Learning *R*

Carl James Schwarz

StatMathComp Consulting by Schwarz
cschwarz.stat.sfu.ca @ gmail.com

Split-Apply-Combine Paradigm

Introduction to *plyr* package

Table of Contents I

Split-Apply-Combine

Split - Apply - Combine
Performing the same analysis
to multiple chunks of your data
R's implementation of PivotTables (and more) in Excel

Split - Apply - Combine

Split-Apply-Combine

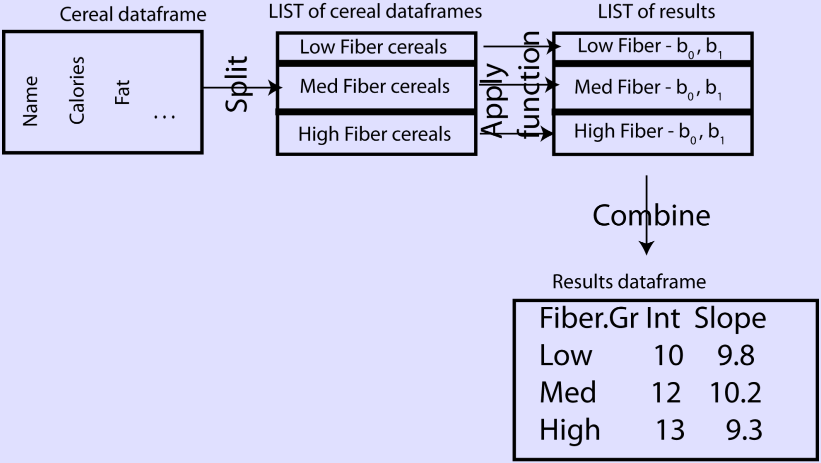
- **Split** up a big data frame
- **Apply** a function to each piece
- **Combine** the results together

Examples

- Compute the mean calories/serving for each display shelf.
- Compute number of accidents and p(fatality) for each day in the year.
- Fit a separate regression line to different fiber groups.
- Do a separate analysis for each year of accident data.

Split - Apply - Combine - Schematic

Find the slope and intercept of the regression of Calories vs. Fat for each Fiber Group in the cereal dataframe.



Cereal dataframe

Name

Calories

Fat

. . .

Split

LIST of cereal dataframes

Low Fiber cereals

Med Fiber cereals

High Fiber cereals

Apply function

LIST of results

Low Fiber - b_0, b_1

Med Fiber - b_0, b_1

High Fiber - b_0, b_1

Combine

Results dataframe

Fiber.Gr	Int	Slope
Low	10	9.8
Med	12	10.2
High	13	9.3

Split - Apply - Combine

Base *R* procedures (AVOID)

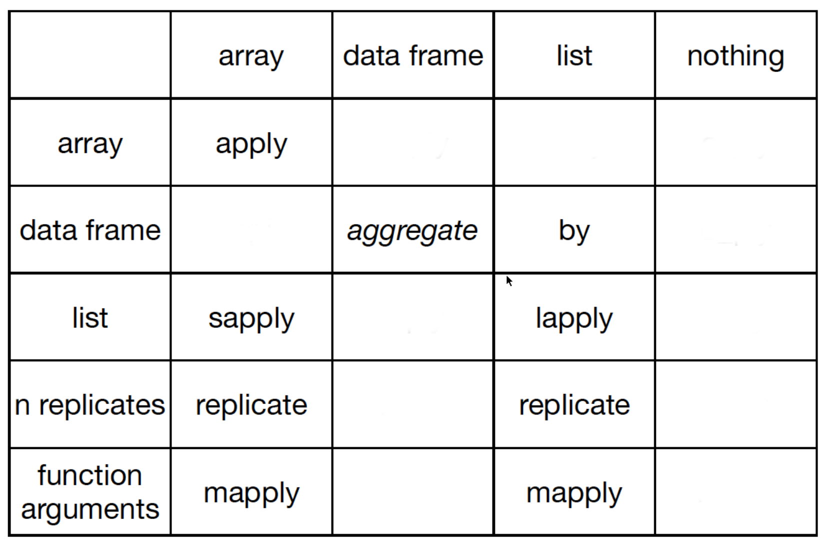
- *by()* takes data.frame → list
- *split()* takes data.frame → list
- *lapply()* takes list → list
- *sapply()* takes list → vector or matrix

plyr package (much more logically arranged) (RECOMMENDED)

