Data science with R

Tidyr Package

Kevins_Kacha

Maseno University

2/21/2022

Objects in R

R consists of a number of data objects to perform various functions. There are 6 types of objects in R Programming.

- Vectors: They are six types of atomic vectors- logical, integer, character, raw, double, and complex.
- LISTS: It contains various types of elements including strings, numbers, vectors, and a nested list inside it. It can also consist of matrices or functions as elements.
- MATRICES: They are used to arrange elements in the two-dimensional layout. They contain elements of the same data type usually numeric.
- ARRAY: It is used to store multi-dimensional data in the required format.
- FACTORS: Factors are data objects that are used in order to categorize and store data as levels. They can be strings or integers. They are extremely useful in data analytics for statistical modeling.
- DATAFRAME: Dataframe is a 2-dimensional data structure wherein each column consists of the value of one variable and each row consists of a value set from each column.

scalar

This is one dimensional object stored in R environment. A scalar data structure is the most basic data type that holds only a single atomic value at a time. Using scalars, more complex data types can be constructed. Let's look at the most commonly used scalar types in R.

- Numeric
- Character
- Integer
- Logical
- Complex example of scalar in R

```
a<-12  #scalar objects in R
a
```

```
## [1] 12
```

```
class(a) # states to which class the object is
```

```
## [1] "numeric"
```

matrix

Creating matrix in R A matrix is a two-dimensional, homogeneous data structure in R. This means that it has two dimensions, rows and columns.\\

A matrix can store data of a single basic type (numeric, logical, character, etc.). Therefore, a matrix can be a combination of two or more vectors.

```
ab<-matrix(1:9, nrow = 3) # ab is our created matrix
ab
```

```
##      [,1] [,2] [,3]
## [1,]    1    4    7
## [2,]    2    5    8
## [3,]    3    6    9
```

```
ac<-matrix(1:6, ncol = 2)
ac
```

```
##      [,1] [,2]
## [1,]    1    4
## [2,]    2    5
## [3,]    3    6
```

combinination of matrix

Using cbind to join two matrix together

```
ad<-cbind(ab,ac)
ad
```

```
##      [,1] [,2] [,3] [,4] [,5]
## [1,]    1    4    7    1    4
## [2,]    2    5    8    2    5
## [3,]    3    6    9    3    6
```

Dataframe

A data frame is a table or a two-dimensional array-like structure in which each column contains values of one variable and each row contains one set of values from each column.

Following are the characteristics of a data frame.

- The column names should be non-empty.
- The row names should be unique.
- The data stored in a data frame can be of numeric, factor or character type.
- Each column should contain same number of data items

```
# Create the data frame.
emp.data <- data.frame(emp_id = c (1:5),
   emp_name = c("Rick","Dan","Michelle","Ryan","Gary"),
   salary = c(623.3,515.2,611.0,729.0,843.25),
   start_date = as.Date(c("2012-01-01", "2013-09-23",
          "2014-11-15", "2014-05-11","2015-03-27")),
   stringsAsFactors = FALSE)
emp.data
```

```
##   emp_id emp_name salary start_date
## 1      1     Rick 623.30 2012-01-01
## 2      2      Dan 515.20 2013-09-23
## 3      3 Michelle 611.00 2014-11-15
## 4      4     Ryan 729.00 2014-05-11
## 5      5     Gary 843.25 2015-03-27
```

Packages

Loading the packages to be used

```
library(tidyverse, warn.conflicts = FALSE)
```

```
## -- Attaching packages --------------------------------------- tidyverse 1.3.1 --
## v ggplot2 3.3.5     v purrr   0.3.4
## v tibble  3.1.6     v dplyr   1.0.7
## v tidyr   1.1.4     v stringr 1.4.0
## v readr   2.1.2     v forcats 0.5.1
## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

```
library(tidyr)
library(dplyr)
```

Dataframes

```
size<-c(1,2,2,3,4,5,6,7)
weight<-c(1,2,3,5,6,7,8,4)
mouse.data<-data.frame(size,weight)
mouse.data
```

