R_Tidyverse_Package

Norman Lo

1/10/2022

Tidyverse

The **tidyverse** is an opinionated collection of R packages designed for data science. **dplyr** and **purrr** are the two most popular packages in tidyverse for data analytics. In this section, we combine the use of dplyr and tidyr packages to focus on data cleaning and manipulation on **tbl** (similar to datafrmae) object. Even though the main stream for this process is processed with plyr, reshape2, and data.table packages, dplyr and tidyr has been gaining its popularity in the data science industry.

1 - dplyr Package

1.1 - tbl Object and data.frame

Note: if you want to use both dplyr and plyr, the plyr package should be imported first, then import the dplyr package. The reason is some of the object names are the same in the two packages and the last one being imported will be called.

```
# Importing the magritter package for pipe operation
library(magrittr)

# Importing the diamonds data set from ggplot2 library
data(diamonds, package='ggplot2')
dim(head(diamonds, n=4))
```

```
## [1]  4 10
```

```
# Using pipe(%>%) to pass on an object to another object
diamonds %>% head(4) %>% dim
```

```
## [1]  4 10
```

```
# The diamonds data set is saved as tbl, in fact, it's saved as tbl_df
class(diamonds)
```

```
## [1] "tbl_df"     "tbl"        "data.frame"
```

```
# If we are not using dplyr or tbl package, the data set will be displayed as regular dataframe.
head(diamonds)
```

```
##   carat       cut color clarity depth table price    x    y    z
## 1  0.23     Ideal     E     SI2  61.5    55   326 3.95 3.98 2.43
## 2  0.21   Premium     E     SI1  59.8    61   326 3.89 3.84 2.31
## 3  0.23      Good     E     VS1  56.9    65   327 4.05 4.07 2.31
## 4  0.29   Premium     I     VS2  62.4    58   334 4.20 4.23 2.63
## 5  0.31      Good     J     SI2  63.3    58   335 4.34 4.35 2.75
## 6  0.24 Very Good     J    VVS2  62.8    57   336 3.94 3.96 2.48
```

```
# After loading the dplyr package, the data set will be displayed as tbl object
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
```

```
head(diamonds)
```

```
## # A tibble: 6 x 10
##   carat cut       color clarity depth table price     x     y     z
##   <dbl> <ord>     <ord> <ord>   <dbl> <dbl> <int> <dbl> <dbl> <dbl>
## 1  0.23 Ideal     E     SI2      61.5    55   326  3.95  3.98  2.43
## 2  0.21 Premium   E     SI1      59.8    61   326  3.89  3.84  2.31
## 3  0.23 Good      E     VS1      56.9    65   327  4.05  4.07  2.31
## 4  0.29 Premium   I     VS2      62.4    58   334  4.2   4.23  2.63
## 5  0.31 Good      J     SI2      63.3    58   335  4.34  4.35  2.75
## 6  0.24 Very Good J     VVS2     62.8    57   336  3.94  3.96  2.48
```

```
# tbl displays the first 10 rows without the head() function
diamonds
```

```
## # A tibble: 53,940 x 10
##    carat cut       color clarity depth table price     x     y     z
##    <dbl> <ord>     <ord> <ord>   <dbl> <dbl> <int> <dbl> <dbl> <dbl>
##  1  0.23 Ideal     E     SI2      61.5    55   326  3.95  3.98  2.43
##  2  0.21 Premium   E     SI1      59.8    61   326  3.89  3.84  2.31
##  3  0.23 Good      E     VS1      56.9    65   327  4.05  4.07  2.31
##  4  0.29 Premium   I     VS2      62.4    58   334  4.2   4.23  2.63
##  5  0.31 Good      J     SI2      63.3    58   335  4.34  4.35  2.75
##  6  0.24 Very Good J     VVS2     62.8    57   336  3.94  3.96  2.48
##  7  0.24 Very Good I     VVS1     62.3    57   336  3.95  3.98  2.47
##  8  0.26 Very Good H     SI1      61.9    55   337  4.07  4.11  2.53
##  9  0.22 Fair      E     VS2      65.1    61   337  3.87  3.78  2.49
## 10  0.23 Very Good H     VS1      59.4    61   338  4     4.05  2.39
## # ... with 53,930 more rows
```

1.2 - Select

```
# select() function takes the first agrument data.frame, then the column names.
select(diamonds, carat, price)
```

```
## # A tibble: 53,940 x 2
##    carat price
##    <dbl> <int>
##  1  0.23   326
##  2  0.21   326
##  3  0.23   327
##  4  0.29   334
##  5  0.31   335
##  6  0.24   336
##  7  0.24   336
##  8  0.26   337
##  9  0.22   337
## 10  0.23   338
## # ... with 53,930 more rows
```

```
# Using a pipe(%>%) operator
diamonds %>% select(carat, price)
```

```
## # A tibble: 53,940 x 2
##    carat price
##    <dbl> <int>
##  1  0.23   326
##  2  0.21   326
##  3  0.23   327
##  4  0.29   334
##  5  0.31   335
##  6  0.24   336
##  7  0.24   336
##  8  0.26   337
##  9  0.22   337
## 10  0.23   338
## # ... with 53,930 more rows
```

```
# Also can use vector c() within the pipe operator
diamonds %>% select(c(carat, price))
```

```
## # A tibble: 53,940 x 2
##    carat price
##    <dbl> <int>
##  1  0.23   326
##  2  0.21   326
##  3  0.23   327
##  4  0.29   334
##  5  0.31   335
##  6  0.24   336
##  7  0.24   336
##  8  0.26   337
##  9  0.22   337
## 10  0.23   338
## # ... with 53,930 more rows
```

```
# We can also directly call out the column names with "select_" in standard evaluation version
diamonds %>% select_('carat', 'price')
```

```
## Warning: ‘select_()‘ was deprecated in dplyr 0.7.0.
## Please use ‘select()‘ instead.
```

```
## # A tibble: 53,940 x 2
##    carat price
##    <dbl> <int>
##  1  0.23   326
##  2  0.21   326
##  3  0.23   327
##  4  0.29   334
##  5  0.31   335
##  6  0.24   336
##  7  0.24   336
##  8  0.26   337
##  9  0.22   337
## 10  0.23   338
## # ... with 53,930 more rows
```

```
# If we saving the column names into a vector
theCols <- c('carat', 'price')
# We can use .dots to call out the column names in a vector
diamonds %>% select_(.dots=theCols)
```

```
## # A tibble: 53,940 x 2
##    carat price
##    <dbl> <int>
##  1  0.23   326
##  2  0.21   326
##  3  0.23   327
##  4  0.29   334
##  5  0.31   335
##  6  0.24   336
##  7  0.24   336
##  8  0.26   337
##  9  0.22   337
## 10  0.23   338
## # ... with 53,930 more rows
```

```
# An alternative for select_, is nesting select() with one_of()
diamonds %>% select(one_of('carat', 'price'))
```

```
## # A tibble: 53,940 x 2
##    carat price
##    <dbl> <int>
## 1  0.23   326
## 2  0.21   326
## 3  0.23   327
## 4  0.29   334
## 5  0.31   335
## 6  0.24   336
## 7  0.24   336
## 8  0.26   337
## 9  0.22   337
## 10  0.23   338
## # ... with 53,930 more rows
```

```
# In this case, we can pass in the vector of column names directly
diamonds %>% select(one_of(theCols))
```

```
## # A tibble: 53,940 x 2
##    carat price
##    <dbl> <int>
##  1  0.23   326
##  2  0.21   326
##  3  0.23   327
##  4  0.29   334
##  5  0.31   335
##  6  0.24   336
##  7  0.24   336
##  8  0.26   337
##  9  0.22   337
## 10  0.23   338
## # ... with 53,930 more rows
```

```
# We can also use a traditional R method
diamonds[,c('carat', 'price')]
```

```
## # A tibble: 53,940 x 2
##    carat price
##    <dbl> <int>
##  1  0.23   326
##  2  0.21   326
##  3  0.23   327
##  4  0.29   334
##  5  0.31   335
##  6  0.24   336
##  7  0.24   336
##  8  0.26   337
##  9  0.22   337
## 10  0.23   338
## # ... with 53,930 more rows
```

```
# We can also select using the column index
select(diamonds, 1, 7)
```

```
## # A tibble: 53,940 x 2
##    carat price
##    <dbl> <int>
##  1  0.23   326
##  2  0.21   326
##  3  0.23   327
##  4  0.29   334
##  5  0.31   335
##  6  0.24   336
##  7  0.24   336
##  8  0.26   337
##  9  0.22   337
## 10  0.23   338
## # ... with 53,930 more rows
```

```
diamonds %>% select(1, 7)
```

```
## # A tibble: 53,940 x 2
##    carat price
##    <dbl> <int>
##  1  0.23   326
##  2  0.21   326
##  3  0.23   327
##  4  0.29   334
##  5  0.31   335
##  6  0.24   336
##  7  0.24   336
##  8  0.26   337
##  9  0.22   337
## 10  0.23   338
## # ... with 53,930 more rows
```

