R-Lab: Working with Data Frames

STA 321 Topics in Advanced Statistics

West Chester University

Contents

1 Intorduction

In this note, we continue to introduce a few more base R functions that are also commonly used in data management.

2 with() and within() Functions

with() and **within()** are two closely related yet different base R functions that are useful in data management.

2.1 The with() Function

`with()` function enables us to define a new variable based on the variables in a **data frame** using basic **R expressions** that include mathematical and logical operations. We can add the newly defined variables to the existing data frame as usual.

with() Syntax

```
with(data-frame, R-expression)
```

Example 1

```
Num <- c(1400,1200,1100,1700,1500)
Cost <- c(1200,1300,1400,1500,1600)
##
dataA <- data.frame(Num,Cost,stringsAsFactors = FALSE)
##
product <- with(dataA, Num*Cost)
quotient <- with(dataA, Cost/Num)
logical <- with(dataA, Num > Cost)
pander(cbind(product = product, quotient = quotient, logical = logical))
```

product	quotient	logical
1680000	0.8571	1
1560000	1.083	0
1540000	1.273	0
2550000	0.8824	1
2400000	1.067	0

```
## add the new variables to data frame dataA
dataA$product = product
dataA$quotient = quotient
dataA$logical = logical
##
pander(dataA)
```

Num	Cost	product	quotient	logical
1400	1200	1680000	0.8571	TRUE
1200	1300	1560000	1.083	FALSE
1100	1400	1540000	1.273	FALSE
1700	1500	2550000	0.8824	TRUE
1500	1600	2400000	1.067	FALSE

## The **within()** Function

within() function allows us to create a copy of the data frame and add a column that would eventually store the result of the R expression.

```
Num <- c(1400,1200,1100,1700,1500)
Cost <- c(1200,1300,1400,1500,1600)
##
dataA <- data.frame(Num,Cost,stringsAsFactors = FALSE)
##
dataB <- within(dataA, Product <- Num*Cost)   # defined Product and added to dataA simultaneously
dataC <- within(dataB, Quotient <- Cost/Num)
dataD <- within(dataC, Logical <- Num > Cost)
pander(dataD)
```

Num	Cost	Product	Quotient	Logical
1400	1200	1680000	0.8571	TRUE

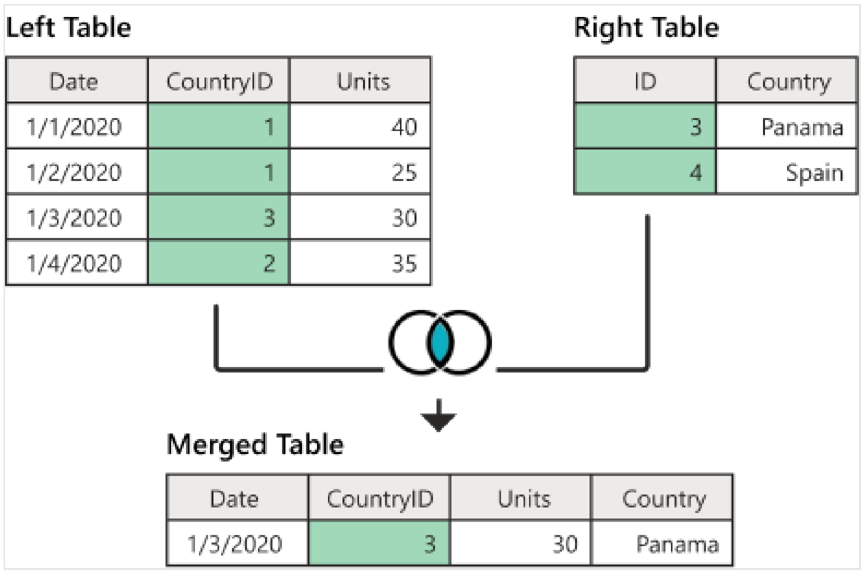
Num	Cost	Product	Quotient	Logical
1200	1300	1560000	1.083	FALSE
1100	1400	1540000	1.273	FALSE
1700	1500	2550000	0.8824	TRUE
1500	1600	2400000	1.067	FALSE

3 The merge() Function - Table Joins

The R **merge()** function allows merging two data frames by **row names** (common key). This function allows us to perform different database (SQL) joins, like left join, inner join, right join, or full join, among others. In this note, we only introduce four different ways of merging datasets in base R with examples. We will introduce the SQL clause in R later.

3.1 Inner Join

The following figure illustrates how A **left joins** B and the resulting merged data set.