- *xyply()* where *x* and *y* are d=data.frame, l=list, a=array, _=nothing
- Hadley Wickham (2011).
  The Split-Apply-Combine Strategy for Data Analysis.
  Journal of Statistical Software, 40(1), 1-29.
  `http://www.jstatsoft.org/v40/i01/`.
- *dplyr* package - more advanced and only for data.frames.

Split - Apply - Combine

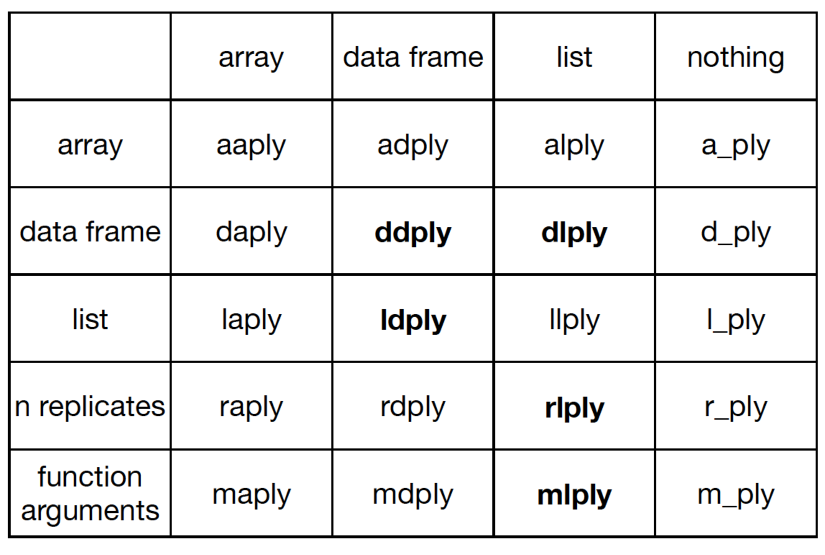
AVOID: Base *R* (input data structure (left); output data structure (top))



	array	data frame	list	nothing
array	apply			
data frame		*aggregate*	by	
list	sapply		lapply	
n replicates	replicate		replicate	
function arguments	mapply		mapply	

Split - Apply - Combine

USE: *plyr* package (input data structure (left); output data structure (top))



	array	data frame	list	nothing
array	aaply	adply	alply	a_ply
data frame	daply	**ddply**	**dlply**	d_ply
list	laply	**ldply**	llply	l_ply
n replicates	raply	rdply	**rlply**	r_ply
function arguments	maply	mdply	**mlply**	m_ply

Split - Apply - Combine - *ddply() + summarize()*

Cereal dataset.

Find the number of cereals and the mean calories/serving for each shelf

```
cereal <- read.csv(file.path(..., ’cereal.csv’),
                   header=TRUE, as.is=TRUE,
                   strip.white=TRUE)
cereal[1:5,]

library(plyr)
sumstats <- plyr::ddply(cereal, "shelf", plyr::summarize,
                ncereal=length(name),
                mean.calories=mean(calories))
sumstats
```

```
> sumstats
  shelf ncereal mean.calories
1     1      20      100.5000
2     2      21      107.6190
3     3      36      106.1111
```

Split - Apply - Combine - *ddply()* + *summarize()*

CAUTION: Because of conflicts between the *plyr* and *dplyr* packages, ALWAYS

- *plyr::* before the function name
- *plyr::summarize* - this is particularly important.