1.3 - Partial Match

```
# We can find partial match from the data frame using start_with, ends_with, and contains
diamonds %>% select(starts_with('c'))
```

```
## # A tibble: 53,940 x 4
##    carat cut       color clarity
##    <dbl> <ord>     <ord> <ord>
##  1  0.23 Ideal     E     SI2
##  2  0.21 Premium   E     SI1
##  3  0.23 Good      E     VS1
##  4  0.29 Premium   I     VS2
##  5  0.31 Good      J     SI2
##  6  0.24 Very Good J     VVS2
##  7  0.24 Very Good I     VVS1
##  8  0.26 Very Good H     SI1
##  9  0.22 Fair      E     VS2
## 10  0.23 Very Good H     VS1
## # ... with 53,930 more rows
```

```
diamonds %>% select(ends_with('e'))
```

```
## # A tibble: 53,940 x 2
##    table price
##    <dbl> <int>
##  1    55   326
##  2    61   326
##  3    65   327
##  4    58   334
##  5    58   335
##  6    57   336
##  7    57   336
##  8    55   337
##  9    61   337
## 10    61   338
## # ... with 53,930 more rows
```

```
diamonds %>% select(contains('l'))
```

```
## # A tibble: 53,940 x 3
##    color clarity table
##    <ord> <ord>   <dbl>
##  1 E     SI2        55
##  2 E     SI1        61
##  3 E     VS1        65
##  4 I     VS2        58
##  5 J     SI2        58
##  6 J     VVS2       57
##  7 I     VVS1       57
##  8 H     SI1        55
##  9 E     VS2        61
## 10 H     VS1        61
## # ... with 53,930 more rows
```

```
# Using "matches" to find specific columns
# e.g. we can search for column names with "r" + "."(wildcard) + "t"
diamonds %>% select(matches('r.+t'))
```

```
## # A tibble: 53,940 x 2
##    carat clarity
##    <dbl> <ord>
##  1  0.23 SI2
##  2  0.21 SI1
##  3  0.23 VS1
##  4  0.29 VS2
##  5  0.31 SI2
##  6  0.24 VVS2
##  7  0.24 VVS1
##  8  0.26 SI1
##  9  0.22 VS2
## 10  0.23 VS1
## # ... with 53,930 more rows
```

```r
# We can also remove the columns by using '-'
diamonds %>% select(-carat, -price)
```

```
## # A tibble: 53,940 x 8
##    cut       color clarity depth table     x     y     z
##    <ord>     <ord> <ord>   <dbl> <dbl> <dbl> <dbl> <dbl>
##  1 Ideal     E     SI2      61.5    55  3.95  3.98  2.43
##  2 Premium   E     SI1      59.8    61  3.89  3.84  2.31
##  3 Good      E     VS1      56.9    65  4.05  4.07  2.31
##  4 Premium   I     VS2      62.4    58  4.2   4.23  2.63
##  5 Good      J     SI2      63.3    58  4.34  4.35  2.75
##  6 Very Good J     VVS2     62.8    57  3.94  3.96  2.48
##  7 Very Good I     VVS1     62.3    57  3.95  3.98  2.47
##  8 Very Good H     SI1      61.9    55  4.07  4.11  2.53
##  9 Fair      E     VS2      65.1    61  3.87  3.78  2.49
## 10 Very Good H     VS1      59.4    61  4     4.05  2.39
## # ... with 53,930 more rows
```

```r
diamonds %>% select(-c('carat', 'price'))
```

```
## # A tibble: 53,940 x 8
##    cut       color clarity depth table     x     y     z
##    <ord>     <ord> <ord>   <dbl> <dbl> <dbl> <dbl> <dbl>
##  1 Ideal     E     SI2      61.5    55  3.95  3.98  2.43
##  2 Premium   E     SI1      59.8    61  3.89  3.84  2.31
##  3 Good      E     VS1      56.9    65  4.05  4.07  2.31
##  4 Premium   I     VS2      62.4    58  4.2   4.23  2.63
##  5 Good      J     SI2      63.3    58  4.34  4.35  2.75
##  6 Very Good J     VVS2     62.8    57  3.94  3.96  2.48
##  7 Very Good I     VVS1     62.3    57  3.95  3.98  2.47
##  8 Very Good H     SI1      61.9    55  4.07  4.11  2.53
##  9 Fair      E     VS2      65.1    61  3.87  3.78  2.49
## 10 Very Good H     VS1      59.4    61  4     4.05  2.39
## # ... with 53,930 more rows
```

```r
diamonds %>% select(-1, -7)
```

```
## # A tibble: 53,940 x 8
##    cut       color clarity depth table     x     y     z
##    <ord>     <ord> <ord>   <dbl> <dbl> <dbl> <dbl> <dbl>
##  1 Ideal     E     SI2      61.5    55  3.95  3.98  2.43
##  2 Premium   E     SI1      59.8    61  3.89  3.84  2.31
##  3 Good      E     VS1      56.9    65  4.05  4.07  2.31
##  4 Premium   I     VS2      62.4    58  4.2   4.23  2.63
##  5 Good      J     SI2      63.3    58  4.34  4.35  2.75
##  6 Very Good J     VVS2     62.8    57  3.94  3.96  2.48
##  7 Very Good I     VVS1     62.3    57  3.95  3.98  2.47
##  8 Very Good H     SI1      61.9    55  4.07  4.11  2.53
##  9 Fair      E     VS2      65.1    61  3.87  3.78  2.49
## 10 Very Good H     VS1      59.4    61  4     4.05  2.39
## # ... with 53,930 more rows
```

```
diamonds %>% select(-c(1,7))
```

```
## # A tibble: 53,940 x 8
##    cut       color clarity depth table     x     y     z
##    <ord>     <ord> <ord>   <dbl> <dbl> <dbl> <dbl> <dbl>
##  1 Ideal     E     SI2      61.5    55  3.95  3.98  2.43
##  2 Premium   E     SI1      59.8    61  3.89  3.84  2.31
##  3 Good      E     VS1      56.9    65  4.05  4.07  2.31
##  4 Premium   I     VS2      62.4    58  4.2   4.23  2.63
##  5 Good      J     SI2      63.3    58  4.34  4.35  2.75
##  6 Very Good J     VVS2     62.8    57  3.94  3.96  2.48
##  7 Very Good I     VVS1     62.3    57  3.95  3.98  2.47
##  8 Very Good H     SI1      61.9    55  4.07  4.11  2.53
##  9 Fair      E     VS2      65.1    61  3.87  3.78  2.49
## 10 Very Good H     VS1      59.4    61  4     4.05  2.39
## # ... with 53,930 more rows
```

```
diamonds %>% select_(.dots=c('-carat', '-price'))
```

```
## # A tibble: 53,940 x 8
##    cut       color clarity depth table     x     y     z
##    <ord>     <ord> <ord>   <dbl> <dbl> <dbl> <dbl> <dbl>
##  1 Ideal     E     SI2      61.5    55  3.95  3.98  2.43
##  2 Premium   E     SI1      59.8    61  3.89  3.84  2.31
##  3 Good      E     VS1      56.9    65  4.05  4.07  2.31
##  4 Premium   I     VS2      62.4    58  4.2   4.23  2.63
##  5 Good      J     SI2      63.3    58  4.34  4.35  2.75
##  6 Very Good J     VVS2     62.8    57  3.94  3.96  2.48
##  7 Very Good I     VVS1     62.3    57  3.95  3.98  2.47
##  8 Very Good H     SI1      61.9    55  4.07  4.11  2.53
##  9 Fair      E     VS2      65.1    61  3.87  3.78  2.49
## 10 Very Good H     VS1      59.4    61  4     4.05  2.39
## # ... with 53,930 more rows
```

```
# Another way is to use one_of() to select columns
diamonds %>% select(-one_of('carat', 'price'))
```

```
## # A tibble: 53,940 x 8
##    cut       color clarity depth table     x     y     z
##    <ord>     <ord> <ord>   <dbl> <dbl> <dbl> <dbl> <dbl>
##  1 Ideal     E     SI2      61.5    55  3.95  3.98  2.43
##  2 Premium   E     SI1      59.8    61  3.89  3.84  2.31
##  3 Good      E     VS1      56.9    65  4.05  4.07  2.31
##  4 Premium   I     VS2      62.4    58  4.2   4.23  2.63
##  5 Good      J     SI2      63.3    58  4.34  4.35  2.75
##  6 Very Good J     VVS2     62.8    57  3.94  3.96  2.48
##  7 Very Good I     VVS1     62.3    57  3.95  3.98  2.47
##  8 Very Good H     SI1      61.9    55  4.07  4.11  2.53
##  9 Fair      E     VS2      65.1    61  3.87  3.78  2.49
## 10 Very Good H     VS1      59.4    61  4     4.05  2.39
## # ... with 53,930 more rows
```

