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')]
```

dplyr

```
> ss
   ridageyr    bpcat bmxht      hs military
5        56  Stage 2 158.7 Smoking Military
6        61  Stage 1 161.8      No       No
7        42     <NA>    NA Smoking       No
8        56 Elevated 152.8 Smoking       No
9        65  Stage 2 172.4 Smoking       No
10       26   Normal 152.5      No       No
```

filter()

filter()

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

```
> filter(ss,ridageyr==56)
  ridageyr     bpcat bmxht       hs military
1       56  Stage 2 158.7 Smoking Military
2       56 Elevated 152.8 Smoking       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
5       56  Stage 2 158.7 Smoking Military
8       56 Elevated 152.8 Smoking       No
```

filter()

- equivalently, using indexing:

```
> ss[ss$ridageyr==56,]
  ridageyr     bpcat bmxht      hs military
5       56  Stage 2 158.7 Smoking Military
8       56 Elevated 152.8 Smoking       No
```

filter()

- for more than one rule, use commas:

```
> filter(ss,ridageyr==56,military=='Military')
  ridageyr   bpcat bmxht      hs military
1       56 Stage 2 158.7 Smoking Military
```

filter()

- equivalently, using `subset()`:

```
> subset(ss,ridageyr==56&military=='Military')
  ridageyr    bpcat bmxht      hs military
5       56 Stage 2 158.7 Smoking Military
```

filter()

- equivalently, using indexing:

```
> ss[ss$ridageyr==56&ss$military=='Military',]
  ridageyr   bpcat bmxht      hs military
5       56 Stage 2 158.7 Smoking Military
```

filter()

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

```
> filter(ss,bmxht<=160)
  ridageyr     bpcat bmxht       hs military
1       56  Stage 2 158.7 Smoking Military
2       56 Elevated 152.8 Smoking       No
3       26   Normal 152.5       No       No
```

filter()

- equivalently, using `subset()`:

```
> subset(ss,bmxht<=160)
   ridageyr    bpcat bmxht      hs military
5        56  Stage 2 158.7 Smoking Military
8        56 Elevated 152.8 Smoking       No
10       26   Normal 152.5      No       No
```

filter()

- equivalently, using indexing:

```
> ss[!is.na(ss$bmxht)&ss$bmxht<=160,]
   ridageyr    bpcat bmxht      hs military
5        56  Stage 2 158.7 Smoking Military
8        56 Elevated 152.8 Smoking       No
10       26   Normal 152.5      No       No
```

- as previously discussed, with indexing, it is necessary to include `!is.na()`

arrange()

arrange()

- to sort data, use `arrange()`:

```
> arrange(ss,ridageyr)
  ridageyr     bpcat bmxht      hs military
1       26    Normal 152.5      No       No
2       42      <NA>    NA Smoking       No
3       56   Stage 2 158.7 Smoking Military
4       56 Elevated 152.8 Smoking       No
5       61   Stage 1 161.8      No       No
6       65   Stage 2 172.4 Smoking       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
10       26    Normal 152.5      No       No
7        42      <NA>    NA Smoking       No
5        56   Stage 2 158.7 Smoking Military
8        56 Elevated 152.8 Smoking       No
6        61   Stage 1 161.8      No       No
9        65   Stage 2 172.4 Smoking       No
```

arrange()

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

```
> arrange(ss,ridageyr,bpcat)
  ridageyr     bpcat bmxht      hs military
1       26    Normal 152.5      No       No
2       42      <NA>    NA Smoking       No
3       56 Elevated 152.8 Smoking       No
4       56  Stage 2 158.7 Smoking Military
5       61  Stage 1 161.8      No       No
6       65  Stage 2 172.4 Smoking       No
```

arrange()

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

```
> ss[order(ss$ridageyr,ss$bpcat),]
   ridageyr    bpcat bmxht      hs military
10       26   Normal 152.5      No       No
7        42     <NA>    NA Smoking       No
8        56 Elevated 152.8 Smoking       No
5        56  Stage 2 158.7 Smoking Military
6        61  Stage 1 161.8      No       No
9        65  Stage 2 172.4 Smoking       No
```

arrange()

- to sort some variables in descending order, add `desc()`:

```
> arrange(ss,desc(ridageyr),bpcat)
  ridageyr     bpcat bmxht       hs military
1       65  Stage 2 172.4 Smoking       No
2       61  Stage 1 161.8       No       No
3       56 Elevated 152.8 Smoking       No
4       56  Stage 2 158.7 Smoking Military
5       42      <NA>    NA Smoking       No
6       26    Normal 152.5       No       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   Stage 2 158.7
6   Stage 1 161.8
7      <NA>    NA
8  Elevated 152.8
9   Stage 2 172.4
10   Normal 152.5
```

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

select()

- equivalently, using `subset()` and its previously *not* discussed `select` argument:

```
> subset(ss,select=c(bpcat,bmxht))
      bpcat bmxht
5   Stage 2 158.7
6   Stage 1 161.8
7      <NA>    NA
8  Elevated 152.8
9   Stage 2 172.4
10   Normal 152.5
```

select()

- equivalently, using indexing:

```
> ss[,c('bpcat','bmxht')]
      bpcat bmxht
5   Stage 2 158.7
6   Stage 1 161.8
7      <NA>    NA
8  Elevated 152.8
9   Stage 2 172.4
10   Normal 152.5
```

mutate()

mutate()

- to add columns, use mutate():

```
> ss<-mutate(ss,inches=bmxht*0.393701)
> ss
  ridageyr     bpcat bmxht      hs military   inches
1       56  Stage 2 158.7 Smoking Military 62.48035
2       61  Stage 1 161.8      No       No 63.70082
3       42      <NA>    NA Smoking       No       NA
4       56 Elevated 152.8 Smoking       No 60.15751
5       65  Stage 2 172.4 Smoking       No 67.87405
6       26   Normal 152.5      No       No 60.03940
```

- 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       56  Stage 2 158.7 Smoking Military 62.48035
2       61  Stage 1 161.8      No       No 63.70082
3       42     <NA>    NA Smoking       No       NA
4       56 Elevated 152.8 Smoking       No 60.15751
5       65  Stage 2 172.4 Smoking       No 67.87405
6       26   Normal 152.5      No       No 60.03940
```

summarize()

summarize()

- for summaries, use `summarize()`:

```
> summarize(ss,
+           age=mean(ridageyr),
+           height=mean(inches,na.rm=TRUE))
  age   height
1  51 62.85043
```

- 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
51.00000 62.85043
```

summarize()

- using different types of summaries:

```
> summarize(ss,
+           mean.age=mean(ridageyr),
+           median.height=median(inches,na.rm=TRUE))
  mean.age median.height
1       51      62.48035
```

summarize()

- using base R:

```
> data.frame(mean.age=mean(ss$ridageyr),
+            median.height=median(ss$inches,na.rm=TRUE))
  mean.age median.height
1       51       62.48035
```

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       56
```

- 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
4 Elevated       No       56
```

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 No             61.9     43.5
2 Smoking        63.5     54.8
```

- 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
No       61.87011    43.50
Smoking 63.50397    54.75
```

group_by()

- equivalently, using the previously *not* discussed function `aggregate()`, from base R:

```
> aggregate(cbind(inches,ridageyr)~hs,data=ss,FUN=mean)
       hs   inches ridageyr
1      No 61.87011     43.5
2 Smoking 63.50397     59.0
```

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 No                61.9     43.5
2 Smoking           63.5     54.8
```

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     73557       69 bpxdi1    72
5770  73557       69 bpxdi2    76
11539 73557       69 bpxdi3    74
17308 73557       69 bpxdi4    NA
2     73558       54 bpxdi1    62
5771  73558       54 bpxdi2    80
```

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
73557.1 73557       69     1    72
73557.2 73557       69     2    76
73557.3 73557       69     3    74
73557.4 73557       69     4    NA
73558.1 73558       54     1    62
73558.2 73558       54     2    80
```

Reshaping data from wide to long

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