```
##   size weight
## 1    1      1
## 2    2      2
## 3    2      3
## 4    3      5
## 5    4      6
## 6    5      7
## 7    6      8
## 8    7      4
```

Tibble

```
size<-c(1,2,2,3,4,5,6,7)
weight<-c(1,2,3,5,6,7,8,4)
mouse.data <- tibble(size,weight)
mouse.data
```

```
## # A tibble: 8 x 2
##    size weight
##   <dbl>  <dbl>
## 1     1      1
## 2     2      2
## 3     2      3
## 4     3      5
## 5     4      6
## 6     5      7
## 7     6      8
## 8     7      4
```

gather()

gather() used to gather columns into key-value pairs.

```
table1
```

```
## # A tibble: 6 x 4
##   country      year  cases population
##   <chr>       <int>  <int>      <int>
## 1 Afghanistan  1999    745   19987071
## 2 Afghanistan  2000   2666   20595360
## 3 Brazil       1999  37737  172006362
## 4 Brazil       2000  80488  174504898
## 5 China        1999 212258 1272915272
## 6 China        2000 213766 1280428583
```

```
gather1<-gather(table1, key="case",value = "Totals",cases,population)
head(gather1) # it only views 6 rows
```

```
## # A tibble: 6 x 4
##   country      year case  Totals
##   <chr>       <int> <chr>  <int>
## 1 Afghanistan  1999 cases    745
## 2 Afghanistan  2000 cases   2666
## 3 Brazil       1999 cases  37737
## 4 Brazil       2000 cases  80488
## 5 China        1999 cases 212258
## 6 China        2000 cases 213766
```

gather using Iris data

Converts wide data to long data

```
head(iris) # it only views 6 rows
```

```
##   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
## 1          5.1         3.5          1.4         0.2  setosa
## 2          4.9         3.0          1.4         0.2  setosa
## 3          4.7         3.2          1.3         0.2  setosa
## 4          4.6         3.1          1.5         0.2  setosa
## 5          5.0         3.6          1.4         0.2  setosa
## 6          5.4         3.9          1.7         0.4  setosa
```

```
gather2<-gather(iris, key = "flower_att", value = "measurement",
       Sepal.Length, Sepal.Width, Petal.Length, Petal.Width)
# iris %>% gather(flower_att,measurement,-Species)
#iris %>% gather(key = "flower_att", value = "measurement", -Species)
head(gather2)
```

```
##   Species   flower_att measurement
## 1  setosa Sepal.Length         5.1
## 2  setosa Sepal.Length         4.9
## 3  setosa Sepal.Length         4.7
## 4  setosa Sepal.Length         4.6
## 5  setosa Sepal.Length         5.0
## 6  setosa Sepal.Length         5.4
```

pivot_longer()

Converts wide data to long data

```
head(iris) # it only views 6 rows
```

```
##   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
## 1          5.1         3.5          1.4         0.2  setosa
## 2          4.9         3.0          1.4         0.2  setosa
## 3          4.7         3.2          1.3         0.2  setosa
## 4          4.6         3.1          1.5         0.2  setosa
## 5          5.0         3.6          1.4         0.2  setosa
## 6          5.4         3.9          1.7         0.4  setosa
```

```
#pivot_longer(iris,c(Sepal.Length, Sepal.Width, Petal.Length, Petal.Width))
gather2<-pivot_longer(iris,-c(Species), names_to= "flower_att",
             values_to="measurement")
head(gather2)
```

```
## # A tibble: 6 x 3
##   Species flower_att   measurement
##   <fct>   <chr>              <dbl>
## 1 setosa  Sepal.Length         5.1
## 2 setosa  Sepal.Width          3.5
## 3 setosa  Petal.Length         1.4
## 4 setosa  Petal.Width          0.2
## 5 setosa  Sepal.Length         4.9
## 6 setosa  Sepal.Width          3
```

spread()

Converts long data to wide data

