Session 5: Data Manipulation In R

Agenda

- **Renaming and Recoding Variables**
- **Dealing With Missing Values**
- **Merging & Concatenating Datasets**
- **Ordering Operations**
- **Using SQL Statements To Manipulate Data Frames**
- **Data Type Conversion**
- **Data Values**

Renaming & Recoding Variables

Recoding Variables

- Recoding involves :
- Convert a continuous variable into a Categorical
- Replace miscoded values with correct values
- Create a decision variable (Yes/No) based on a threshold cutoff.
- Logical operators are widely used for recoding :

Operators	Description
<	Less than
<=	Less than or equal to
>	Greater than
>=	Greater than or equal to
==	Exactly equal to
!=	Not equal to
!x	Not x
x | y	x or y
x & y	x and y
isTRUE(x)	Test if x is TRUE

Recoding Variables (contd.)

#Convert a continuous variable age to the categorical variable agecat (Young, Middle Aged, Elder)

```
# Age 99 is assigned NA
company$age[ company$age == 99]        <-    NA
```

```
#Assign Category “Elder”
company$agecat[ company$age      > 75]     <-    "Elder“
#Assign Category “Middle Aged”
company$agecat[company$age     >= 55 &
                          company$age     <= 75]     <-    "Middle Aged“
#Assign Category “Young”
company$agecat[company$age      < 55]      <-    "Young"
```

Recoding Variables(contd.)

#Using within() Function

```
company      <- within(company,{
                              agecat <- NA
                              agecat[age > 75]                  <- "Elder"
                              agecat[age >= 55 & age <= 75]     <- "Middle Aged"
                              agecat[age < 55]                  <- "Young" })
```

Renaming Variables

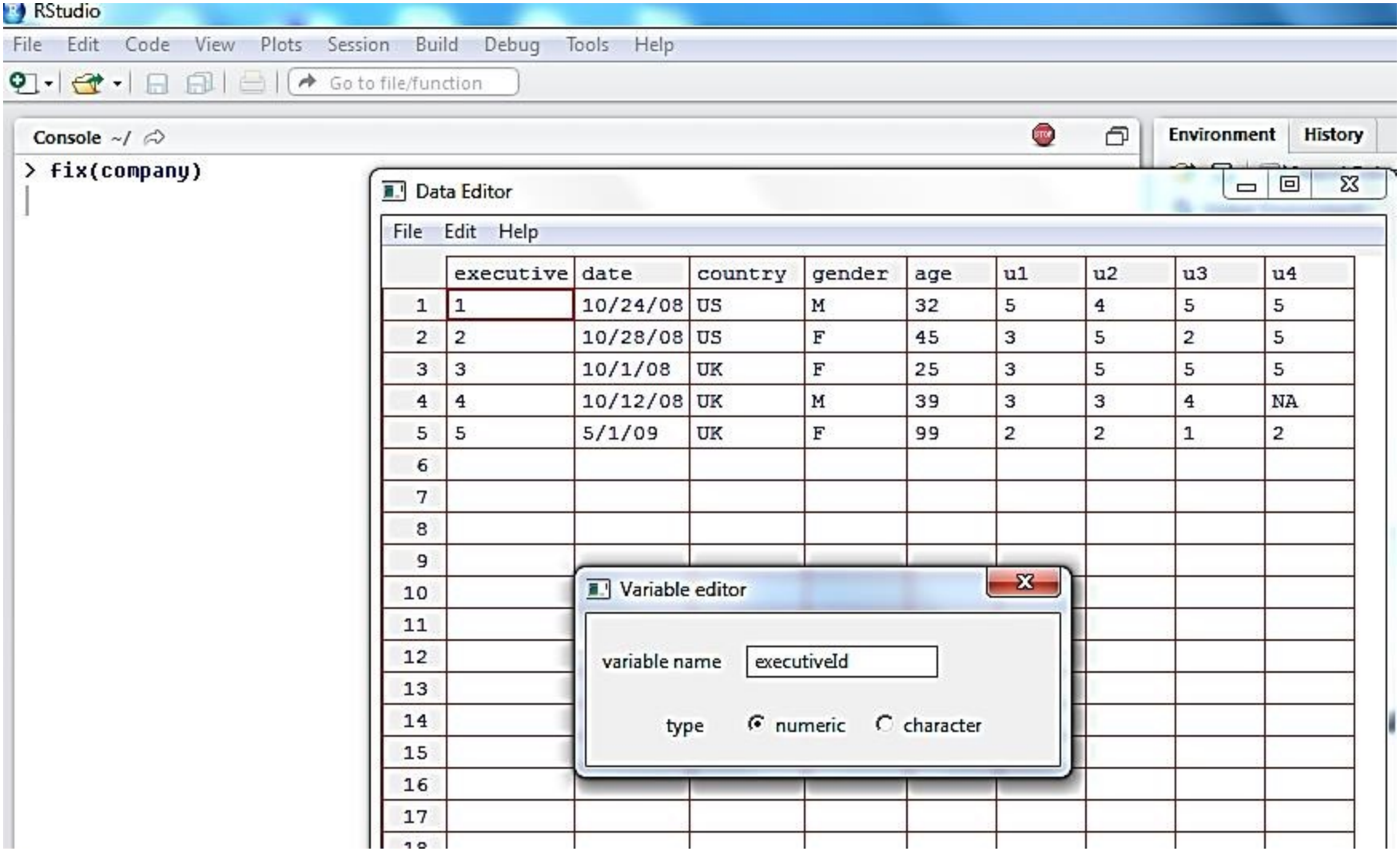
The **reshape package** has a rename ( ) function that's useful for altering the names of variables.

```
rename(dataframe, c(oldname="newname", oldname="newname",…))
```

