Supplement: dplyr and tidyr

BIOSTAT 213: Introduction to Computing in the R Software Environment

Yea-Hung Chen, PhD, MS

dplyr

Introduction

dplyr

- the dplyr package provides functions for manipulating and summarizing data
- many of the package's functions reproduce base R capabilities
- but, the different framework make the package very powerful

dplyr

- to install the package (necessary just once ever):

```
> install.packages('dplyr')
```

dplyr

- to load the package (necessary just once per session):

```
> library(dplyr)
```

dplyr

- resources:
  - introduction
  - cheat sheet
  - documentation

dplyr

- for simplicity of presentation on these slides, we'll use a subset of the NHANES data throughout this section:

```
> ss<-ii[5:10,c('ridageyr','bpcat','bmxht','hs','military')]
```

- to save space, I've also renamed the `History of smoking` level in `hs`:

```
> levels(ss$hs)
[1] "History of smoking" "No"
> levels(ss$hs)<-c('Smoking','No')
> levels(ss$hs)
[1] "Smoking" "No"
```

dplyr

```
> ss
   ridageyr         bpcat bmxht      hs military
5        22        Normal 165.4 Smoking       No
6        32     Elevated 151.3      No       No
7        56 Hypertension 179.4      No       No
8        46 Hypertension 176.7 Smoking       No
9        45        Normal 177.8 Smoking       No
10       30        Normal 163.6 Smoking       No
```

filter()

filter()

- to subset rows, use `filter()`:

```
> filter(ss,ridageyr==56)
  ridageyr         bpcat bmxht hs military
1       56 Hypertension 179.4 No       No
```

- the first argument is the data
- the second argument is the selection rule

filter()

- equivalently, using `subset()`:

```
> subset(ss,ridageyr==56)
  ridageyr        bpcat bmxht hs military
7       56 Hypertension 179.4 No       No
```

filter()

- equivalently, using indexing:

```
> ss[ss$ridageyr==56,]
  ridageyr        bpcat bmxht hs military
7       56 Hypertension 179.4 No       No
```

filter()

- for more than one rule, use commas:

```
> filter(ss,ridageyr==56,military=='No')
  ridageyr        bpcat bmxht hs military
1       56 Hypertension 179.4 No       No
```

filter()

- equivalently, using `subset()`:

```
> subset(ss,ridageyr==56&military=='No')
  ridageyr        bpcat bmxht hs military
7       56 Hypertension 179.4 No       No
```

filter()

- equivalently, using indexing:

```
> ss[ss$ridageyr==56&ss$military=='No',]
  ridageyr        bpcat bmxht hs military
7       56 Hypertension 179.4 No       No
```

filter()

- example involving a variable that contains missing values (bmxht):

```
> filter(ss,bmxht<=160)
  ridageyr    bpcat bmxht hs military
1       32 Elevated 151.3 No       No
```

filter()

- equivalently, using `subset()`:

```
> subset(ss,bmxht<=160)
  ridageyr    bpcat bmxht hs military
6       32 Elevated 151.3 No       No
```

filter()

- equivalently, using indexing:

```
> ss[!is.na(ss$bmxht)&ss$bmxht<=160,]
  ridageyr    bpcat bmxht hs military
6       32 Elevated 151.3 No       No
```

- as previously discussed, with indexing, it is necessary to include `!is.na()` if the indexing variable has missing values

arrange()

arrange()

- to sort data, use arrange():

```
> arrange(ss,ridageyr)
  ridageyr         bpcat bmxht       hs military
1       22        Normal 165.4 Smoking       No
2       30        Normal 163.6 Smoking       No
3       32      Elevated 151.3       No       No
4       45        Normal 177.8 Smoking       No
5       46  Hypertension 176.7 Smoking       No
6       56  Hypertension 179.4       No       No
```

- the first argument is the data
- the second argument is the sorting variable

arrange()

- equivalently, using indexing and `order()`:

```
> ss[order(ss$ridageyr),]
   ridageyr        bpcat bmxht      hs military
5        22       Normal 165.4 Smoking       No
10       30       Normal 163.6 Smoking       No
6        32     Elevated 151.3       No       No
9        45       Normal 177.8 Smoking       No
8        46 Hypertension 176.7 Smoking       No
7        56 Hypertension 179.4       No       No
```

arrange()