```
head(table2)
```

```
## # A tibble: 6 x 4
##   country      year type           count
##   <chr>       <int> <chr>          <int>
## 1 Afghanistan  1999 cases            745
## 2 Afghanistan  1999 population  19987071
## 3 Afghanistan  2000 cases           2666
## 4 Afghanistan  2000 population  20595360
## 5 Brazil       1999 cases           37737
## 6 Brazil       1999 population 172006362
```

```
#spread(table2, key = "type",value = "count")
spread1<-table2 %>% spread(key = "type",value = "count")
head(spread1)
```

```
## # A tibble: 6 x 4
##   country      year  cases population
##   <chr>       <int>  <int>      <int>
## 1 Afghanistan  1999    745   19987071
## 2 Afghanistan  2000   2666   20595360
## 3 Brazil       1999  37737  172006362
## 4 Brazil       2000  80488  174504898
## 5 China        1999 212258 1272915272
```

pivot_wider()

Converts long data to wide data

```
head(table2)
```

```
## # A tibble: 6 x 4
##   country      year type           count
##   <chr>       <int> <chr>          <int>
## 1 Afghanistan  1999 cases            745
## 2 Afghanistan  1999 population  19987071
## 3 Afghanistan  2000 cases           2666
## 4 Afghanistan  2000 population  20595360
## 5 Brazil       1999 cases           37737
## 6 Brazil       1999 population 172006362
```

```
#pivot_wider(table2, names_from = "type", values_from = "count")
spread2<-table2 %>% pivot_wider( names_from = "type", values_from = "count")
head(spread2)
```

```
## # A tibble: 6 x 4
##   country      year  cases population
##   <chr>       <int>  <int>      <int>
## 1 Afghanistan  1999    745   19987071
## 2 Afghanistan  2000   2666   20595360
## 3 Brazil       1999  37737  172006362
## 4 Brazil       2000  80488  174504898
## 5 China        1999 212258 1272915272
```

separate()

It is used to separate data in the one column into multiple columns.

```
table3
```

```
## # A tibble: 6 x 3
##   country      year rate
## * <chr>       <int> <chr>
## 1 Afghanistan  1999 745/19987071
## 2 Afghanistan  2000 2666/20595360
## 3 Brazil       1999 37737/172006362
## 4 Brazil       2000 80488/174504898
## 5 China        1999 212258/1272915272
## 6 China        2000 213766/1280428583
```

```
table3 %>% separate(rate,c("cases","population"),sep = "/")
```

```
## # A tibble: 6 x 4
##   country      year cases  population
##   <chr>       <int> <chr>  <chr>
## 1 Afghanistan  1999 745    19987071
## 2 Afghanistan  2000 2666   20595360
## 3 Brazil       1999 37737  172006362
## 4 Brazil       2000 80488  174504898
## 5 China        1999 212258 1272915272
## 6 China        2000 213766 1280428583
```

separate()

It is used to separate data in the one column into multiple columns.

```
stocks <- tibble(time = as.Date('2009-01-01') + 0:4,stock = rnorm(5, 0, 1)+15)
stocks
```

```
## # A tibble: 5 x 2
##   time       stock
##   <date>     <dbl>
## 1 2009-01-01  13.4
## 2 2009-01-02  13.2
## 3 2009-01-03  15.0
## 4 2009-01-04  14.2
## 5 2009-01-05  15.4
```

```
st<-stocks %>% separate(time,c("year","month","day"))
st
```

```
## # A tibble: 5 x 4
##   year  month day   stock
##   <chr> <chr> <chr> <dbl>
## 1 2009  01    01     13.4
## 2 2009  01    02     13.2
## 3 2009  01    03     15.0
## 4 2009  01    04     14.2
## 5 2009  01    05     15.4
```

unite()

This unites multiple columns to form one column.