Illustration :

```
library(reshape)
company <- rename(company, c(executive="executiveID", date="testDate"))
```

Renaming Variables(contd.)



Renaming Variables(contd.)

Using names() function :

```
> names(company)
  [1] "executive"    "date"    "country"   "gender"     "age“    “u1"   “u2"
  [8] “u3"       “u4"       “u5"
> # To rename date to testDate
names(company)[2]   <- "testDate"
> company
  executive  test  Date   country   Gender    Age     u1     u2     u3     u4     u5
1     1          10/24/08      US          M        32      5      4      5      5      5
2     2          10/28/08      US          F        45      3      5      2      5      5
3     3          10/01/08      UK          F        25      3      5      5      5      2
4     4          10/12/08      UK          M        39      3      3      4     NA     NA
5     5          05/01/09      UK          F        99      2      2      1      2      1
```

Other way :

```
names(company)[6 : 10] <- paste(“TEST",  1:5,  sep="")

names(company)[6 : 10]  <- c(“TEST1" , “TEST2",  “TEST3",  “TEST4",  “TEST5")
```

Dealing With Missing Values

Missing Values

In R :

Missing values can be referred as **NA or NaN or NULL**

There is one more type of value which is Inf or Infinite

Following is the explanation for all the above:

- In statistical data sets, we often encounter missing data, which are represented with **NA** in R. The motivation of NA, meaning '**N**ot **A**vailable', is to handle the case where specifications to an operation is not complete.
- **NaN**, meaning '**N**ot **A** **N**umber', is another kind of 'missing' that is produced by numerical computation when the result cannot be defined didn't make mathematical sense or could not be performed properly.
- **NULL** represents that the value in question simply *does not exist*, rather than being existent but unknown.
- **Inf** and **-Inf** represent positive and negative infinities, respectively, resulting numerical calculations

```
> c(1/0, -1/0)
[1] Inf -Inf

> c(Inf+1, Inf-1, Inf-Inf, 1/Inf, Inf/100, Inf/Inf)
[1] Inf Inf NaN 0 Inf NaN

> c(sqrt(-1), sqrt(Inf), sin(sqrt(Inf)))
[1] NaN Inf NaN
Warning messages:
1: In sqrt(-1) : NaNs produced
2: In sin(sqrt(Inf)) : NaNs produced
```

Understanding NA vs NaN vs NULL vs Inf

```
> c(log(Inf), log(0), exp(Inf), exp(-Inf))
[1] Inf -Inf Inf   0

> x <- c(88,NA,12,168,13)
> mean(x)
[1] NA
> mean(x,na.rm=TRUE)
[1] 70.25

> x <- c(88,NaN,12,168,13)
> mean(x)
[1] NaN
> x <- c(88,NULL,12,168,13)
> mean(x)
[1] 70.25
```