Left Table

Date	CountryID	Units
1/1/2020	1	40
1/2/2020	1	25
1/3/2020	3	30
1/4/2020	2	35

Right Table

ID	Country
3	Panama
4	Spain

Merged Table

Date	CountryID	Units	Country
1/3/2020	3	30	Panama

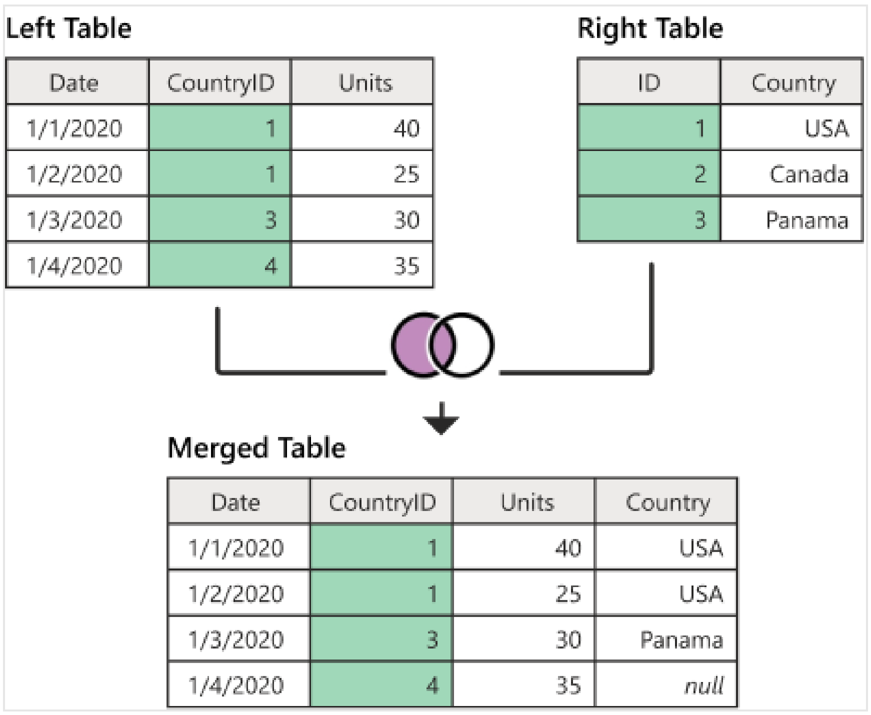
The following code implements the above left-join.

```
A = data.frame(Date = c("1/1/2020", "1/2/2020", "1/3/2020", "1/4/2020"),
               CountryID = c(1,1,3,2),
               Units = c(40, 25, 30, 35))
B = data.frame(ID=c(3,4),
               Country=c( "Panama", "Spain"))
AinnerB = merge(A, B, by.x = "CountryID", by.y = "ID")
pander(AinnerB)
```

CountryID	Date	Units	Country
3	1/3/2020	30	Panama

3.2 Left Join

The following figure illustrates how A **left joins** B and the resulting merged data set.



Left Table

Date	CountryID	Units
1/1/2020	1	40
1/2/2020	1	25
1/3/2020	3	30
1/4/2020	4	35

Right Table

ID	Country
1	USA
2	Canada
3	Panama

Merged Table

Date	CountryID	Units	Country
1/1/2020	1	40	USA
1/2/2020	1	25	USA
1/3/2020	3	30	Panama
1/4/2020	4	35	*null*

The following code implements the above left-join.

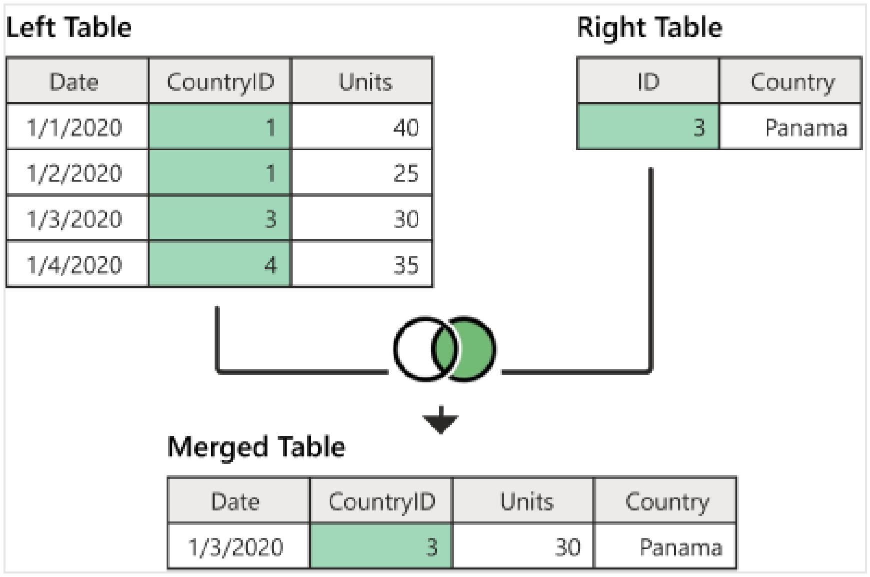
```
A = data.frame(Date = c("1/1/2020", "1/2/2020", "1/3/2020", "1/4/2020"),
               CountryID = c(1,1,3,4),
               Units = c(40, 25, 30, 35))
B = data.frame(ID=c(1,2,3),
               Country=c("USA", "Canada", "Panama"))
AleftB = merge(A, B, by.x = "CountryID", by.y = "ID", all.x = TRUE)
pander(AleftB)
```