- to sort by more than one variable, add commas:

```
> arrange(ss,ridageyr,bpcat)
  ridageyr         bpcat bmxht      hs military
1       22        Normal 165.4 Smoking       No
2       30        Normal 163.6 Smoking       No
3       32     Elevated 151.3       No       No
4       45        Normal 177.8 Smoking       No
5       46 Hypertension 176.7 Smoking       No
6       56 Hypertension 179.4       No       No
```

arrange()

- equivalently, using indexing and `order()`:

```
> ss[order(ss$ridageyr,ss$bpcat),]
   ridageyr         bpcat bmxht      hs military
5        22        Normal 165.4 Smoking       No
10       30        Normal 163.6 Smoking       No
6        32      Elevated 151.3       No       No
9        45        Normal 177.8 Smoking       No
8        46  Hypertension 176.7 Smoking       No
7        56  Hypertension 179.4       No       No
```

arrange()

- to sort some—but not all—variables in descending order, add `desc()`:

```
> arrange(ss,desc(ridageyr),bpcat)
  ridageyr         bpcat bmxht      hs military
1       56 Hypertension 179.4      No       No
2       46 Hypertension 176.7 Smoking       No
3       45       Normal 177.8 Smoking       No
4       32     Elevated 151.3      No       No
5       30       Normal 163.6 Smoking       No
6       22       Normal 165.4 Smoking       No
```

arrange()

- this would be trickier to do using indexing and `order()`

select()

select()

- to subset columns, use `select()`:

```
> select(ss,bpcat,bmxht)
          bpcat bmxht
5        Normal 165.4
6      Elevated 151.3
7  Hypertension 179.4
8  Hypertension 176.7
9        Normal 177.8
10       Normal 163.6
```

- the first argument is the data
- the other arguments are the columns to select

select()

- equivalently, using `subset()`:

```
> subset(ss,select=c(bpcat,bmxht))
          bpcat bmxht
5        Normal 165.4
6      Elevated 151.3
7  Hypertension 179.4
8  Hypertension 176.7
9        Normal 177.8
10       Normal 163.6
```

select()

- equivalently, using indexing:

```
> ss[,c('bpcat','bmxht')]
          bpcat bmxht
5        Normal 165.4
6      Elevated 151.3
7  Hypertension 179.4
8  Hypertension 176.7
9        Normal 177.8
10       Normal 163.6
```

mutate()

mutate()

- to add columns, use mutate():

```
> ss<-mutate(ss,inches=bmxht*0.393701)
> ss
  ridageyr         bpcat bmxht      hs military   inches
1       22        Normal 165.4 Smoking       No 65.11815
2       32     Elevated 151.3      No       No 59.56696
3       56 Hypertension 179.4      No       No 70.62996
4       46 Hypertension 176.7 Smoking       No 69.56697
5       45        Normal 177.8 Smoking       No 70.00004
6       30        Normal 163.6 Smoking       No 64.40948
```

- the first argument is the data
- the second argument is the definition of the new variable

mutate()

- equivalently, using object assignment:

```
> ss$inches<-ss$bmxht*0.393701
> ss
  ridageyr         bpcat bmxht      hs military   inches
1       22        Normal 165.4 Smoking       No 65.11815
2       32     Elevated 151.3      No       No 59.56696
3       56 Hypertension 179.4      No       No 70.62996
4       46 Hypertension 176.7 Smoking       No 69.56697
5       45        Normal 177.8 Smoking       No 70.00004
6       30        Normal 163.6 Smoking       No 64.40948
```

summarize()

summarize()

- for summaries, use `summarize()`:

```
> summarize(ss,
+           age=mean(ridageyr),
+           height=mean(inches,na.rm=TRUE))
  age   height
1 38.5 66.54859
```

- the first argument is the data
- the other arguments define the summaries

summarize()

- equivalently, using `apply()`:

```
> apply(ss[,c('ridageyr','inches')],2,mean,na.rm=TRUE)
ridageyr   inches
38.50000 66.54859
```

summarize()

- using different types of summaries:

```
> summarize(ss,
+           mean.age=mean(ridageyr),
+           median.height=median(inches,na.rm=TRUE))
  mean.age median.height
1     38.5       67.34256
```

summarize()

- using base R:

```
> c(mean.age=mean(ss$ridageyr),
+   median.height=median(ss$inches,na.rm=TRUE))
     mean.age median.height
     38.50000      67.34256
```