As can be seen, R automatically skip over NULL in computation, but not skip over on NA and NaN

Understanding NA vs NaN vs NULL vs Inf

```
> x <- c(0, Inf, -Inf, NaN, NA)
> is.finite(x)
[1]  TRUE FALSE FALSE FALSE FALSE
> is.infinite(x)
[1] FALSE  TRUE  TRUE FALSE FALSE
> is.nan(x)
[1] FALSE FALSE FALSE  TRUE FALSE
> is.na(x)
[1] FALSE FALSE FALSE  TRUE  TRUE
```

Also, note that *is.na()* returns TRUE both for NA and NaN values, while *is.nan()* returns TRUE only for NaNs.

```
R
> x<-c(NA,NaN)
> is.na(x)
[1] TRUE TRUE
>is.nan(x)
[1] FALSE TRUE
1
2
3
4
5
> x<-c(NA,NaN)
> is.na(x)
[1] TRUE TRUE
> is.nan(x)
[1] FALSE  TRUE
```

Recoding Values To Missing

Recode values to missing.

```
# Value 99 means missing in age
#  Recoding the variable:


company$age[company$age == 99]  <-  NA
```

Excluding Missing Values From Analyses

```
#Excluding Missing Values from Analysis using complete.cases

newdata <- company[complete.cases(company), ]

# na.omit() to omit all rows that contain NA values:

newdata = na.omit(company)
```

	executive	date	country	gender	age	Test1	Test2	Test3	Test4	Test5
1	1	10/24/08	US	M	32	5	4	5	5	5
2	2	10/28/08	US	F	45	3	5	2	5	5
3	3	10/1/08	UK	F	25	3	5	5	5	2
5	5	5/1/09	UK	F	99	2	2	1	2	1

Merging & Concatenating Datasets

Merging Datasets

Adding columns

To merge two data frames (datasets) horizontally, we can use the merge() function by one or more common key variables (that is an inner join). For example:

```
total <- merge(dataframeA, dataframeB, by="ID")
```

merges dataframeA and dataframeB by ID. Similarly,

```
total <- merge(dataframeA, dataframeB, by=c("ID","Country"))
```

NOTE If you're joining two matrices or data frames horizontally and don't want to specify a common key, you can use the cbind() function :

```
total <- cbind(A, B)
```

Understanding Merge

- R provide us to **MERGE** function to join two data tables. We can do inner join, outer joins, full join and cartesian product through merge. Its syntax is
- **merge(DT1, DT2, by=[by], all = FALSE, all.x = all, all.y = all,sort = TRUE….)**
- **DT1, DT2** – data tables to be used for merging
- **by** – specifications of the columns used for merging. We can specify multiple columns also like c(“Col1”, “Col2”).
- **all** – logical; Should be either TRUE or FALSE.
- **all.x** – logical; if TRUE, it works like an left join. The default is FALSE, it works like inner join.
- **all.y** – logical; if TRUE, it works like an right join. The default is FALSE, it works like inner join.
- **sort –** logical. Default is True.
- Lets try to understand with the help of example. Suppose, we have two data frame
- > stu=data.table(roll_no=c(3,1,2,5,4),
names=c(“Peter”,”Jack”,”David”,”James”,”John”))
  > marks=data.table(roll_no=c(4,2,3,6,1), Maths=c(89,92,76,67,90),

Merging Datasets

Type of Join	R Syntax
Inner join	merge(dataframeA, dataframeB, by="common_key_column/s")
Outer join	merge(dataframeA, dataframeB, by="common_key_column/s", all=TRUE)
Left outer	merge(dataframeA, dataframeB, by="common_key_column/s", all.x=TRUE)
Right outer	merge(dataframeA, dataframeB, by="common_key_column/s",

Inner Join

- > x=merge(stu, marks, by="roll_no", all=F, sort=F)
  > x=merge(stu, marks, by="roll_no", all.x=F, sort=F)
  > x=merge(stu, marks, by="roll_no", all.y=F, sort=F)

	roll_no	names	Maths	Science
1	3	Peter	76	88
2	1	Jack	90	92
3	2	David	92	92
4	4	John	89	98