Split - Apply - Combine - *ddply()* + *summarize()* Exercise I

Find the following quantities for each shelf:

- Standard deviation of calories/serving
- Mean number of calories from fat (1 g of fat has 9 calories)
- Mean proportion of calories from fat of total calories.
- Mean weight/serving

```
> sumstats
  shelf std.calores mean.fcal mean.pcal.fat  mean.wt
1     1    11.45931      5.40    0.05404545 0.991500
2     2    12.20851      9.00    0.07986014 1.015714
3     3    29.01012     11.25    0.09859932       NA
```

Split - Apply - Combine - *ddply()* + *summarize()* Exercise I

```
library(plyr)
sumstats <- plyr::ddply(cereal, "shelf", plyr::summarize,
            std.calores=sd(calories),
            mean.fcal = mean(fat*9),
            mean.pcal.fat = mean( fat*9 / calories),
            mean.wt=mean(weight))
sumstats
```

```
> sumstats
  shelf std.calores mean.fcal mean.pcal.fat  mean.wt
1     1    11.45931      5.40    0.05404545 0.991500
2     2    12.20851      9.00    0.07986014 1.015714
3     3    29.01012     11.25    0.09859932       NA
```

Split - Apply - Combine - *ddply()* + *summarize()* Exercise I

Revise to account for missing values:

```
> sumstats
  shelf std.calores mean.fcal mean.pcal.fat  mean.wt
1     1    11.45931      5.40    0.05404545 0.991500
2     2    12.20851      9.00    0.07986014 1.015714
3     3    29.01012     11.25    0.09859932 1.062353
```

Split - Apply - Combine - *ddply()* + *summarize()* Exercise I

Revise to account for missing values:

```
library(plyr)
sumstats <- plyr::ddply(cereal, "shelf", plyr::summarize,
            std.calores=sd(calories),
            mean.fcal = mean(fat*9),
            mean.pcal.fat = mean( fat*9 / calories),
            mean.wt=mean(weight, na.rm=TRUE))
sumstats
```

```
> sumstats
  shelf std.calores mean.fcal mean.pcal.fat  mean.wt
1     1    11.45931      5.40    0.05404545 0.991500
2     2    12.20851      9.00    0.07986014 1.015714
3     3    29.01012     11.25    0.09859932 1.062353
```

Split - Apply - Combine - *ddply()* + *summarize()* Exercise II

Fit a separate regression line between calories and fat and report the intercept and slope for each shelf.

Recall line for ALL of data is found as:

```
result <- lm(calories ~ fat, data=cereal)
summary(result)
coef(result)
coef(result)[1]
coef(result)[2]
```

```
> sumstats
  shelf intercept      slope
1     1 100.27778  0.3703704
2     2  96.78571 10.8333333
3     3  91.36752 11.7948718
```

Split - Apply - Combine - *ddply()* + *summarize()* Exercise II

```
library(plyr)
sumstats <- plyr::ddply(cereal, "shelf", plyr::summarize,
                 intercept=coef(lm(calories ~fat))[1],
                 slope    =coef(lm(calories ~fat))[2])
sumstats
```

```
> sumstats
  shelf intercept      slope
1     1 100.27778  0.3703704
2     2  96.78571 10.8333333
3     3  91.36752 11.7948718
```

A better method will be demonstrated later that doesn't require repeated model fitting.

Split - Apply - Combine - *ddply()* + *summarize()* Exercise III

Fit a separate regression line between calories and fat and report the intercept and slope for each shelf.

Recall line for ALL of data is found as:

```
result <- lm(calories ~ fat, data=cereal)
summary(result)
coef(result)
coef(result)[1]
coef(result)[2]
```

```
> sumstats
  shelf intercept      slope
1     1 100.27778  0.3703704
2     2  96.78571 10.8333333
3     3  91.36752 11.7948718
```

Split - Apply - Combine - *ddply()* + *summarize()* Exercise IV

Refer to *road-accidents-2010.csv* file in SampleData.