1.4 - Filter

```r
# Using filter() function, we can sort out the specific rows from the data frame with a given condition
# e.g. sort out the rows that has value "Ideal" in the "cut" column
diamonds %>% filter(cut == 'Ideal')
```

```
## # A tibble: 21,551 x 10
##    carat cut   color clarity depth table price     x     y     z
##    <dbl> <ord> <ord> <ord>   <dbl> <dbl> <int> <dbl> <dbl> <dbl>
##  1  0.23 Ideal E     SI2      61.5    55   326  3.95  3.98  2.43
##  2  0.23 Ideal J     VS1      62.8    56   340  3.93  3.9   2.46
##  3  0.31 Ideal J     SI2      62.2    54   344  4.35  4.37  2.71
##  4  0.3  Ideal I     SI2      62      54   348  4.31  4.34  2.68
##  5  0.33 Ideal I     SI2      61.8    55   403  4.49  4.51  2.78
##  6  0.33 Ideal I     SI2      61.2    56   403  4.49  4.5   2.75
##  7  0.33 Ideal J     SI1      61.1    56   403  4.49  4.55  2.76
##  8  0.23 Ideal G     VS1      61.9    54   404  3.93  3.95  2.44
##  9  0.32 Ideal I     SI1      60.9    55   404  4.45  4.48  2.72
## 10  0.3  Ideal I     SI2      61      59   405  4.3   4.33  2.63
## # ... with 21,541 more rows
```

```r
# Using the traditional R syntex,
diamonds[diamonds$cut == 'Ideal',]
```

```
## # A tibble: 21,551 x 10
##    carat cut   color clarity depth table price     x     y     z
##    <dbl> <ord> <ord> <ord>   <dbl> <dbl> <int> <dbl> <dbl> <dbl>
##  1  0.23 Ideal E     SI2      61.5    55   326  3.95  3.98  2.43
##  2  0.23 Ideal J     VS1      62.8    56   340  3.93  3.9   2.46
##  3  0.31 Ideal J     SI2      62.2    54   344  4.35  4.37  2.71
##  4  0.3  Ideal I     SI2      62      54   348  4.31  4.34  2.68
##  5  0.33 Ideal I     SI2      61.8    55   403  4.49  4.51  2.78
##  6  0.33 Ideal I     SI2      61.2    56   403  4.49  4.5   2.75
##  7  0.33 Ideal J     SI1      61.1    56   403  4.49  4.55  2.76
##  8  0.23 Ideal G     VS1      61.9    54   404  3.93  3.95  2.44
##  9  0.32 Ideal I     SI1      60.9    55   404  4.45  4.48  2.72
## 10  0.3  Ideal I     SI2      61      59   405  4.3   4.33  2.63
## # ... with 21,541 more rows
```

```r
# We can use %in% to select the rows with one of the search values in a specific column
# e.g. sort out the rows that has either value "Ideal" or "Good" in the cut column
diamonds %>% filter(cut %in% c('Ideal', 'Good'))
```

```
## # A tibble: 26,457 x 10
##    carat cut   color clarity depth table price     x     y     z
##    <dbl> <ord> <ord> <ord>   <dbl> <dbl> <int> <dbl> <dbl> <dbl>
##  1  0.23 Ideal E     SI2      61.5    55   326  3.95  3.98  2.43
##  2  0.23 Good  E     VS1      56.9    65   327  4.05  4.07  2.31
##  3  0.31 Good  J     SI2      63.3    58   335  4.34  4.35  2.75
##  4  0.3  Good  J     SI1      64      55   339  4.25  4.28  2.73
##  5  0.23 Ideal J     VS1      62.8    56   340  3.93  3.9   2.46
##  6  0.31 Ideal J     SI2      62.2    54    344  4.35  4.37  2.71
##  7  0.3  Ideal I     SI2      62      54    348  4.31  4.34  2.68
##  8  0.3  Good  J     SI1      63.4    54    351  4.23  4.29  2.7
##  9  0.3  Good  J     SI1      63.8    56    351  4.23  4.26  2.71
## 10  0.3  Good  I     SI2      63.3    56    351  4.26  4.3   2.71
## # ... with 26,447 more rows
```

```
# We can use any R standard logical operators with filter() function
diamonds %>% filter(price >= 1000)
```

```
## # A tibble: 39,441 x 10
##    carat cut       color clarity depth table price     x     y     z
##    <dbl> <ord>     <ord> <ord>   <dbl> <dbl> <int> <dbl> <dbl> <dbl>
##  1  0.7  Ideal     E     SI1      62.5    57  2757  5.7   5.72  3.57
##  2  0.86 Fair      E     SI2      55.1    69  2757  6.45  6.33  3.52
##  3  0.7  Ideal     G     VS2      61.6    56  2757  5.7   5.67  3.5
##  4  0.71 Very Good E     VS2      62.4    57  2759  5.68  5.73  3.56
##  5  0.78 Very Good G     SI2      63.8    56  2759  5.81  5.85  3.72
##  6  0.7  Good      E     VS2      57.5    58  2759  5.85  5.9   3.38
##  7  0.7  Good      F     VS1      59.4    62  2759  5.71  5.76  3.4
##  8  0.96 Fair      F     SI2      66.3    62  2759  6.27  5.95  4.07
##  9  0.73 Very Good E     SI1      61.6    59  2760  5.77  5.78  3.56
## 10  0.8  Premium   H     SI1      61.5    58  2760  5.97  5.93  3.66
## # ... with 39,431 more rows
```

```
diamonds %>% filter(price != 1000)
```

```
## # A tibble: 53,915 x 10
##    carat cut       color clarity depth table price     x     y     z
##    <dbl> <ord>     <ord> <ord>   <dbl> <dbl> <int> <dbl> <dbl> <dbl>
##  1  0.23 Ideal     E     SI2      61.5    55   326  3.95  3.98  2.43
##  2  0.21 Premium   E     SI1      59.8    61   326  3.89  3.84  2.31
##  3  0.23 Good      E     VS1      56.9    65   327  4.05  4.07  2.31
##  4  0.29 Premium   I     VS2      62.4    58   334  4.2   4.23  2.63
##  5  0.31 Good      J     SI2      63.3    58   335  4.34  4.35  2.75
##  6  0.24 Very Good J     VVS2     62.8    57   336  3.94  3.96  2.48
##  7  0.24 Very Good I     VVS1     62.3    57   336  3.95  3.98  2.47
##  8  0.26 Very Good H     SI1      61.9    55   337  4.07  4.11  2.53
##  9  0.22 Fair      E     VS2      65.1    61   337  3.87  3.78  2.49
## 10  0.23 Very Good H     VS1      59.4    61   338  4     4.05  2.39
## # ... with 53,905 more rows
```

```
# Filtering with condition 1 AND condition 2
diamonds %>% filter(carat>2, price<14000)
```

```
## # A tibble: 644 x 10
##    carat cut     color clarity depth table price     x     y     z
##    <dbl> <ord>   <ord> <ord>   <dbl> <dbl> <int> <dbl> <dbl> <dbl>
##  1  2.06 Premium J     I1       61.2    58  5203  8.1   8.07  4.95
##  2  2.14 Fair    J     I1       69.4    57  5405  7.74  7.7   5.36
##  3  2.15 Fair    J     I1       65.5    57  5430  8.01  7.95  5.23
##  4  2.22 Fair    J     I1       66.7    56  5607  8.04  8.02  5.36
##  5  2.01 Fair    I     I1       67.4    58  5696  7.71  7.64  5.17
##  6  2.01 Fair    I     I1       55.9    64  5696  8.48  8.39  4.71
##  7  2.27 Fair    J     I1       67.6    55  5733  8.05  8     5.43
##  8  2.03 Fair    H     I1       64.4    59  6002  7.91  7.85  5.07
##  9  2.03 Fair    H     I1       66.6    57  6002  7.81  7.75  5.19
## 10  2.06 Good    H     I1       64.3    58  6091  8.03  7.99  5.15
## # ... with 634 more rows
```

```
diamonds %>% filter(carat>2 & price<14000)
```

```
## # A tibble: 644 x 10
##    carat cut     color clarity depth table price     x     y     z
##    <dbl> <ord>   <ord> <ord>   <dbl> <dbl> <int> <dbl> <dbl> <dbl>
##  1  2.06 Premium J     I1       61.2    58  5203  8.1   8.07  4.95
##  2  2.14 Fair    J     I1       69.4    57  5405  7.74  7.7   5.36
##  3  2.15 Fair    J     I1       65.5    57  5430  8.01  7.95  5.23
##  4  2.22 Fair    J     I1       66.7    56  5607  8.04  8.02  5.36
##  5  2.01 Fair    I     I1       67.4    58  5696  7.71  7.64  5.17
##  6  2.01 Fair    I     I1       55.9    64  5696  8.48  8.39  4.71
##  7  2.27 Fair    J     I1       67.6    55  5733  8.05  8     5.43
##  8  2.03 Fair    H     I1       64.4    59  6002  7.91  7.85  5.07
##  9  2.03 Fair    H     I1       66.6    57  6002  7.81  7.75  5.19
## 10  2.06 Good    H     I1       64.3    58  6091  8.03  7.99  5.15
## # ... with 634 more rows
```

```
# Filtering with condition 1 OR condition 2
diamonds %>% filter(carat<1 | carat>5)
```

```
## # A tibble: 34,881 x 10
##    carat cut       color clarity depth table price     x     y     z
##    <dbl> <ord>     <ord> <ord>   <dbl> <dbl> <int> <dbl> <dbl> <dbl>
##  1  0.23 Ideal     E     SI2      61.5    55   326  3.95  3.98  2.43
##  2  0.21 Premium   E     SI1      59.8    61   326  3.89  3.84  2.31
##  3  0.23 Good      E     VS1      56.9    65   327  4.05  4.07  2.31
##  4  0.29 Premium   I     VS2      62.4    58   334  4.2   4.23  2.63
##  5  0.31 Good      J     SI2      63.3    58   335  4.34  4.35  2.75
##  6  0.24 Very Good J     VVS2     62.8    57   336  3.94  3.96  2.48
##  7  0.24 Very Good I     VVS1     62.3    57   336  3.95  3.98  2.47
##  8  0.26 Very Good H     SI1      61.9    55   337  4.07  4.11  2.53
##  9  0.22 Fair      E     VS2      65.1    61   337  3.87  3.78  2.49
## 10  0.23 Very Good H     VS1      59.4    61   338  4     4.05  2.39
## # ... with 34,871 more rows
```

```
# When using filter_() function, we need to pass in the condition in string
diamonds %>% filter_("cut == 'Ideal'")
```

```
## Warning: ‘filter_()‘ was deprecated in dplyr 0.7.0.
## Please use ‘filter()‘ instead.
## See vignette(’programming’) for more help
```

```
## # A tibble: 21,551 x 10
##    carat cut   color clarity depth table price     x     y     z
##    <dbl> <ord> <ord> <ord>   <dbl> <dbl> <int> <dbl> <dbl> <dbl>
##  1  0.23 Ideal E     SI2      61.5    55   326  3.95  3.98  2.43
##  2  0.23 Ideal J     VS1      62.8    56   340  3.93  3.9   2.46
##  3  0.31 Ideal J     SI2      62.2    54   344  4.35  4.37  2.71
##  4  0.3  Ideal I     SI2      62      54   348  4.31  4.34  2.68
##  5  0.33 Ideal I     SI2      61.8    55   403  4.49  4.51  2.78
##  6  0.33 Ideal I     SI2      61.2    56   403  4.49  4.5   2.75
##  7  0.33 Ideal J     SI1      61.1    56   403  4.49  4.55  2.76
##  8  0.23 Ideal G     VS1      61.9    54   404  3.93  3.95  2.44
##  9  0.32 Ideal I     SI1      60.9    55   404  4.45  4.48  2.72
## 10  0.3  Ideal I     SI2      61      59   405  4.3   4.33  2.63
## # ... with 21,541 more rows
```

```
# We can also use "~" to replace the outter quote ""
diamonds %>% filter_(~cut == 'Ideal')
```

```
## # A tibble: 21,551 x 10
##    carat cut   color clarity depth table price     x     y     z
##    <dbl> <ord> <ord> <ord>   <dbl> <dbl> <int> <dbl> <dbl> <dbl>
##  1  0.23 Ideal E     SI2      61.5    55   326  3.95  3.98  2.43
##  2  0.23 Ideal J     VS1      62.8    56   340  3.93  3.9   2.46
##  3  0.31 Ideal J     SI2      62.2    54   344  4.35  4.37  2.71
##  4  0.3  Ideal I     SI2      62      54   348  4.31  4.34  2.68
##  5  0.33 Ideal I     SI2      61.8    55   403  4.49  4.51  2.78
##  6  0.33 Ideal I     SI2      61.2    56   403  4.49  4.5   2.75
##  7  0.33 Ideal J     SI1      61.1    56   403  4.49  4.55  2.76
##  8  0.23 Ideal G     VS1      61.9    54   404  3.93  3.95  2.44
##  9  0.32 Ideal I     SI1      60.9    55   404  4.45  4.48  2.72
## 10  0.3  Ideal I     SI2      61      59   405  4.3   4.33  2.63
## # ... with 21,541 more rows
```

```
# The advantage for using "~" instead of the quote "" is that variable can be used in filter_()
# e.g.
theCut <- 'Ideal'
diamonds %>% filter_(~cut == theCut)
```

```
## # A tibble: 21,551 x 10
##    carat cut   color clarity depth table price     x     y     z
##    <dbl> <ord> <ord> <ord>   <dbl> <dbl> <int> <dbl> <dbl> <dbl>
##  1  0.23 Ideal E     SI2      61.5    55   326  3.95  3.98  2.43
##  2  0.23 Ideal J     VS1      62.8    56   340  3.93  3.9   2.46
##  3  0.31 Ideal J     SI2      62.2    54   344  4.35  4.37  2.71
##  4  0.3  Ideal I     SI2      62      54   348  4.31  4.34  2.68
##  5  0.33 Ideal I     SI2      61.8    55   403  4.49  4.51  2.78
##  6  0.33 Ideal I     SI2      61.2    56   403  4.49  4.5   2.75
##  7  0.33 Ideal J     SI1      61.1    56   403  4.49  4.55  2.76
##  8  0.23 Ideal G     VS1      61.9    54   404  3.93  3.95  2.44
##  9  0.32 Ideal I     SI1      60.9    55   404  4.45  4.48  2.72
## 10  0.3  Ideal I     SI2      61      59   405  4.3   4.33  2.63
## # ... with 21,541 more rows
```

```r
# When both the column name and row value are stored into variables,
# we can use sprintf() function to combine in filter_()
# Note: it's not recommended with more complex conditions
# e.g.
theCol <- 'cut'
theCut <- 'Ideal'
diamonds %>% filter_(sprintf("%s == '%s'", theCol, theCut))
```

```
## # A tibble: 21,551 x 10
##    carat cut   color clarity depth table price     x     y     z
##    <dbl> <ord> <ord> <ord>   <dbl> <dbl> <int> <dbl> <dbl> <dbl>
##  1  0.23 Ideal E     SI2      61.5    55   326  3.95  3.98  2.43
##  2  0.23 Ideal J     VS1      62.8    56   340  3.93  3.9   2.46
##  3  0.31 Ideal J     SI2      62.2    54   344  4.35  4.37  2.71
##  4  0.3  Ideal I     SI2      62      54   348  4.31  4.34  2.68
##  5  0.33 Ideal I     SI2      61.8    55   403  4.49  4.51  2.78
##  6  0.33 Ideal I     SI2      61.2    56   403  4.49  4.5   2.75
##  7  0.33 Ideal J     SI1      61.1    56   403  4.49  4.55  2.76
##  8  0.23 Ideal G     VS1      61.9    54   404  3.93  3.95  2.44
##  9  0.32 Ideal I     SI1      60.9    55   404  4.45  4.48  2.72
## 10  0.3  Ideal I     SI2      61      59   405  4.3   4.33  2.63
## # ... with 21,541 more rows
```

```r
# A preferred method is using the interp() function in the lazyeval pacakge
library(lazyeval)

# Define the variables into an formula
# as.name(theCol) -> cut
# theCut -> 'Ideal'
interp(~ a == b, a=as.name(theCol), b=theCut)
```

```
## ~cut == "Ideal"
```

```r
# Put the formula into the filter_() function
diamonds %>% filter_(interp(~ a == b, a=as.name(theCol), b=theCut))
```

```
## # A tibble: 21,551 x 10
##    carat cut   color clarity depth table price     x     y     z
##    <dbl> <ord> <ord> <ord>   <dbl> <dbl> <int> <dbl> <dbl> <dbl>
##  1  0.23 Ideal E     SI2      61.5    55   326  3.95  3.98  2.43
##  2  0.23 Ideal J     VS1      62.8    56   340  3.93  3.9   2.46
##  3  0.31 Ideal J     SI2      62.2    54   344  4.35  4.37  2.71
##  4  0.3  Ideal I     SI2      62      54   348  4.31  4.34  2.68
##  5  0.33 Ideal I     SI2      61.8    55   403  4.49  4.51  2.78
##  6  0.33 Ideal I     SI2      61.2    56   403  4.49  4.5   2.75
##  7  0.33 Ideal J     SI1      61.1    56   403  4.49  4.55  2.76
##  8  0.23 Ideal G     VS1      61.9    54   404  3.93  3.95  2.44
##  9  0.32 Ideal I     SI1      60.9    55   404  4.45  4.48  2.72
## 10  0.3  Ideal I     SI2      61      59   405  4.3   4.33  2.63
## # ... with 21,541 more rows
```

```r
# If using dplyr 0.6.0 version, we can combine the use of filter() function and UQ() function from rlan
# install.packages("rlang")
library(rlang)
```

```
##
## Attaching package: 'rlang'

## The following objects are masked from 'package:lazyeval':
##
##     as_name, call_modify, call_standardise, expr_label, expr_text,
##     f_env, f_env<-, f_label, f_lhs, f_lhs<-, f_rhs, f_rhs<-, f_text,
##     is_atomic, is_call, is_formula, is_lang, is_pairlist, missing_arg

## The following object is masked from 'package:magrittr':
##
##     set_names
```

```r
diamonds %>% filter(UQ(as.name(theCol)) == theCut)
```

```
## # A tibble: 21,551 x 10
##    carat cut   color clarity depth table price     x     y     z
##    <dbl> <ord> <ord> <ord>   <dbl> <dbl> <int> <dbl> <dbl> <dbl>
##  1  0.23 Ideal E     SI2      61.5    55   326  3.95  3.98  2.43
##  2  0.23 Ideal J     VS1      62.8    56   340  3.93  3.9   2.46
##  3  0.31 Ideal J     SI2      62.2    54   344  4.35  4.37  2.71
##  4  0.3  Ideal I     SI2      62      54   348  4.31  4.34  2.68
##  5  0.33 Ideal I     SI2      61.8    55   403  4.49  4.51  2.78
##  6  0.33 Ideal I     SI2      61.2    56   403  4.49  4.5   2.75
##  7  0.33 Ideal J     SI1      61.1    56   403  4.49  4.55  2.76
##  8  0.23 Ideal G     VS1      61.9    54   404  3.93  3.95  2.44
##  9  0.32 Ideal I     SI1      60.9    55   404  4.45  4.48  2.72
## 10  0.3  Ideal I     SI2      61      59   405  4.3   4.33  2.63
## # ... with 21,541 more rows
```