- Output of above three statements are same and all are work like inner join. Its SQL equivalent are "Select s.roll_no, names, Maths, Science from stu s inner join marks m on s.roll_no = m.roll"

Left Join

- >merge(stu, marks, by="roll_no", all.x=T)

	roll_no	names	Maths	Science
1	1	Jack	90	92
2	2	David	92	92
3	3	Peter	76	88
4	4	John	89	98
5	5	James	NA	NA

- In this case, all.x=TRUE, that means include entire data of table exists on the **left** side and coerced NA for non matching columns of table exists on the **right** hand side.
- Its SQL equivalent are "**Select s.roll_no, names, Maths, Science from stu s left join marks m on s.roll_no = m.roll_no**"

Right Join

- > merge(stu, marks, by="roll_no", all.y=T)

	roll_no	names	Maths	Science
1	1	Jack	90	92
2	2	David	92	92
3	3	Peter	76	88
4	4	John	89	98
5	5	James	NA	NA
6	6	NA	67	91

- In this case, all.y=TRUE, that means include entire data of table exists on the **right** side and coerced NA for non matching columns of table exists on the **left** hand side.
- Its SQL equivalent are "**Select s.roll_no, names, Maths, Science from stu s right join marks m on stu.roll_no = marks.roll_no**" **Full join**

Full Join

- > merge(stu, marks, by=”roll_no”, all=T)
- It includes entire data of both the tables and coerced NA where data does not exist.
- Its SQL equivalent are “**Select s.roll_no, names, Maths, Science from stu s full join marks m on s.roll_no = m.roll_no**”

	roll_no	names	Maths	Science
1:	1	Jack	90	92
2:	2	David	92	92
3:	3	Peter	76	88
4:	4	John	89	98
5:	5	James	NA	NA
6:	6	NA	67	91

Merging Datasets

Adding rows

To join two data frames (datasets) vertically, we can use the rbind() function

:

```
total <- rbind(dataframeA, dataframeB)
```

The two data frames :

- must have the same variables

Ordering Operations

Ordering Operations using dplyr

- By default R interprets from inside to out, for e.g. in mtcars data:
- Select specific columns, aggregate the miles per gallon and weight by the number of cylinders and gear, and filter so that the rows selected have an average miles per gallon greater than 15 :
- Syntax :

```
filter( summarise( select( group_by(mtcars, cyl, gear), mpg, cyl, wt, gear, am), avgmpg =
mean(mpg), avgwt = mean(wt)), avgmpg > 15)
```

- The chain function and %>% operator allows the user to write the functions in the order they will be processed by R :

```
Install.packages(dplyr)
library(dplyr)
mtcars %>% group_by(cyl, gear) %>%
select(mpg, cyl, wt, gear, am) %>%
summarise(avgmpg = mean(mpg), avgwt = mean(wt)) %>%
filter(avgmpg > 15)
```

Using SQL Statements To Manipulate Data Frames

Ordering and Limiting

- Sorting (descending order) and limiting output from an SQL select statement on the iris data frame in R.

```
> library(sqldf)
> sqldf('select * from iris order by "Sepal.Length"
desc limit 3')

  Sepal.Length Sepal.Width Petal.Length
Petal.Width Species
1          7.9         3.8          6.4
2.0 virginica
2          7.7         3.8          6.7
2.2 virginica
3          7.7         2.6          6.9
2.3 virginica
```

Per Group Max , Min and Count

- # Count of Sepal Length , Mx and Min Sepal Length per group of species
 sqldf("select Species ,max('Sepal.Length') HighLength, min('Sepal.Length') LowLength,count('Sepal.Length') as Cnt from iris group by Species")

Sl No	Species	HighLength	LowLength	Count
1	Setosa	5.8	4.3	50
2	Versicolor	7.0	4.9	50
3	Virginica	7.9	4.9	50

Join

- #Defined a newdataframe Aka , joined with iris and performed aggregation
- # with join and using keywords
- > Aka <- data.frame(Species = levels(iris$Species),
- Aka = c("S", "Ve", "Vi"))
-
- > sqldf('select Aka, avg("Sepal.Length") from iris join Aka using(Species) group by Species')