Piping

Piping

- one of the unique features of dplyr is its modular nature

Piping

- for example, combining `filter()` and `bpcat()`:

```
> ss %>%
+   filter(bpcat=='Elevated') %>%
+   select(bpcat,military,ridageyr)
     bpcat military ridageyr
1 Elevated       No       32
```

- note that `filter()` and `select()` no longer specify the data
- this feature is known as piping

Piping

- equivalently, using indexing:

```
> ss[!is.na(ss$bpcat)&ss$bpcat=='Elevated',
+    c('bpcat','military','ridageyr')]
     bpcat military ridageyr
2 Elevated       No       32
```

group_by()

group_by()

- to conduct stratified analysis, use group_by():

```
> ss %>%
+   group_by(hs) %>%
+   summarize(mean.height=mean(inches,na.rm=TRUE),
+             mean.age=mean(ridageyr))
# A tibble: 2 x 3
  hs      mean.height mean.age
  <fct>         <dbl>    <dbl>
1 Smoking        67.3     35.8
2 No             65.1     44
```

- the argument to group_by() indicates the stratifying variable

group_by()

- a tibble is an object type specific to the so-called tidyverse (a set of affiliated packages that includes ggplot2, dplyr, tidyr, and others)

group_by()

- equivalently, using `apply()` and `tapply()`:

```
> apply(ss[c('inches','ridageyr')],2,
+       FUN=function(cc){tapply(cc,ss$hs,mean,na.rm=TRUE)})
          inches ridageyr
Smoking 67.27366    35.75
No      65.09846    44.00
```

group_by()

- equivalently, using `aggregate()` (not discussed in class):

```
> aggregate(cbind(inches,ridageyr)~hs,data=ss,FUN=mean)
       hs   inches ridageyr
1 Smoking 67.27366    35.75
2      No 65.09846    44.00
```

group_by()

- using different types of summaries:

```
> ss %>%
+   group_by(hs) %>%
+   summarize(median.height=mean(inches,na.rm=TRUE),
+             mean.age=mean(ridageyr))
# A tibble: 2 x 3
  hs      median.height mean.age
  <fct>           <dbl>    <dbl>
1 Smoking          67.3     35.8
2 No               65.1     44
```

group_by()

- this would be trickier to do using base R

tidyr

tidyr

- the tidyr package provides functions for reshaping data

tidyr

- to install the package (necessary just once ever):

```
> install.packages('tidyr')
```

tidyr

- to load the package (necessary just once per session):

```
> library(tidyr)
```

tidyr

- resources:
  - cheat sheet
  - documentation

Reshaping data from wide to long

- to reshape data from wide to long, use `gather()`:

```
> long.gather<-gather(select(ii,seqn,ridageyr,bpxdi1,bpxdi2,
+                            bpxdi3,bpxdi4),
+                     key='order',
+                     value='bpxdi',
+                     -seqn,-ridageyr)
```

- the first argument is the data
- `key`: the name for the new key variable
- `value`: the name for the value variable
- the arguments `-seqn` and `-ridageyr` indicate that `seqn` and `ridageyr` are *not* repeated measures

Reshaping data from wide to long

- checking work:

```
> head(long.gather[order(long.gather$seqn),])
       seqn ridageyr  order bpxdi
1     83732       62 bpxdi1    70
4210  83732       62 bpxdi2    64
8419  83732       62 bpxdi3    62
12628 83732       62 bpxdi4    NA
2     83733       53 bpxdi1    88
4211  83733       53 bpxdi2    88
```

Reshaping data from wide to long

- equivalently, using `reshape()`:

```
> kk<-c('seqn','ridageyr','bpxdi1','bpxdi2','bpxdi3','bpxdi4')
> long.reshape<-reshape(data=ii[,kk],
+                       varying=kk[3:6],
+                       sep='',direction='long',
+                       idvar='seqn',timevar='order')
```

Reshaping data from wide to long

- checking work:

```
> head(long.reshape[order(long.reshape$seqn),])
         seqn ridageyr order bpxdi
83732.1 83732       62     1    70
83732.2 83732       62     2    64
83732.3 83732       62     3    62
83732.4 83732       62     4    NA
83733.1 83733       53     1    88
83733.2 83733       53     2    88
```

Reshaping data from wide to long

- I prefer `reshape()` because it is able to extract numbers from variable names