- Read data into *R*.
- Convert input date to internal *R* dates.
- Find number of accidents by day of year (use *ddply()* and *summarize()* in *plyr* package)
- Plot # accidents/day by day of year.
- Fit a *lowess()* smoother to data using *geom_smooth()*

Look at number of accident by day of the week

- Extract day of the week using *format()* or *weekdays()* functions.
- Use *geom_boxplot()* as seen earlier

Split - Apply - Combine - *ddply()* + *summarize()* Exercise IV

```
# The accident data
accidents <- read.csv(file.path(... ,'road-accidents-2010.cs
                 header=TRUE,
                 as.is=TRUE, strip.white=TRUE)
accidents[1:5,]
str(accidents)
```

```
> accidents[1:5,]
.....
  Accident_Severity Number_of_Vehicles Number_of_Casualties
1                 3                  2                    1
2                 3                  1                    1

> str(accidents)
'data.frame': 154414 obs. of  33 variables:
...
 $ Date    : chr  "11/01/2010" "11/01/2010" "12/01/2010" "02/
...
```

Split - Apply - Combine - *ddply()* + *summarize()* Exercise IV

```

# Convert date to internal date format
accidents$mydate <- as.Date(accidents$Date,
                format="%d/%m/%Y")
sum(is.na(accidents$mydate))
accidents[1:5,]
str(accidents)
```

```
> accidents[1:5,]
...
  Urban_or_Rural_Area Did_Police_Officer_Attend_Scene_of_Acc
1                   1
2                   1
> str(accidents)
'data.frame': 154414 obs. of  33 variables:
 $ Date         : chr  "11/01/2010" "11/01/2010" "12/01/2010
 $ mydate     : Date, format: "2010-01-11" "2010-01-11" "2010
>
```

Split - Apply - Combine - *ddply()* + *summarize()* Exercise IV

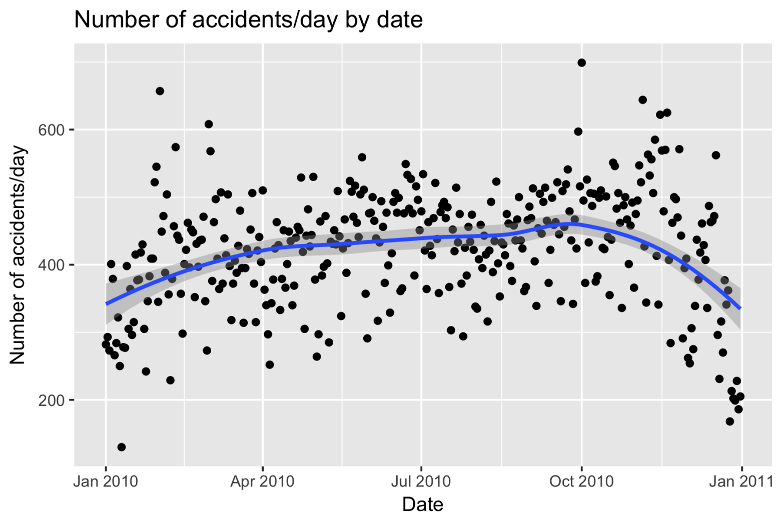
```
# Summarize number of accidents by date
library(plyr)
naccidents <- plyr::ddply(accidents, "mydate",
                          plyr::summarize,
                          freq=length(Accident_Index))
naccidents[1:5,]
str(naccidents)
```

```
> naccidents[1:5,]
      mydate freq
1 2010-01-01  282
2 2010-01-02  293
...
> str(naccidents)
'data.frame': 365 obs. of  2 variables:
 $ mydate: Date, format: "2010-01-01" "2010-01-02" "2010-01-
 $ freq  : int  282 293 273 401 379 266 284 322 250 130 ...
```