Sl No	Aka	Avg(Sepal.Length)
1	S	5.006
2	Ve	5.936
3	Vi	6.588

Join

- # With a where clause
- > sqldf('select Aka, avg("Sepal.Length") from iris, Aka where iris.Species = Aka.Species group by iris.Species')

Sl No	Aka	Avg(Sepal.Length)
1	S	5.006
2	Ve	5.936
3	Vi	6.588

Nested Select

- #For each Species, find the average Sepal Length among those rows where Sepal Length exceeds the average Sepal Length for that Species.
- sqldf("select iris.Species as Species]', avg(Sepal_Length) as Avg_of_SLs _GT_avg SL
  from iris, (select Species, avg(Sepal_Length) SLavg from iris group by Species) Slavg where iris.Species = SLavg.Species and Sepal_Length > SLavg group by iris.Species")

Sl No	Species	Avg_of_SLs_GT_avgSL
1	Sestosa	5.313636
2	Versicolor	6.375000
3	Virginica	7.159091

Nested Select

- # For each species identify the two rows with the largest sepal lengths
- > sqldf('select * from iris i where rowid in (select rowid from iris where Species = i.Species order by "Sepal_Length" desc limit 2) order by i.Species, i."Sepal_Length" desc')

Sl No	Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
1	5.8	4.0	1.2	0.2	Setosa
2	5.7	4.4	1.5	0.4	Setosa
3	7.0	3.2	4.7	1.4	Versicolor
4	6.9	3.1	4.9	1.5	Versicolor
5	7.9	3.8	6.4	2.0	Virginica
6	7.7	3.8	6.7	2.2	Virginica

Data Type Conversion

Data Type Conversion

Type conversions in R work as you would expect. For example, adding a character string to a numeric vector converts all the elements in the vector to character.

- Use **is.***foo* to test for data type *foo*. Returns TRUE or FALSE
  for e.g. : **is.numeric(), is.character(), is.vector(), is.matrix(), is.data.frame()**
- Use **as.***foo* to explicitly convert it.
  for e.g. **: as.numeric(), as.character(), as.vector(), as.matrix(), as.data.frame)**

	to one long vector	To matrix	To data frame
From vector	c (x, y)	cbind (x, y) rbind (x, y)	data.frame(x,y)
From matrix	as.vector(mymatrix)		as.data.frame(mymatrix)
From data frame		as.matrix(myframe)	

Date Values

Date Values

Dates are represented as the number of days since 1970-01-01, with negative values for earlier dates.

```
# use as.Date( ) to convert strings to dates
mydates <- as.Date(c("2007-06-22", "2004-02-13"))
# number of days between 6/22/07 and 2/13/04
days <- mydates[1] - mydates[2]
```

Sys.Date( ) returns today's date.
date() returns the current date and time.

The following symbols can be used with the **format( )** function to print dates.

Symbol	Meaning	Example
%d	day as a number (0-31)	01-31
%a	abbreviated weekday	Mon
%A	unabbreviated weekday	Monday
%m	month (00-12)	00-12
%b	abbreviated month	Jan
%B	unabbreviated month	January
%y	2-digit year	07
%Y	4-digit year	2007

Date Values(contd.)

Here is an example

```
# print today's date
today <- Sys.Date()
format(today, format="%B %d %Y")
"June 20 2007"
```

Date Conversion

Character to Date

Use the **as.Date( )** function to convert character data to dates.
The format is **as.Date(***x***,***"format"***)**, where *x* is the character data and *format* gives the appropriate format.

```
# convert date info in format 'mm/dd/yyyy'
strDates <- c("01/05/1965", "08/16/1975")
dates <- as.Date(strDates, "%m/%d/%Y")
```

The default format is **yyyy-mm-dd**

```
mydates <- as.Date(c("2007-06-22", "2004-02-13"))
```

Date to Character

To convert dates to character data using the **as.Character( )** function.

```
# convert dates to character data
strDates <- as.character(dates)
```

Next class – Writing Functions In R

- Control & Flow Operators
- Make A Script In R
- Writing Functions In R
- Simple Functions In R
- Complex Functions In R
- Creating R package
- Exercise