```
st # the origal separated data can be united back
```

```
## # A tibble: 5 x 4
##   year  month day   stock
##   <chr> <chr> <chr> <dbl>
## 1 2009  01    01     13.4
## 2 2009  01    02     13.2
## 3 2009  01    03     15.0
## 4 2009  01    04     14.2
## 5 2009  01    05     15.4
```

```
st1<-st %>% unite("date",c("year","month","day"),sep = "-")
st1
```

```
## # A tibble: 5 x 2
##   date       stock
##   <chr>      <dbl>
## 1 2009-01-01  13.4
## 2 2009-01-02  13.2
## 3 2009-01-03  15.0
## 4 2009-01-04  14.2
## 5 2009-01-05  15.4
```

mutate()

It adds a column to the existing data in R

```
st1
```

```
## # A tibble: 5 x 2
##   date       stock
##   <chr>      <dbl>
## 1 2009-01-01  13.4
## 2 2009-01-02  13.2
## 3 2009-01-03  15.0
## 4 2009-01-04  14.2
## 5 2009-01-05  15.4
```

```
date2<-st1 %>% mutate(stock2 = stock/2)
date2
```

```
## # A tibble: 5 x 3
##   date       stock stock2
##   <chr>      <dbl>  <dbl>
## 1 2009-01-01  13.4   6.72
## 2 2009-01-02  13.2   6.62
## 3 2009-01-03  15.0   7.51
## 4 2009-01-04  14.2   7.11
## 5 2009-01-05  15.4   7.69
```

using chop() in tidyr

```
head(mouse.data)
```

```
## # A tibble: 6 x 2
##    size weight
##   <dbl>  <dbl>
## 1     1      1
## 2     2      2
## 3     2      3
## 4     3      5
## 5     4      6
## 6     5      7
```

```
mouse.data2<- mouse.data %>% chop(c(size))
mouse.data2
```

```
## # A tibble: 8 x 2
##   weight        size
##    <dbl> <list<dbl>>
## 1      1         [1]
## 2      2         [1]
## 3      3         [1]
## 4      5         [1]
## 5      6         [1]
## 6      7         [1]
## 7      8         [1]
## 8      4         [1]
```

unchop()

unchop recalls the exact position on which the data was initially been.

```
head(mouse.data2)
```

```
## # A tibble: 6 x 2
##   weight        size
##    <dbl> <list<dbl>>
## 1      1         [1]
## 2      2         [1]
## 3      3         [1]
## 4      5         [1]
## 5      6         [1]
## 6      7         [1]
```

```
mouse.data3<-mouse.data%>% unchop(size)
mouse.data3
```

```
## # A tibble: 8 x 2
##    size weight
##   <dbl>  <dbl>
## 1     1      1
## 2     2      2
## 3     2      3
## 4     3      5
## 5     4      6
## 6     5      7
```

creating a tibble in R

A tibble is just a lazy dataframe.

```
df <- tibble(
  group = c(1:2, 1, 2),
  item_id = c(1:2, 2, 3),
  item_name = c("a", "a", "b", "b"),
  value1 = c(1, NA, 3, 4),
  value2 = 4:7)
df
```

```
## # A tibble: 4 x 5
##   group item_id item_name value1 value2
##   <dbl>   <dbl> <chr>      <dbl>  <int>
## 1     1       1 a              1      4
## 2     2       2 a             NA      5
## 3     1       2 b              3      6
## 4     2       3 b              4      7
```

using Complete() function

It give all possible combination of the variables in question. what is combination of group,item_i and item_name?

```
complete(df, group, item_id, item_name)
```

```
## # A tibble: 12 x 5
##    group item_id item_name value1 value2
##    <dbl>   <dbl> <chr>      <dbl>  <int>
##  1     1       1 a              1      4
##  2     1       1 b             NA     NA
##  3     1       2 a             NA     NA
##  4     1       2 b              3      6
##  5     1       3 a             NA     NA
##  6     1       3 b             NA     NA
##  7     2       1 a             NA     NA
##  8     2       1 b             NA     NA
##  9     2       2 a             NA      5
## 10     2       2 b             NA     NA
## 11     2       3 a             NA     NA
## 12     2       3 b              4      7
```

Task

Cross all possible group values with the unique pairs of (item_id, item_name) that already exist in the data?

In this we use nesting function.