Split - Apply - Combine - *ddply()* + *summarize()* Exercise IV

```
plotnacc <- ggplot(data=naccidents, aes(x=mydate, y=freq))+
      ggtitle("Number of accidents/day by date")+
      xlab("Date")+ylab("Number of accidents/day")+
      geom_point()+
      geom_smooth()
plotnacc
```

Split - Apply - Combine - *ddply()* + *summarize()* Exercise IV



Split - Apply - Combine - *ddply()* + *summarize()* Exercise V

Refer to *road-accidents-2010.csv* file in SampleData.

- Create 0/1 variable if fatality occurs (no or yes; check codebook for *Accident_ Severity*).
  Use the magic incantation of *recode()* function in *car* package.
- Find proportion of accidents with fatality by day of year
  - The mean of a 0/1 variable is the proportion.
    Use the magic incantation of *ddply()* and *summarize()* in the *plyr* package.
- Plot proportion of fatalities by day of year.
- Fit a *lowess()* smoother to data from *geom_ smooth()*
- Plot proportion of fatalities by day of the week
  - Hint: Extract weekday using *format()*.
  - Hint: Use *geom_ boxplot()* as seen earlier with some jittering and notches.

Split - Apply - Combine - *ddply()* + *summarize()* Exercise V

```
names(accidents)
unique(accidents$Accident_Severity)
library(car)
accidents$Fatality <- recode(accidents$Accident_Severity,
              ’ 1=1; 2:hi=0’)
accidents[1:5, c("Accident_Severity", "Fatality")]
xtabs(~Fatality + Accident_Severity, data=accidents)
```

```
> accidents[1:5, c("Accident_Severity", "Fatality")]
  Accident_Severity Fatality
1                 3        0
2                 3        0
```

```
> xtabs(~Fatality + Accident_Severity, data=accidents)
        Accident_Severity
Fatality      1      2      3
       0      0  20440 132243
       1   1731      0      0
```

Split - Apply - Combine - *ddply()* + *summarize()* Exercise V

The *summarize()* and *ddply()* functions in *plyr* package are quite useful for simple summaries by groups.
Example of the Split-Apply-Combine paradigm to be explained later.

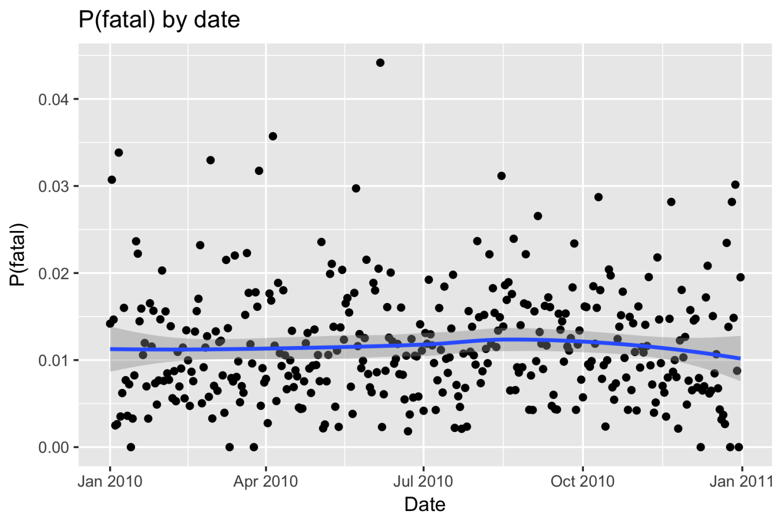
```
library(plyr)
pfatal.df <- plyr::ddply(accidents, "mydate", plyr::summariz
            freq=length(mydate),
            pfatal=mean(Fatality))
pfatal.df[1:5,]
```

```
> pfatal.df[1:5,]
      mydate freq      pfatal
1 2010-01-01  282 0.014184397
2 2010-01-02  293 0.030716724
3 2010-01-03  273 0.014652015
4 2010-01-04  401 0.002493766
5 2010-01-05  379 0.002638522
```