CountryID	Date	Units	Country
1	1/1/2020	40	USA
1	1/2/2020	25	USA
3	1/3/2020	30	Panama
4	1/4/2020	35	NA

Note that, left-join produces missing values of the record in A and does not have any information in B.

3.3 Right Join

The following figure illustrates how A **right joins** B and the resulting merged data set.



The following code implements the above left-join.

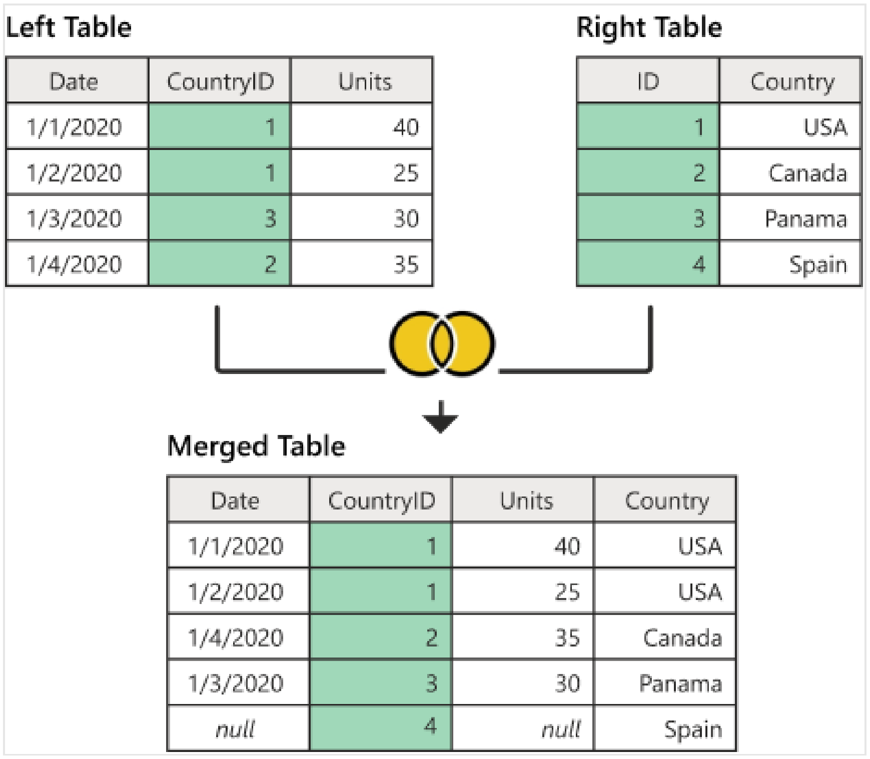
```
A = data.frame(Date = c("1/1/2020", "1/2/2020", "1/3/2020", "1/4/2020"),
               CountryID = c(1,1,3,4),
               Units = c(40, 25, 30, 35))
B = data.frame(ID=c(3),
               Country=c("Panama"))
ArightB = merge(A, B, by.x = "CountryID", by.y = "ID", all.y = TRUE)
pander(ArightB)
```

CountryID	Date	Units	Country
3	1/3/2020	30	Panama

Note also that right-join could also produce missing values.

3.4 Full (outer) Join

The following figure illustrates how A **Full outer joins** B and the resulting merged data set.