1.5 - Slice

Unlike filter(), slice() function chooses rows by their ordinal position in the tbl. Grouped tbls use the ordinal position within the group. Vector indexing is required to pass into the slice() function.

```r
# e.g. slicing the first 5 rows of data from the data frame
diamonds %>% slice(1:5)
```

```
## # A tibble: 5 x 10
##   carat cut     color clarity depth table price     x     y     z
##   <dbl> <ord>   <ord> <ord>   <dbl> <dbl> <int> <dbl> <dbl> <dbl>
## 1  0.23 Ideal   E     SI2      61.5    55   326  3.95  3.98  2.43
## 2  0.21 Premium E     SI1      59.8    61   326  3.89  3.84  2.31
## 3  0.23 Good    E     VS1      56.9    65   327  4.05  4.07  2.31
## 4  0.29 Premium I     VS2      62.4    58   334  4.2   4.23  2.63
## 5  0.31 Good    J     SI2      63.3    58   335  4.34  4.35  2.75
```

```r
# Suppose we want to slice the first 5 rows, the 8th row, and 15th to 20th rows
diamonds %>% slice(1:5, 8, 15:20)
```

```
## # A tibble: 12 x 10
##    carat cut       color clarity depth table price     x     y     z
##    <dbl> <ord>     <ord> <ord>   <dbl> <dbl> <int> <dbl> <dbl> <dbl>
##  1  0.23 Ideal     E     SI2      61.5    55   326  3.95  3.98  2.43
##  2  0.21 Premium   E     SI1      59.8    61   326  3.89  3.84  2.31
##  3  0.23 Good      E     VS1      56.9    65   327  4.05  4.07  2.31
##  4  0.29 Premium   I     VS2      62.4    58   334  4.2   4.23  2.63
##  5  0.31 Good      J     SI2      63.3    58   335  4.34  4.35  2.75
##  6  0.26 Very Good H     SI1      61.9    55   337  4.07  4.11  2.53
##  7  0.2  Premium   E     SI2      60.2    62   345  3.79  3.75  2.27
##  8  0.32 Premium   E     I1       60.9    58   345  4.38  4.42  2.68
##  9  0.3  Ideal     I     SI2      62      54   348  4.31  4.34  2.68
## 10  0.3  Good      J     SI1      63.4    54   351  4.23  4.29  2.7
## 11  0.3  Good      J     SI1      63.8    56   351  4.23  4.26  2.71
## 12  0.3  Very Good J     SI1      62.7    59   351  4.21  4.27  2.66
```