Split - Apply - Combine - *ddply()* + *summarize()* Exercise V

```
plotpfatal <- ggplot(data=pfatal.df,
              aes(x=mydate, y=pfatal))+
     ggtitle("P(fatal) by date")+
     xlab("Date")+ylab("P(fatal)")+
     geom_point()+
     geom_smooth()
plotpfatal
```

Split - Apply - Combine - *ddply()* + *summarize()* Exercise V

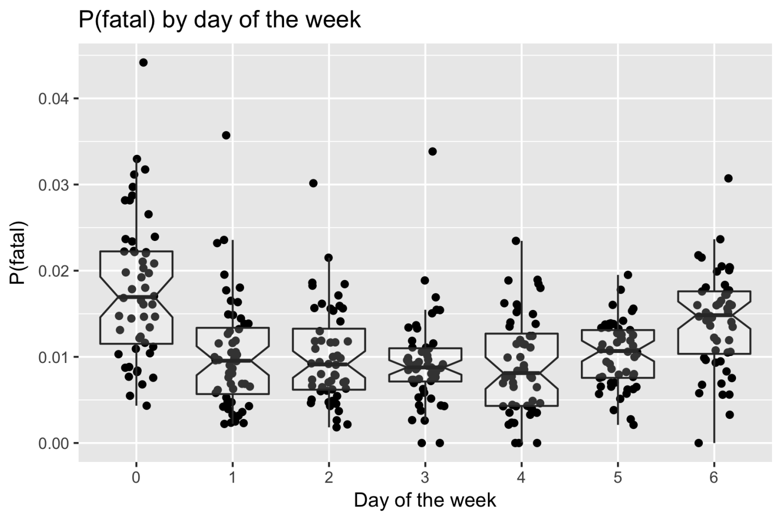


Split - Apply - Combine - *ddply()* + *summarize()* Exercise V

```
# Extract day of the week - leave as character
pfatal.df$weekday <- format(pfatal$mydate, format="%w") # l
pfatal.df[1:10,]

plotpfatal2 <- ggplot(data=pfatal.df, aes(x=weekday, y=pfata
  ggtitle("P(fatal) by day of the week")+
  xlab("Day of the week")+ylab("P(fatal)")+
  geom_point(position=position_jitter(w=0.2))+
  geom_boxplot(notch=TRUE, alpha=0.2)
plotpfatal2
```

Split - Apply - Combine - *ddply()* + *summarize()* Exercise V



Split - Apply - Combine - Exercise

Refer back to the accidents dataset. For each day. compute

- Number of accidents
- Proportion of fatalities
- MEAN weather severity (*Weather_ Conditions*). Not really valid but a close approximation)
- Day of the week (0=Sunday)

Use *plyr::summarize*

Plot number of accidents over the year with the SIZE of point related to mean weather conditions.
Add loess curve.

Split - Apply - Combine - Exercise

```
accidents <- read.csv(file.path(...,'road-accidents-2010.csv
             as.is=TRUE, strip.white=TRUE)
# Convert date to internal date format
accidents$mydate <- as.Date(accidents$Date,
                       format="%d/%m/%Y")
# Create the fatality variable
accidents$Fatality <- accidents$Accident_Severity == 1
```

Split - Apply - Combine - Exercise VI

Using *ddply()* and *summarize()*

```
naccidents <- plyr::ddply(accidents, "mydate", plyr::summar
                    freq=length(mydate),
                    pfatal=mean(Fatality),
                    mean.weather=mean(Weather_Conditions),
                    dow=format(mydate, "%w")[1])
naccidents[1:5,]
```