Left Table

Date	CountryID	Units
1/1/2020	1	40
1/2/2020	1	25
1/3/2020	3	30
1/4/2020	2	35

Right Table

ID	Country
1	USA
2	Canada
3	Panama
4	Spain

Merged Table

Date	CountryID	Units	Country
1/1/2020	1	40	USA
1/2/2020	1	25	USA
1/4/2020	2	35	Canada
1/3/2020	3	30	Panama
null	4	*null*	Spain

The following code implements the above left-join.

```r
A = data.frame(Date = c("1/1/2020", "1/2/2020", "1/3/2020", "1/4/2020"),
               CountryID = c(1,1,3,2),
               Units = c(40, 25, 30, 35))
B = data.frame(ID=c(1,2,3,4),
               Country=c("USA", "Canada", "Panama", "Spain"))
AfullB = merge(A, B, by.x = "CountryID", by.y = "ID", all = TRUE)
pander(AfullB)
```

CountryID	Date	Units	Country
1	1/1/2020	40	USA
1	1/2/2020	25	USA
2	1/4/2020	35	Canada
3	1/3/2020	30	Panama
4	NA	NA	Spain

4 Subsetting Data Frame

There are two different ways for subsetting a data frame: subsetting by rows and by columns.

We first define the following working data set.

```r
working.data <- data.frame(
  id = c(10,11,12,13,14,15,16,17),
  name = c('sai','ram','deepika','sahithi','kumar','scott','Don','Lin'),
  gender = c('M','M',NA,'F','M','M','M','F'),
  dob = as.Date(c('1990-10-02','1981-3-24','1987-6-14','1985-8-16',
                  '1995-03-02','1991-6-21','1986-3-24','1990-8-26')),
  state = c('CA','NY',NA,NA,'DC','DW','AZ','PH'),
  row.names=c('r1','r2','r3','r4','r5','r6','r7','r8')
)
pander(working.data)
```

	id	name	gender	dob	state
r1	10	sai	M	1990-10-02	CA
r2	11	ram	M	1981-03-24	NY
r3	12	deepika	NA	1987-06-14	NA
r4	13	sahithi	F	1985-08-16	NA
r5	14	kumar	M	1995-03-02	DC
r6	15	scott	M	1991-06-21	DW
r7	16	Don	M	1986-03-24	AZ
r8	17	Lin	F	1990-08-26	PH

4.1 Subsetting by Columns

This is a relatively easy job - we can simply select or drop variables to make a subset. The following is just an example.

4.2 Subsetting by Rows

```
# subset by row name
pander(subset(working.data, subset=rownames(df) == 'r1'))
```

id	**name**	**gender**	**dob**	**state**

```
# subset row by the vector of row names
pander(subset(working.data, rownames(df) %in% c('r1','r2','r3')))
```

id	**name**	**gender**	**dob**	**state**

```
# subset by condition
pander(subset(working.data, gender == 'M'))
```

	id	name	gender	dob	state
r1	10	sai	M	1990-10-02	CA
r2	11	ram	M	1981-03-24	NY
r5	14	kumar	M	1995-03-02	DC
r6	15	scott	M	1991-06-21	DW
r7	16	Don	M	1986-03-24	AZ

```
# subset by condition with %in%
pander(subset(working.data, state %in% c('CA','DC')))
```

	id	name	gender	dob	state
r1	10	sai	M	1990-10-02	CA
r5	14	kumar	M	1995-03-02	DC

```
# subset by multiple conditions using |
pander(subset(working.data, gender == 'M' | state == 'PH'))
```

	id	name	gender	dob	state
r1	10	sai	M	1990-10-02	CA
r2	11	ram	M	1981-03-24	NY
r5	14	kumar	M	1995-03-02	DC
r6	15	scott	M	1991-06-21	DW
r7	16	Don	M	1986-03-24	AZ
r8	17	Lin	F	1990-08-26	PH

5 Create Publication-ready Tables Manually

We create publication-ready tables manually in R markdown. These tables can be used when you want to aggregate information from different outputs.

5.1 Regular text style table.

It is simple and easy to use. However, there is no mechanism to specify cell alignment.

Num	Cost	Product	Quotient	Logical
1400	1200	1680000	0.8571	TRUE
1200	1300	1560000	1.083	FALSE
1100	1400	1540000	1.273	FALSE
1700	1500	2550000	0.8824	TRUE
1500	1600	2400000	1.067	FALSE

5.2 Markdown Table

Markdown style table. colon(:) is used to specify the cell alignment.

Num	Cost	Product	Quotient	Logical
1400	1200	1680000	0.8571	TRUE
1200	1300	1560000	1.083	FALSE
1100	1400	1540000	1.273	FALSE
1700	1500	2550000	0.8824	TRUE
1500	1600	2400000	1.067	FALSE