Intro to Dplyr

Desiree Narango

October 12, 2015

What is Dplyr?

Dplyr introduces a new 'grammar of Data Manipulation'. It uses simple verbs and syntax for basic data manipulation of dataframes in R.

Why use Dplyr?

- Helps you organize data
- Faster than Base R, Plyr
- Data Frame Friendly
- Understands column names
- Uses Piping
- KEEPS A RECORD OF YOUR MANIPULATIONS!!

What are the main functions?

function	Action	Similar to (in excel)
filter()	remove rows	'filter'
arrange()	reorder rows	'sort'
select()	select certain columns	'delete', 'copy/paste'
mutate()	create new variables	equations
summarize()	calculate summary stats	pivot tables

Other useful functions

function	Action	Similar to (in excel)
group_by()	Group a dataframe by one or more variables	?
desc()	sort in decending order	'sort'
distinct()	select distinct values	base::unique
inner_join()	merge two dataframes (x and y that match)	vlookup
left_join()	merge two dataframes (keep all x rows)	vlookup

Using dplyr

First load the program and upload your data

```
library(dplyr)
trees_raw<-read.csv("C:/Users/dnarango/Desktop/trees.csv")
veg<-read.csv("C:/Users/dnarango/Desktop/veg.csv")

#library(RCurl)
#fileURL <- getURL('https://raw.githubusercontent.com/SCBI-MigBirds/MigBirds/master/data/exampleBirdDat
#trees_raw <- read.csv(text = fileURL)
```

JOIN

- Basic Syntax: left_join(dataframeX,dataframeY, by="column")

You can use left_join or inner_join to merge two datasets

```
####inner_join: keeps only x rows with x and y
####left_join: keeps all x rows

# base::paste makes a new unique variable by combining 'site' and 'point'
trees_raw$ID<-paste(trees_raw$site, trees_raw$point)
veg$ID<-paste(veg$site, veg$Point)

# Now we join by our new unique variable
trees<-left_join(trees_raw, veg, by="ID")
```

FILTER

- Basic Syntax: filter(dataframe, logical condition)

You can use filter to select only those *rows* that conform to some logic. For example, from the *trees* dataframe, select only those rows where the *condition* column says 'planted'

```
trees_planted<-filter(trees, Condition=='planted' )
```

You can also filter by multiple columns or multiple criteria

```
blackcherry_planted<-filter(trees, treespp=="black cherry"| treespp=="norway spruce")

largetrees<-filter(trees,Condition!="dead" & height>=20)
```

Logic Symbols

Symbol	Expression
==	equal to
!=	not equal to
>, >=	greater than, greater than or equal to
<,<=	less than, less than or equal to
&	x AND y
|	x OR y

MUTATE

- Basic Syntax: mutate(dataframe, newvariable = operation)

```
trees[,6:15][is.na(trees[,6:15])]<- 0   ### First replace NAs with 0
mutate(trees, dbhsum_1_2 =dbh1+dbh2)    ### adds dbh1 and dbh2
```

You can also use base:functions in *mutate* but may need to conform to that syntax

```
trees_sum<-mutate(trees, dbhsum =rowSums(trees[6:15]))
mutate(trees, basal=trees_sum$dbhsum^2 * 0.00007854)
```

SELECT

- Basic Syntax: select(dataframe, column1, column2, column3, etc.)

Select can be easier to use than indexing in base because select can recognize column names

in base

```
trees_dbh<-trees_sum[,c(1,4,6:15,22)]     #by position
trees_dbh<-trees_sum[,c('site.x','treespp', 'dbh1', 'dbh2','dbh3','dbh4','dbh5')]  #by name
```

In *dplyr*

```
trees_dbh<-select(trees_sum, site.x, treespp, dbh1:dbh10, dbhsum)
```

ARRANGE

- Basic Syntax: arrange(dataframe, column)

```
trees_dbh<-arrange(trees_dbh, dbhsum)
trees_dbh<-arrange(trees_dbh, desc(dbhsum))
```

SUMMARISE

- Basic Syntax: summarise(dataframe, newvariable=function(variable))

```
summarize(trees_sum, N=sum(dbhsum))  ##not useful?
```

```
##        N
## 1 5222.9
```

```
sum(trees_sum$dbhsum)                ##same as
```

```
## [1] 5222.9
```

Used alone, summarise is not super useful but used in conjunction with group_by, summarise can be much more versatile. Use group_by and see how it affects your dataframe

```
trees_sum_group<-group_by(trees_sum, site.x)
##compare with
head(trees_sum_group)
head(trees_sum)
```

```
summarize(trees_sum_group,
          total=sum(dbhsum)
          )

#different groupings

trees_spp_group<-group_by(trees_sum, treespp)
summarize(trees_spp_group,
          total=sum(dbhsum))

#multiple groupings - crazy!

trees_spp_site_group<-group_by(trees_sum, site.x, treespp)
summarize(trees_spp_site_group,
          total=sum(dbhsum))
```

Piping

Makes it possible to easily chain operations + Basic Syntax: %<%, means 'then'

Here is the code to group_by, *then*, summarise

```
finaldata<-trees_sum %>%
              group_by(site.x, treespp) %>%
              summarize(total=max(dbhsum))

head(finaldata)
```

```
## Source: local data frame [6 x 3]
## Groups: site.x
##
##       site.x           treespp total
## 1 ahmafatdc1    american holly    12
## 2 ahmafatdc1      black cherry    15
## 3 ahmafatdc1     bradford pear     8
## 4 ahmafatdc1      deodar cedar    18
## 5 ahmafatdc1          euonymus    29
## 6 ahmafatdc1 flowering dogwood     4
```

Practice Exercise

Can we put all our data manipulation into one line of code? Steps:

1. Only 2014 sites that do not have forest cover
2. Only include the variables: site, shrubspp, height, all dbhs, and pavement
3. Create a sum of all dbh
4. Calculate the basal area (hint: basal area is the dbh^2 * 0.00007854)
5. Sum dbh across site and tree spp
6. Find mean basal for each site, tree spp
7. count all the trees
8. Descending order by treecount
9. Use Piping
10. Name your dataframe 'Final Data'

Get more practice by using the *SWIRL* package tutorial

ANSWER

```
trees[,6:15][is.na(trees[,6:15])]<- 0

finaldata2<-trees %>%
  filter(year=="2014" & forest==0)%>%
  select(site.x, treespp,height, dbh1:dbh10, pavement)%>%
  mutate(dbhsum=dbh1+dbh2+dbh3+dbh4+dbh4+dbh5+dbh6+dbh7+dbh8+dbh9+dbh10)  %>%
  mutate(basal=(dbhsum^2)*0.00007854) %>%
  group_by(site.x, treespp) %>%
  summarize(total=sum(dbhsum, na.rm=TRUE),
            mean_basal=mean(basal),
            treecount=n() )%>%
  arrange(desc(treecount))
```