Note that the return data frame will not have the original index. When using a negative value, we are removing the row.

```r
# e.g. removing the first row
diamonds %>% slice(-1)
```

```
## # A tibble: 53,939 x 10
##    carat cut       color clarity depth table price     x     y     z
##    <dbl> <ord>     <ord> <ord>   <dbl> <dbl> <int> <dbl> <dbl> <dbl>
##  1  0.21 Premium   E     SI1      59.8    61   326  3.89  3.84  2.31
##  2  0.23 Good      E     VS1      56.9    65   327  4.05  4.07  2.31
##  3  0.29 Premium   I     VS2      62.4    58   334  4.2   4.23  2.63
##  4  0.31 Good      J     SI2      63.3    58   335  4.34  4.35  2.75
##  5  0.24 Very Good J     VVS2     62.8    57   336  3.94  3.96  2.48
##  6  0.24 Very Good I     VVS1     62.3    57   336  3.95  3.98  2.47
##  7  0.26 Very Good H     SI1      61.9    55   337  4.07  4.11  2.53
##  8  0.22 Fair      E     VS2      65.1    61   337  3.87  3.78  2.49
##  9  0.23 Very Good H     VS1      59.4    61   338  4     4.05  2.39
## 10  0.3  Good      J     SI1      64      55   339  4.25  4.28  2.73
## # ... with 53,929 more rows
```

1.6 - Mutate

mutate() function is used to update values of a column or adding new column to the data frame.

```r
# e.g. Adding a new column using price divided by carat
diamonds %>% mutate(price/carat)
```

```
## # A tibble: 53,940 x 11
##    carat cut     color clarity depth table price     x     y     z `price/carat`
##    <dbl> <ord>   <ord> <ord>   <dbl> <dbl> <int> <dbl> <dbl> <dbl>         <dbl>
##  1  0.23 Ideal   E     SI2      61.5    55   326  3.95  3.98  2.43         1417.
##  2  0.21 Premium E     SI1      59.8    61   326  3.89  3.84  2.31         1552.
##  3  0.23 Good    E     VS1      56.9    65   327  4.05  4.07  2.31         1422.
##  4  0.29 Premium I     VS2      62.4    58   334  4.2   4.23  2.63         1152.
##  5  0.31 Good    J     SI2      63.3    58   335  4.34  4.35  2.75         1081.
##  6  0.24 Very G~ J     VVS2     62.8    57   336  3.94  3.96  2.48         1400
##  7  0.24 Very G~ I     VVS1     62.3    57   336  3.95  3.98  2.47         1400
##  8  0.26 Very G~ H     SI1      61.9    55   337  4.07  4.11  2.53         1296.
##  9  0.22 Fair    E     VS2      65.1    61   337  3.87  3.78  2.49         1532.
## 10  0.23 Very G~ H     VS1      59.4    61   338  4     4.05  2.39         1470.
## # ... with 53,930 more rows
```

```
# Display the new column, by selecting the columns in the original data frame,
# then pass into mutate()
diamonds %>% select(carat, price) %>% mutate(price/carat)
```

```
## # A tibble: 53,940 x 3
##    carat price ‘price/carat‘
##    <dbl> <int>         <dbl>
##  1  0.23   326         1417.
##  2  0.21   326         1552.
##  3  0.23   327         1422.
##  4  0.29   334         1152.
##  5  0.31   335         1081.
##  6  0.24   336         1400
##  7  0.24   336         1400
##  8  0.26   337         1296.
##  9  0.22   337         1532.
## 10  0.23   338         1470.
## # ... with 53,930 more rows
```

```
# The new added column will not be given a column name
# We can define the name "Ratio" to the new column,
diamonds %>% select(carat, price) %>% mutate(Ratio=price/carat)
```

```
## # A tibble: 53,940 x 3
##    carat price Ratio
##    <dbl> <int> <dbl>
##  1  0.23   326 1417.
##  2  0.21   326 1552.
##  3  0.23   327 1422.
##  4  0.29   334 1152.
##  5  0.31   335 1081.
##  6  0.24   336 1400
##  7  0.24   336 1400
##  8  0.26   337 1296.
##  9  0.22   337 1532.
## 10  0.23   338 1470.
## # ... with 53,930 more rows
```

```
# The newly added column can be used in the same mutate().
# For instance, creating the "Ratio" column and a "Double" column by multiplying the "Ratio" by 2.
diamonds %>%
  select(carat, price) %>%
  mutate(Ratio=price/carat, Double=Ratio*2)
```

```
## # A tibble: 53,940 x 4
##    carat price Ratio Double
##    <dbl> <int> <dbl>  <dbl>
##  1  0.23   326 1417.  2835.
##  2  0.21   326 1552.  3105.
##  3  0.23   327 1422.  2843.
##  4  0.29   334 1152.  2303.
##  5  0.31   335 1081.  2161.
##  6  0.24   336 1400   2800
##  7  0.24   336 1400   2800
##  8  0.26   337 1296.  2592.
##  9  0.22   337 1532.  3064.
## 10  0.23   338 1470.  2939.
## # ... with 53,930 more rows
```

```
# Note that the code we use previously will not make change of the orignal data frame.
# It can be saved into a new data frame object.
# e.g.
diamonds2 <- diamonds %>%
  select(carat, price) %>%
  mutate(Ratio=price/carat, Double=Ratio*2)

diamonds2
```

```
## # A tibble: 53,940 x 4
##    carat price Ratio Double
##    <dbl> <int> <dbl>  <dbl>
##  1  0.23   326 1417.  2835.
##  2  0.21   326 1552.  3105.
##  3  0.23   327 1422.  2843.
##  4  0.29   334 1152.  2303.
##  5  0.31   335 1081.  2161.
##  6  0.24   336 1400   2800
##  7  0.24   336 1400   2800
##  8  0.26   337 1296.  2592.
##  9  0.22   337 1532.  3064.
## 10  0.23   338 1470.  2939.
## # ... with 53,930 more rows
```

```
# We can continue to add new column to the new data frame
diamonds2 <- diamonds2 %>%
  mutate(Quadruple=Double*2)

diamonds2
```

```
## # A tibble: 53,940 x 5
##    carat price Ratio Double Quadruple
##    <dbl> <int> <dbl>  <dbl>     <dbl>
##  1  0.23   326 1417.  2835.     5670.
##  2  0.21   326 1552.  3105.     6210.
##  3  0.23   327 1422.  2843.     5687.
##  4  0.29   334 1152.  2303.     4607.
##  5  0.31   335 1081.  2161.     4323.
##  6  0.24   336 1400   2800      5600
##  7  0.24   336 1400   2800      5600
##  8  0.26   337 1296.  2592.     5185.
##  9  0.22   337 1532.  3064.     6127.
## 10  0.23   338 1470.  2939.     5878.
## # ... with 53,930 more rows
```

```
# magrittr package also has a pipe operator (%<>%) to mutate the data frame
# e.g.
diamonds3 <- diamonds
diamonds3
```

```
## # A tibble: 53,940 x 10
##    carat cut       color clarity depth table price     x     y     z
##    <dbl> <ord>     <ord> <ord>   <dbl> <dbl> <int> <dbl> <dbl> <dbl>
##  1  0.23 Ideal     E     SI2      61.5    55   326  3.95  3.98  2.43
##  2  0.21 Premium   E     SI1      59.8    61   326  3.89  3.84  2.31
##  3  0.23 Good      E     VS1      56.9    65   327  4.05  4.07  2.31
##  4  0.29 Premium   I     VS2      62.4    58   334  4.2   4.23  2.63
##  5  0.31 Good      J     SI2      63.3    58   335  4.34  4.35  2.75
##  6  0.24 Very Good J     VVS2     62.8    57   336  3.94  3.96  2.48
##  7  0.24 Very Good I     VVS1     62.3    57   336  3.95  3.98  2.47
##  8  0.26 Very Good H     SI1      61.9    55   337  4.07  4.11  2.53
##  9  0.22 Fair      E     VS2      65.1    61   337  3.87  3.78  2.49
## 10  0.23 Very Good H     VS1      59.4    61   338  4     4.05  2.39
## # ... with 53,930 more rows
```

```
diamonds3 %<>%
  select(carat, price) %>%
  mutate(Ratio=price/carat, Double=Ratio*2)
diamonds3
```

```
## # A tibble: 53,940 x 4
##    carat price Ratio Double
##    <dbl> <int> <dbl>  <dbl>
##  1  0.23   326 1417.  2835.
##  2  0.21   326 1552.  3105.
##  3  0.23   327 1422.  2843.
##  4  0.29   334 1152.  2303.
##  5  0.31   335 1081.  2161.
##  6  0.24   336 1400   2800
##  7  0.24   336 1400   2800
##  8  0.26   337 1296.  2592.
##  9  0.22   337 1532.  3064.
## 10  0.23   338 1470.  2939.
## # ... with 53,930 more rows
```