```
complete(df, group, nesting(item_id, item_name))
```

```
## # A tibble: 8 x 5
##   group item_id item_name value1 value2
##   <dbl>   <dbl> <chr>      <dbl>  <int>
## 1     1       1 a              1      4
## 2     1       2 a             NA     NA
## 3     1       2 b              3      6
## 4     1       3 b             NA     NA
## 5     2       1 a             NA     NA
## 6     2       2 a             NA      5
## 7     2       2 b             NA     NA
## 8     2       3 b              4      7
```

Missing values

When dealing with the missing values in a data set we use drop_na() to drops rows where any column specified contains a missing value.

```
drop_na(df) # yields same results as in the second
```

```
## # A tibble: 3 x 5
##   group item_id item_name value1 value2
##   <dbl>   <dbl> <chr>      <dbl>  <int>
## 1     1       1 a              1      4
## 2     1       2 b              3      6
## 3     2       3 b              4      7
```

```
df %>% drop_na()# yields same results as in the first
```

```
## # A tibble: 3 x 5
##   group item_id item_name value1 value2
##   <dbl>   <dbl> <chr>      <dbl>  <int>
## 1     1       1 a              1      4
## 2     1       2 b              3      6
## 3     2       3 b              4      7
```

Missing values

When dealing with the missing values in a specific column in data set we use drop_na(x,y) to drops rows where the specified column contains a missing value.

Dropping rowa based on x attribute

```
df <- tibble(x = c(1, 2, NA), y = c("a", NA, "b"))
df %>% drop_na() #dropping all missing values
```

```
## # A tibble: 1 x 2
##       x y
##   <dbl> <chr>
## 1     1 a
```

```
df %>% drop_na(x) #Dropping rows based on missing values x
```

```
## # A tibble: 2 x 2
##       x y
##   <dbl> <chr>
## 1     1 a
## 2     2 <NA>
```

Missing values

When dealing with the missing values in a specific column in data set we use drop_na(y) to drops rows where the specified column contains a missing value.

Dropping rows based on y attribute

```
df <- tibble(x = c(1, 2, NA), y = c("a", NA, "b"))
df %>% drop_na() #dropping all missing values
```

```
## # A tibble: 1 x 2
##       x y
##   <dbl> <chr>
## 1     1 a
```

```
df %>% drop_na(y) #Dropping rows based on missing values x
```

```
## # A tibble: 2 x 2
##       x y
##   <dbl> <chr>
## 1     1 a
## 2    NA b
```

Expand()

Expand() generates all combination of variables found in a dataset. Lets first create a dataset.

```
fruits <- tibble(type = c("apple", "orange", "apple",
                          "orange","orange", "orange"),
                 year = c(2010, 2010, 2012, 2010, 2011, 2012),
                 size = factor(c("XS", "S", "M", "S", "S", "M"),
                               levels = c("XS", "S", "M", "L")),
                 weights = rnorm(6, as.numeric(size) + 2))
fruits
```

```
## # A tibble: 6 x 4
##   type    year size  weights
##   <chr>  <dbl> <fct>   <dbl>
## 1 apple   2010 XS       2.52
## 2 orange  2010 S        2.20
## 3 apple   2012 M        5.87
## 4 orange  2010 S        5.22
## 5 orange  2011 S        4.55
## 6 orange  2012 M        4.18
```

Expand()

Expand() generates all combination of variables found in a dataset. Give all possible combinations fruits type and (fruit type and fruit size) not necessarily present in the dataset.

