0.74            0.104
# ... with 5 more variables: Wave.Height <dbl>, Wave.Period <dbl>,
#   Battery.Life <dbl>, Measurement.Timestamp.Label <fct>,
#   Measurement.ID <fct>
```

If we compare the values of this table to the ones in the previous section, we can see that all of these columns are divided by 100.

mutate_if()

For this example, let's perform the same manipulation as the previous section, but instead we will use the mutate_if() function. The first argument of the mutate_if() function will be the type of column we want to manipulate. In this example, we use "is.double" because we only want to manipulate these columns. The next argument is the function. This can be seen below:

```
water.quality.tbl %>% mutate_if(is.double, divide.by.100, na.rm = TRUE)
```

```
# A tibble: 36,411 x 10
   Beach.Name Measurement.Tim~ Water.Temperatu~ Turbidity Transducer.Depth
   <fct>      <fct>                       <dbl>     <dbl>            <dbl>
 1 Montrose ~ 2013-08-30T08:0~            0.203    0.0118          0.00891
 2 Osterman ~ 2013-08-31T23:0~            0.215    0.0351          0.0154
 3 Ohio Stre~ 2013-09-03T03:0~            0.219    0.0497          0.0104
 4 Calumet B~ 2013-09-03T16:0~            0.232    0.0363          0.0120
 5 63rd Stre~ 2013-09-18T10:0~            0.189    0.0756          0.0152
 6 Rainbow B~ 2014-05-21T13:0~            0.271    0.0074          0.00104
 7 Calumet B~ 2014-05-28T12:0~            0.162    0.0126          0.0151
 8 Montrose ~ 2014-05-28T12:0~            0.144    0.0336          0.0139
 9 Montrose ~ 2014-05-28T13:0~            0.145    0.0272          0.0140
10 Calumet B~ 2014-05-28T13:0~            0.163    0.0128          0.0152
# ... with 36,401 more rows, and 5 more variables: Wave.Height <dbl>,
#   Wave.Period <dbl>, Battery.Life <dbl>,
#   Measurement.Timestamp.Label <fct>, Measurement.ID <fct>
```

As we can see from the results, the only variables that were edited were the columns that were double types.

mutate_at()

The mutate_at() function allows us to select specific columns and make manipulations. For example, say we wanted to convert the water temperature from Celsius to Farenheit. We can create a function to accomplish this (convert.to.farenheit), and then call this function in the mutate_at() function, but only have it operate on the temperature column.

```
convert.to.farenheit <- function(x, na.rm=FALSE) (x*9/5+32)
water.quality.tbl %>% mutate_at(c("Water.Temperature"), convert.to.farenheit)
```

```
# A tibble: 36,411 x 10
   Beach.Name Measurement.Tim~ Water.Temperatu~ Turbidity Transducer.Depth
   <fct>      <fct>                       <dbl>     <dbl>            <dbl>
 1 Montrose ~ 2013-08-30T08:0~             68.5      1.18            0.891
 2 Osterman ~ 2013-08-31T23:0~             70.7      3.51            1.54
 3 Ohio Stre~ 2013-09-03T03:0~             71.4      4.97            1.04
 4 Calumet B~ 2013-09-03T16:0~             73.8      3.63            1.20
 5 63rd Stre~ 2013-09-18T10:0~             66.0      7.56            1.52
 6 Rainbow B~ 2014-05-21T13:0~             80.8      0.74            0.104
 7 Calumet B~ 2014-05-28T12:0~             61.2      1.26            1.51
 8 Montrose ~ 2014-05-28T12:0~             57.9      3.36            1.39
 9 Montrose ~ 2014-05-28T13:0~             58.1      2.72            1.40
10 Calumet B~ 2014-05-28T13:0~             61.3      1.28            1.52
# ... with 36,401 more rows, and 5 more variables: Wave.Height <dbl>,
#   Wave.Period <dbl>, Battery.Life <dbl>,
#   Measurement.Timestamp.Label <fct>, Measurement.ID <fct>
```

From the table, we can see the that the water temperature is now in degrees Farenheit.

Extended by Samriti and Vishal

Using dplyr::Transmute

Change Variables And Drop All Others. This function works like a combination of mutate and select: it may be used to modify values in a data frame, and then drops any column not explicitly specified

Usage transmute(.self, ...)

transmute_(.self, ..., .dots)

Example we wanted to convert the water temperature from Celsius to Farenheit in water.quality.tbl data frame

```
water.quality.tbl %>% transmute(Water.Temperature = Water.Temperature*9/5+32/100)
```

```
# A tibble: 36,411 x 1
   Water.Temperature
               <dbl>
 1              36.9
 2              39.0
 3              39.7
 4              42.1
 5              34.3
 6              49.1
 7              29.5
 8              26.2
 9              26.4
10              29.7
# ... with 36,401 more rows
```

Summarise()

Function:Create one or more scalar variables summarizing the variables of an existing tbl.

The scoped variants of summarise() make it easy to apply the same transformation to multiple variables. There are three variants.

summarise_all() affects every variable summarise_all(.tbl, .funs, . . .)

summarise_at() affects variables selected with a character vector or vars() summarise_if(.tbl, .predicate, .funs, . . .)

summarise_if() affects variables selected with a predicate function summarise_at(.tbl, .vars, .funs, . . . , .cols = NULL)

Description The _if() variants apply a predicate function (a function that returns TRUE or FALSE) to determine the relevant subset of columns. Here we apply mean() to the numeric columns:

```
water.quality.tbl %>%
  summarise_if(is.numeric, mean, na.rm = TRUE)
```

```
## # A tibble: 1 x 6
##   Water.Temperatu~ Turbidity Transducer.Depth Wave.Height Wave.Period
##              <dbl>     <dbl>            <dbl>       <dbl>       <dbl>
## 1             19.4      4.70             1.57      -1454.      -1450.
## # ... with 1 more variable: Battery.Life <dbl>
```