1.7 - Summarize

summarize() function will return mean, max, median or other similar statistics. It allows direct call of the column name from the data frame, similar to with() in regular R.

```
# e.g. suppose we are interested the mean price in diamonds data set.
summarize(diamonds, mean(price))
```

```
## # A tibble: 1 x 1
##   ‘mean(price)‘
##           <dbl>
## 1         3933.
```

```
# Using pipe operator,
diamonds %>% summarize(mean(price))
```

```
## # A tibble: 1 x 1
##   ‘mean(price)‘
##           <dbl>
## 1         3933.
```

```
# One of the advantage using summarize() is that we can nest several statistics in one line of code.
diamonds %>%
  summarize(AvgPrice = mean(price),
            MedianPrice = median(price),
            AvgCarat = mean(carat))
```

```
## # A tibble: 1 x 3
##   AvgPrice MedianPrice AvgCarat
##      <dbl>       <dbl>    <dbl>
## 1    3933.        2401    0.798
```

1.8 - Group By

group_by() function complement the summarize() and make it more powerful to use. In most cases, we segment the data frame into different groups, then pass into summarize() function for specific statistics.

```
# e.g. suppose we are interested to know the average price for each cut grade
diamonds %>%
  group_by(cut) %>%
  summarize(AvgPrice = mean(price))
```

```
## # A tibble: 5 x 2
##   cut       AvgPrice
##   <ord>        <dbl>
## 1 Fair         4359.
## 2 Good         3929.
## 3 Very Good    3982.
## 4 Premium      4584.
## 5 Ideal        3458.
```

```
# The combination (group_by() & summarize()) is much more efficient (faster) that using aggregate()
# It also improves the readability and nesting process.
diamonds %>%
  group_by(cut) %>%
  summarize(AvgPrice=mean(price), SumCarat=sum(carat))
```

```
## # A tibble: 5 x 3
##   cut       AvgPrice SumCarat
##   <ord>        <dbl>    <dbl>
## 1 Fair         4359.    1684.
## 2 Good         3929.    4166.
## 3 Very Good    3982.    9743.
## 4 Premium      4584.   12301.
## 5 Ideal        3458.   15147.
```

```
diamonds %>%
  group_by(cut, color) %>%
  summarize(AvgPrice=mean(price), SumCarat=sum(carat))
```

```
## `summarise()` has grouped output by 'cut'. You can override using the `.groups` argument.
```

```
## # A tibble: 35 x 4
## # Groups:   cut [5]
##    cut   color AvgPrice SumCarat
##    <ord> <ord>    <dbl>    <dbl>
##  1 Fair  D        4291.     150.
##  2 Fair  E        3682.     192.
##  3 Fair  F        3827.     282.
##  4 Fair  G        4239.     321.
##  5 Fair  H        5136.     369.
##  6 Fair  I        4685.     210.
##  7 Fair  J        4976.     160.
##  8 Good  D        3405.     493.
##  9 Good  E        3424.     695.
## 10 Good  F        3496.     705.
## # ... with 25 more rows
```

1.9 - Arrange

arrange() function can be used for sorting and ordering the data. Its application is more intuitive than the regular R order() and sort().

```
# e.g. suppose we want to order the group_by summary by the average price for each cut grade.
diamonds %>%
  group_by(cut) %>%
  summarize(AvgPrice=mean(price), SumCarat=sum(carat)) %>%
  arrange(AvgPrice)
```

```
## # A tibble: 5 x 3
##   cut       AvgPrice SumCarat
##   <ord>        <dbl>    <dbl>
## 1 Ideal        3458.   15147.
## 2 Good         3929.    4166.
## 3 Very Good    3982.    9743.
## 4 Fair         4359.    1684.
## 5 Premium      4584.   12301.
```

```
# Note that the data frame will be ordered in an ascending order by default
# We can arrange the data frame in descending order with desc()
diamonds %>%
  group_by(cut) %>%
  summarize(AvgPrice=mean(price), SumCarat=sum(carat)) %>%
  arrange(desc(AvgPrice))
```

```
## # A tibble: 5 x 3
##   cut       AvgPrice SumCarat
##   <ord>        <dbl>    <dbl>
## 1 Premium      4584.   12301.
## 2 Fair         4359.    1684.
## 3 Very Good    3982.    9743.
## 4 Good         3929.    4166.
## 5 Ideal        3458.   15147.
```

1.10 - Do

do() is a general purpose complement to the specialized manipulation functions, such as filter(), select(), mutate(), summarize(), and arrange(). We can also use do() to perform arbitrary computation, returning either a data frame or arbitrary objects which will be sorted in a list.

```
# e.g. suppose we are interested to get the top N prices in each cut grade
# First we create a function topN to arrange the price in descending order and return N rows
topN <- function(x, N=5){
  x %>% arrange(desc(price)) %>% head(N)
}

# We then nest the do() and group_by() to identify the top prices in each cut group.
diamonds %>% group_by(cut) %>% do(topN(., N=3))
```

```
## # A tibble: 15 x 10
## # Groups:   cut [5]
##    carat cut       color clarity depth table price     x     y     z
##    <dbl> <ord>     <ord> <ord>   <dbl> <dbl> <int> <dbl> <dbl> <dbl>
##  1  2.01 Fair      G     SI1      70.6    64 18574  7.43  6.64  4.69
##  2  2.02 Fair      H     VS2      64.5    57 18565  8     7.95  5.14
##  3  4.5  Fair      J     I1       65.8    58 18531 10.2  10.2   6.72
##  4  2.8  Good      G     SI2      63.8    58 18788  8.9   8.85  0
##  5  2.07 Good      I     VS2      61.8    61 18707  8.12  8.16  5.03
##  6  2.67 Good      F     SI2      63.8    58 18686  8.69  8.64  5.54
##  7  2    Very Good G     SI1      63.5    56 18818  7.9   7.97  5.04
##  8  2    Very Good H     SI1      62.8    57 18803  7.95  8     5.01
##  9  2.03 Very Good H     SI1      63      60 18781  8     7.93  5.02
## 10  2.29 Premium   I     VS2      60.8    60 18823  8.5   8.47  5.16
## 11  2.29 Premium   I     SI1      61.8    59 18797  8.52  8.45  5.24
## 12  2.04 Premium   H     SI1      58.1    60 18795  8.37  8.28  4.84
## 13  1.51 Ideal     G     IF       61.7    55 18806  7.37  7.41  4.56
## 14  2.07 Ideal     G     SI2      62.5    55 18804  8.2   8.13  5.11
## 15  2.15 Ideal     G     SI2      62.6    54 18791  8.29  8.35  5.21
```

```
# Note that the return object from the previous comment is a data frame.
# If we define the return object name in the do(), it will return as a list for each cut grade.
diamonds %>% group_by(cut) %>% do(top = topN(., N=3))
```

```
## # A tibble: 5 x 2
## # Rowwise:
##   cut       top
##   <ord>     <list>
## 1 Fair      <tibble [3 x 10]>
## 2 Good      <tibble [3 x 10]>
## 3 Very Good <tibble [3 x 10]>
## 4 Premium   <tibble [3 x 10]>
## 5 Ideal     <tibble [3 x 10]>
```

```
# We can retreive the list elements if it's saved as data frame.
topByCut <- diamonds %>% group_by(cut) %>% do(top = topN(., N=3))

# The data frame has 5 rows and each rows contains a list of 3 rows data
class(topByCut)
```

```
## [1] "rowwise_df" "tbl_df"     "tbl"        "data.frame"
```

```
class(topByCut$top)
```

```
## [1] "list"
```

```
class(topByCut$top[[1]])
```

```
## [1] "tbl_df"     "tbl"        "data.frame"
```

```
# The first row in topByCut data frame will return the cut grade "Fair" prices
topByCut$top[[1]]
```

```
## # A tibble: 3 x 10
##   carat cut   color clarity depth table price     x     y     z
##   <dbl> <ord> <ord> <ord>   <dbl> <dbl> <int> <dbl> <dbl> <dbl>
## 1  2.01 Fair  G     SI1       70.6    64 18574  7.43  6.64  4.69
## 2  2.02 Fair  H     VS2       64.5    57 18565  8     7.95  5.14
## 3  4.5  Fair  J     I1        65.8    58 18531 10.2  10.2   6.72
```

2 - Tidyverse

2.1 - Combine Rows and Columns

Similar to base R rbind() and cbind(), dplyr has the similar functions bind_rows() and bind_cols(). The two are not exactly the same, which dplyr functions only apply to data.frame or tibble. Base R functions can be more generally applied to combine vectors into matrices or data.frame.

```
# Import dplyr package
library(dplyr)
library(tibble)

# Create a tibble with two columns
sportLeague <- tibble(sport=c("Hockey", "Baseball", "Foodball", "Basketball"),
                      league=c("NHL", "MLB", "NFL", "NBA"))

# Create a tibble with one column
trophy <- tibble(trophy=c("Stanley Cup", "Commissioner's Trophy",
                          "Vince Lombardi Trophy", "Larry O'Brien Trophy"))

# Combine the two tibbles into one
trophies1 <- bind_cols(sportLeague, trophy)

# Use tribble create another tibble
trophies2 <- tribble(
  ~sport, ~league, ~trophy,
  "Golf", "PGA", "Wanamaker Trophy",
  "Tennis", "Wimbledon", "Wimbledon Trophy"
)

# Combine the trophies1 with trophies2 (adding new row)
trophies <- bind_rows(trophies1, trophies2)

trophies
```

```
## # A tibble: 6 x 3
##   sport      league    trophy
##   <chr>      <chr>     <chr>
## 1 Hockey     NHL       Stanley Cup
## 2 Baseball   MLB       Commissioner’s Trophy
## 3 Foodball   NFL       Vince Lombardi Trophy
## 4 Basketball NBA       Larry O’Brien Trophy
## 5 Golf       PGA       Wanamaker Trophy
## 6 Tennis     Wimbledon Wimbledon Trophy
```