```
fruits %>% expand(type)
```

```
## # A tibble: 2 x 1
##   type
##   <chr>
## 1 apple
## 2 orange
```

```
fruits_ex1<-fruits %>% expand(type,size)
head(fruits_ex1)
```

```
## # A tibble: 6 x 2
##   type   size
##   <chr>  <fct>
## 1 apple  XS
## 2 apple  S
## 3 apple  M
## 4 apple  L
## 5 orange XS
## 6 orange S
```

Expand()

Expand() generates all combination of variables found in a dataset. Give all possible combinations fruits type, fruit size and year, not necessarily present in the dataset.

```
fruits_ex2<-fruits %>% expand(type, size, year)
head(fruits_ex2)
```

```
## # A tibble: 6 x 3
##   type  size   year
##   <chr> <fct> <dbl>
## 1 apple XS     2010
## 2 apple XS     2011
## 3 apple XS     2012
## 4 apple S      2010
## 5 apple S      2011
## 6 apple S      2012
```

nesting()

Nesting() is a helper that only finds combinations already present in the data. i.e Only combinations that already appear in the data

```
fruits %>% expand(nesting(type))
```

```
## # A tibble: 2 x 1
##   type
##   <chr>
## 1 apple
## 2 orange
```

There are only two combination of fruits in the dataset # nesting() Find all possible combination of fruit type and fruit size

```
fruits %>% expand(nesting(type, size))
```

```
## # A tibble: 4 x 2
##   type   size
##   <chr>  <fct>
## 1 apple  XS
## 2 apple  M
## 3 orange S
## 4 orange M
```

nesting()

Find all possible combination of fruit type, fruit size and year

```
fruits %>% expand(nesting(type, size, year))
```

```
## # A tibble: 5 x 3
##   type   size   year
##   <chr>  <fct> <dbl>
## 1 apple  XS     2010
## 2 apple  M      2012
## 3 orange S      2010
## 4 orange S      2011
## 5 orange M      2012
```

full_seq()

It is used to fill in values of continuous variables.

```
fruits %>% expand(type, size, full_seq(year, 1))
```

```
## # A tibble: 24 x 3
##    type  size  `full_seq(year, 1)`
##    <chr> <fct>                <dbl>
##  1 apple XS                    2010
##  2 apple XS                    2011
##  3 apple XS                    2012
##  4 apple S                     2010
##  5 apple S                     2011
##  6 apple S                     2012
##  7 apple M                     2010
##  8 apple M                     2011
##  9 apple M                     2012
## 10 apple L                     2010
## # ... with 14 more rows
```

full_seq()

It is used to fill in values of continuous variables.

```
fruits %>% expand(type, size, 2010:2013)
```

```
## # A tibble: 32 x 3
##    type  size  `2010:2013`
##    <chr> <fct>       <int>
##  1 apple XS           2010
##  2 apple XS           2011
##  3 apple XS           2012
##  4 apple XS           2013
##  5 apple S            2010
##  6 apple S            2011
##  7 apple S            2012
##  8 apple S            2013
##  9 apple M            2010
## 10 apple M            2011
## # ... with 22 more rows
```

Expand_grid()

When using expand_grid(), it returns a tibble, not a data frame.

```
expand_grid(x = 1:4, y = 1:3)
```

```
## # A tibble: 12 x 2
##        x     y
##    <int> <int>
##  1     1     1
##  2     1     2
##  3     1     3
##  4     2     1
##  5     2     2
##  6     2     3
##  7     3     1
##  8     3     2
##  9     3     3
## 10     4     1
## 11     4     2
## 12     4     3
```

Expand_grid()

When using expand_grid(), it returns a tibble, not a data frame.

```
expand_grid(l1 = letters, l2 = LETTERS)
```

```
## # A tibble: 676 x 2
##    l1    l2
##    <chr> <chr>
##  1 a     A
##  2 a     B
##  3 a     C
##  4 a     D
##  5 a     E
##  6 a     F
##  7 a     G
##  8 a     H
##  9 a     I
## 10 a     J
## # ... with 666 more rows
```

This is the end of the session