Note: bind_cols and bind_rows can be used to combine multiple tibble or data.frame

2.2 - Join

Joining table or data.frame is very important first step in data manipulation. In base R, we can use plyr or data.table to join two tables or data.frames. With dplyr package, we can use left_join(), right_join(), inner_join(), full_join(), semi_join(), and anti_join() for different join settings. We are using “diamonds” data set to demonstrate the use of join functions in dplyr.

```
library(readr)
colorsURL <- 'http://www.jaredlander.com/data/DiamondColors.csv'
diamondColors <- read_csv(colorsURL)
```

```
##
## -- Column specification --------------------------------------------------------
## cols(
##   Color = col_character(),
##   Description = col_character(),
##   Details = col_character()
## )
```

```
diamondColors
```

```
## # A tibble: 10 x 3
##    Color Description          Details
##    <chr> <chr>                <chr>
##  1 D     Absolutely Colorless No color
##  2 E     Colorless            Minute traces of color
##  3 F     Colorless            Minute traces of color
##  4 G     Near Colorless       Color is dificult to detect
##  5 H     Near Colorless       Color is dificult to detect
##  6 I     Near Colorless       Slightly detectable color
##  7 J     Near Colorless       Slightly detectable color
##  8 K     Faint Color          Noticeable color
##  9 L     Faint Color          Noticeable color
## 10 M     Faint Color          Noticeable color
```

```
data(diamonds, package='ggplot2')
unique(diamonds$color)
```

```
## [1] E I J H F G D
## Levels: D < E < F < G < H < I < J
```

```
class(diamonds)
```

```
## [1] "tbl_df"     "tbl"        "data.frame"
```

```
library(dplyr)
```

Using left_join() with column 'color' in diamonds and 'Color' in diamondColors. Note that we are defining "diamonds" as the left tbl and "diamondColors" as the right tbl. We are joining the two tbls with different column names "color" from left and "Color" from right. When using argument "by", a vector of equality of the string for left table column name and right table column name.

```
left_join(diamonds, diamondColors, by=c('color'='Color'))
```

```
## # A tibble: 53,940 x 12
##    carat cut       color clarity depth table price     x     y     z Description
##    <dbl> <ord>     <chr> <ord>   <dbl> <dbl> <int> <dbl> <dbl> <dbl> <chr>
##  1  0.23 Ideal     E     SI2      61.5    55   326  3.95  3.98  2.43 Colorless
##  2  0.21 Premium   E     SI1      59.8    61   326  3.89  3.84  2.31 Colorless
##  3  0.23 Good      E     VS1      56.9    65   327  4.05  4.07  2.31 Colorless
##  4  0.29 Premium   I     VS2      62.4    58   334  4.2   4.23  2.63 Near Colorl~
##  5  0.31 Good      J     SI2      63.3    58   335  4.34  4.35  2.75 Near Colorl~
##  6  0.24 Very Go~ J     VVS2     62.8    57   336  3.94  3.96  2.48 Near Colorl~
##  7  0.24 Very Go~ I     VVS1     62.3    57   336  3.95  3.98  2.47 Near Colorl~
## 8  0.26 Very Go~ H     SI1      61.9    55   337  4.07  4.11  2.53 Near Colorl~
## 9  0.22 Fair     E     VS2      65.1    61   337  3.87  3.78  2.49 Colorless
## 10 0.23 Very Go~ H     VS1      59.4    61   338  4     4.05  2.39 Near Colorl~
## # ... with 53,930 more rows, and 1 more variable: Details <chr>
```

```
# Note: Since the data type of the two joined columns are different ('color' is factor and 'Color' is c
# after joining the two tbls, an warning message stated the column will be forced to be "character".

# If we only want to extract some specific columns, we can also use the pipe operator
# e.g. only select carat, color, price, description, and details columns after join
left_join(diamonds, diamondColors, by=c('color'='Color')) %>%
  select(carat, color, price, Description, Details)
```

```
## # A tibble: 53,940 x 5
##    carat color price Description    Details
##    <dbl> <chr> <int> <chr>          <chr>
##  1  0.23 E       326 Colorless      Minute traces of color
##  2  0.21 E       326 Colorless      Minute traces of color
##  3  0.23 E       327 Colorless      Minute traces of color
##  4  0.29 I       334 Near Colorless Slightly detectable color
##  5  0.31 J       335 Near Colorless Slightly detectable color
##  6  0.24 J       336 Near Colorless Slightly detectable color
##  7  0.24 I       336 Near Colorless Slightly detectable color
##  8  0.26 H       337 Near Colorless Color is dificult to detect
##  9  0.22 E       337 Colorless      Minute traces of color
## 10  0.23 H       338 Near Colorless Color is dificult to detect
## # ... with 53,930 more rows
```

```
# Note: A left join will keep all fo the left tbl rows and match the rows from the right tbl.
# If a value in the right tbl cannot be found in the left tbl, it will be dropped.
# As observed, the joined tbl "Color" and "Description" distinct count is less than the "diamondColors"

# Before Join:
diamondColors %>% distinct(Color, Description)  # total 10 colors in the original tbl
```

```
## # A tibble: 10 x 2
##    Color Description
##    <chr> <chr>
##  1 D     Absolutely Colorless
##  2 E     Colorless
##  3 F     Colorless
##  4 G     Near Colorless
##  5 H     Near Colorless
##  6 I     Near Colorless
##  7 J     Near Colorless
##  8 K     Faint Color
##  9 L     Faint Color
## 10 M     Faint Color
```

```
# After Join:
left_join(diamonds, diamondColors, by=c('color'='Color')) %>%
  distinct(color, Details)  # only 7 colors were matched to the left tbl
```

```
## # A tibble: 7 x 2
##   color Details
##   <chr> <chr>
## 1 E     Minute traces of color
## 2 I     Slightly detectable color
## 3 J     Slightly detectable color
## 4 H     Color is dificult to detect
## 5 F     Minute traces of color
## 6 G     Color is dificult to detect
## 7 D     No color
```

```
# Using a right_join() function, we are keeping all of the existing rows from the right tbl
# and match with the rows in the left tbl.
# In right join case, the joined tbl contains more rows than the left tbl (diamonds).
# Before Join:
diamonds %>% nrow
```

```
## [1] 53940
```

```
# After Join:
right_join(diamonds, diamondColors, by=c('color'='Color')) %>%
  nrow
```

```
## [1] 53943
```

```
# inner_join() returns a joined tbl with all the matches from the left and right tbls.
inner_join(diamonds, diamondColors, by=c('color'='Color'))
```

```
## # A tibble: 53,940 x 12
##    carat cut       color clarity depth table price     x     y     z Description
##    <dbl> <ord>     <chr> <ord>   <dbl> <dbl> <int> <dbl> <dbl> <dbl> <chr>
##  1  0.23 Ideal     E     SI2      61.5    55   326  3.95  3.98  2.43 Colorless
##  2  0.21 Premium   E     SI1      59.8    61   326  3.89  3.84  2.31 Colorless
##  3  0.23 Good      E     VS1      56.9    65   327  4.05  4.07  2.31 Colorless
##  4  0.29 Premium   I     VS2      62.4    58   334  4.2   4.23  2.63 Near Colorl~
##  5  0.31 Good      J     SI2      63.3    58   335  4.34  4.35  2.75 Near Colorl~
##  6  0.24 Very Go~ J     VVS2     62.8    57   336  3.94  3.96  2.48 Near Colorl~
##  7  0.24 Very Go~ I     VVS1     62.3    57   336  3.95  3.98  2.47 Near Colorl~
##  8  0.26 Very Go~ H     SI1      61.9    55   337  4.07  4.11  2.53 Near Colorl~
##  9  0.22 Fair      E     VS2      65.1    61   337  3.87  3.78  2.49 Colorless
## 10  0.23 Very Go~ H     VS1      59.4    61   338  4     4.05  2.39 Near Colorl~
## # ... with 53,930 more rows, and 1 more variable: Details <chr>
```

```
# In this example, the inner_join() should return the same rows as the left_join() because
# the right tbl has some rows that cannot be match with the left tbl.
all.equal(left_join(diamonds, diamondColors, by=c('color'='Color')),
          inner_join(diamonds, diamondColors, by=c('color'='Color')))
```

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

```
# full_join() (usually called "Outter Join") will joined tbl with with all the rows from the two tbls,
# even without match.
full_join(diamonds, diamondColors, by=c('color'='Color'))
```

```
## # A tibble: 53,943 x 12
##    carat cut       color clarity depth table price     x     y     z Description
##    <dbl> <ord>     <chr> <ord>   <dbl> <dbl> <int> <dbl> <dbl> <dbl> <chr>
##  1  0.23 Ideal     E     SI2      61.5    55   326  3.95  3.98  2.43 Colorless
##  2  0.21 Premium   E     SI1      59.8    61   326  3.89  3.84  2.31 Colorless
##  3  0.23 Good      E     VS1      56.9    65   327  4.05  4.07  2.31 Colorless
##  4  0.29 Premium   I     VS2      62.4    58   334  4.2   4.23  2.63 Near Colorl~
##  5  0.31 Good      J     SI2      63.3    58   335  4.34  4.35  2.75 Near Colorl~
##  6  0.24 Very Go~ J     VVS2     62.8    57   336  3.94  3.96  2.48 Near Colorl~
##  7  0.24 Very Go~ I     VVS1     62.3    57   336  3.95  3.98  2.47 Near Colorl~
##  8  0.26 Very Go~ H     SI1      61.9    55   337  4.07  4.11  2.53 Near Colorl~
##  9  0.22 Fair      E     VS2      65.1    61   337  3.87  3.78  2.49 Colorless
## 10  0.23 Very Go~ H     VS1      59.4    61   338  4     4.05  2.39 Near Colorl~
## # ... with 53,933 more rows, and 1 more variable: Details <chr>
```

```
# In this example, the full_join() should return the same rows as the right_join() because
# the right tbl has 7 rows that cannot be match to the left tbl and will be included.
all.equal(right_join(diamonds, diamondColors, by=c('color'='Color')),
          full_join(diamonds, diamondColors, by=c('color'='Color')))
```

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

```
# semi_join() returns only the first match from the left tbl to the right tbl, which is more like sorti
# If we set "diamondColors" as the left tbl, only the matched colors found in "diamonds" tbl will be re
semi_join(diamondColors, diamonds, by=c('Color'='color'))
```

```
## # A tibble: 7 x 3
##   Color Description          Details
##   <chr> <chr>                <chr>
## 1 D     Absolutely Colorless No color
## 2 E     Colorless            Minute traces of color
## 3 F     Colorless            Minute traces of color
## 4 G     Near Colorless       Color is dificult to detect
## 5 H     Near Colorless       Color is dificult to detect
## 6 I     Near Colorless       Slightly detectable color
## 7 J     Near Colorless       Slightly detectable color
```

```
# anti_join() is the opposite of the semi_join(), which returns the unmatch rows from the left tbl.
# Since no color "K", "L", and "M" can be found in the 'diamonds' tbl, so anti_join() will return the t
anti_join(diamondColors, diamonds, by=c('Color'='color'))
```

```
## # A tibble: 3 x 3
##   Color Description Details
##   <chr> <chr>       <chr>
## 1 K     Faint Color Noticeable color
## 2 L     Faint Color Noticeable color
## 3 M     Faint Color Noticeable color
```

```
# We can also apply the filter() and unique() to achieve the semi_join() or anti_join(), but the later
# when dealing with data.frame.
# semi_join() result,
diamondColors %>% filter(Color %in% unique(diamonds$color))
```

```
## # A tibble: 7 x 3
##   Color Description          Details
##   <chr> <chr>                <chr>
## 1 D     Absolutely Colorless No color
## 2 E     Colorless            Minute traces of color
## 3 F     Colorless            Minute traces of color
## 4 G     Near Colorless       Color is dificult to detect
## 5 H     Near Colorless       Color is dificult to detect
## 6 I     Near Colorless       Slightly detectable color
## 7 J     Near Colorless       Slightly detectable color
```

```
# anti_join() result,
diamondColors %>% filter(!Color %in% unique(diamonds$color))
```

```
## # A tibble: 3 x 3
##   Color Description Details
##   <chr> <chr>       <chr>
## 1 K     Faint Color Noticeable color
## 2 L     Faint Color Noticeable color
## 3 M     Faint Color Noticeable color
```

3.3 - Transform Data Format

Both base R or melt() and dcast() in reshape2 package can be used to make transformation of the wide format data and long format data. tidyr package is more like a advance version of reshape2 package. We are using the Columbia University reaction data set for demonstration.

```
# We are using readr package from Tidyverse to read the text file and save it into a tibble.
library(readr)
emotion <- read_tsv('http://www.jaredlander.com/data/reaction.txt')
```

```
##
## -- Column specification --------------------------------------------------------
## cols(
##   ID = col_double(),
##   Test = col_double(),
##   Age = col_double(),
##   Gender = col_character(),
##   BMI = col_double(),
##   React = col_double(),
##   Regulate = col_double()
## )
```

```
# Note: read_tsv() function will return a message about the data type extracted from the text file.

# Print the tibble / data.frame
emotion
```

```
## # A tibble: 99 x 7
##       ID  Test   Age Gender   BMI React Regulate
##    <dbl> <dbl> <dbl> <chr>  <dbl> <dbl>    <dbl>
##  1     1     1  9.69 F       14.7  4.17     3.15
##  2     1     2 12.3  F       14.6  3.89     2.55
##  3     2     1 15.7  F       19.5  4.39     4.41
##  4     2     2 17.6  F       20.0  2.03     2.2
##  5     3     1  9.52 F       20.9  3.38     2.65
##  6     3     2 11.8  F       24.0  4        3.63
##  7     4     1 16.3  M       25.1  3.15     3.59
##  8     4     2 18.8  M       28.0  3.02     3.54
##  9     5     1 15.8  M       28.4  3.08     2.64
## 10     5     2 18.2  M       19.6  3.17     2.29
## # ... with 89 more rows
```

Note that the return tibble is a wide format. We can tranform the data into a long format tibble by using gather(), which is similar to melt() in reshape2 package. We will stack the “Age”, “BMI”, “React”, and “Regulate” into a single column called “Measurement”. A new column will also be created and called “Type” to identify the column names being stacked.

```
library(tidyr)
```

```
##
## Attaching package: 'tidyr'
```

```
## The following object is masked from 'package:magrittr':
##
##     extract
```

```
emotion %>%
  gather(key=Type, value=Measurement, Age, BMI, React, Regulate)
```

```
## # A tibble: 396 x 5
##       ID  Test Gender Type  Measurement
##    <dbl> <dbl> <chr>  <chr>       <dbl>
##  1     1     1 F      Age          9.69
##  2     1     2 F      Age         12.3
##  3     2     1 F      Age         15.7
##  4     2     2 F      Age         17.6
##  5     3     1 F      Age          9.52
##  6     3     2 F      Age         11.8
##  7     4     1 M      Age         16.3
##  8     4     2 M      Age         18.8
##  9     5     1 M      Age         15.8
## 10     5     2 M      Age         18.2
## # ... with 386 more rows
```

Note: The first argument is the “key” which is used to identify the column names in the original tbl. The second argument “value” is to create a column with a new name from a collections of columns in the original tbl. After given the name to the new column, we then identify the column names that will be stacked into the new column.

```r
# The new tbl will be sorted by 'Type', which is the key defined in gather().
# It would be difficult to identify the changes of the data, so we can arrange it by ID.
emotionLong <- emotion %>%
  gather(key=Type, value=Measurement, Age, BMI, React, Regulate) %>%
  arrange(ID)

emotionLong
```

```
## # A tibble: 396 x 5
##       ID  Test Gender Type     Measurement
##    <dbl> <dbl> <chr>  <chr>          <dbl>
##  1     1     1 F      Age             9.69
##  2     1     2 F      Age            12.3
##  3     1     1 F      BMI            14.7
##  4     1     2 F      BMI            14.6
##  5     1     1 F      React           4.17
##  6     1     2 F      React           3.89
##  7     1     1 F      Regulate        3.15
##  8     1     2 F      Regulate        2.55
##  9     2     1 F      Age            15.7
## 10     2     2 F      Age            17.6
## # ... with 386 more rows
```

```r
# Note: In the original data, each ID has 2 rows and each row contains Age, BMI, React, and Regulate co
# After the transformation, each ID turns into 4 rows and each row will have a Type and measurement col

# We can also appoint the columns to be included in the return tbl, or using "-" to excluded in the ret
# e.g.
emotion %>%
  gather(key=Type, value=Measurement, -ID, -Test, -Gender) %>%
  arrange(ID)
```

```
## # A tibble: 396 x 5
##       ID  Test Gender Type     Measurement
##    <dbl> <dbl> <chr>  <chr>          <dbl>
##  1     1     1 F      Age             9.69
##  2     1     2 F      Age            12.3
##  3     1     1 F      BMI            14.7
##  4     1     2 F      BMI            14.6
##  5     1     1 F      React           4.17
##  6     1     2 F      React           3.89
##  7     1     1 F      Regulate        3.15
##  8     1     2 F      Regulate        2.55
##  9     2     1 F      Age            15.7
## 10     2     2 F      Age            17.6
## # ... with 386 more rows
```

```r
# Check to see if they are the same.
identical(
  emotion %>%
    gather(key=Type, value=Measurement, Age, BMI, React, Regulate) %>%
    arrange(ID),
  emotion %>%
    gather(key=Type, value=Measurement, -ID, -Test, -Gender) %>%
    arrange(ID)
)
```

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

Opposite to gather() is spread(), which is similar to dcast() in reshape2 package. spread() can transform the long format data into wide format data. In general, it can break the stacked data into columns.

```
# e.g. Suppose we are interested to break the emotionLong data into it's original form.
emotionLong %>%
  spread(key=Type, value=Measurement)
```

```
## # A tibble: 99 x 7
##       ID  Test Gender   Age   BMI React Regulate
##    <dbl> <dbl> <chr>  <dbl> <dbl> <dbl>    <dbl>
##  1     1     1 F       9.69  14.7  4.17     3.15
##  2     1     2 F      12.3   14.6  3.89     2.55
##  3     2     1 F      15.7   19.5  4.39     4.41
##  4     2     2 F      17.6   20.0  2.03     2.2
##  5     3     1 F       9.52  20.9  3.38     2.65
##  6     3     2 F      11.8   24.0  4        3.63
##  7     4     1 M      16.3   25.1  3.15     3.59
##  8     4     2 M      18.8   28.0  3.02     3.54
##  9     5     1 M      15.8   28.4  3.08     2.64
## 10     5     2 M      18.2   19.6  3.17     2.29
## # ... with 89 more rows
```