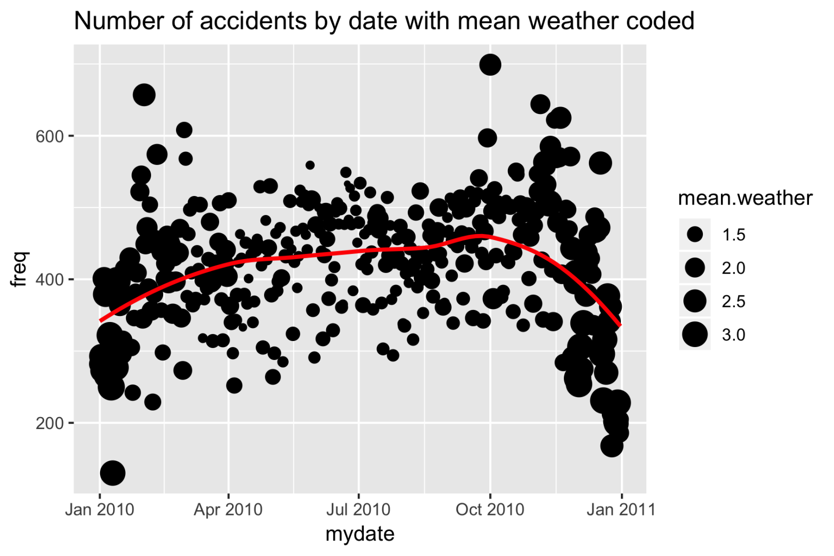
```
> naccidents[1:5,]
      mydate freq      pfatal mean.weather dow
1 2010-01-01  282 0.014184397     2.262411   5
2 2010-01-02  293 0.030716724     2.740614   6
3 2010-01-03  273 0.014652015     2.857143   0
4 2010-01-04  401 0.002493766     2.518703   1
5 2010-01-05  379 0.002638522     2.936675   2
```

Split - Apply - Combine - Exercise VI

Make the plots

```
newplot <- ggplot(data=naccidents,
                aes(x=mydate, y=freq ))+
  ggtitle("Number of accidents by date with mean weather co
  geom_point( aes(size=mean.weather))+
  geom_smooth(method="loess", color="red", se=FALSE)
newplot
```

Split - Apply - Combine - Exercise VI



Split - Apply - Combine - Exercise VI

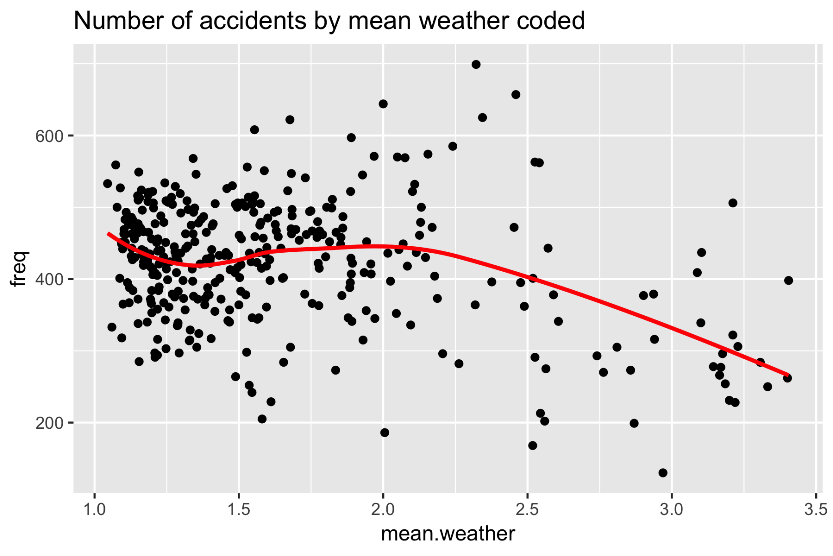
Plot number of accidents vs. mean weather conditions;
Add loess curve

Split - Apply - Combine - Exercise VI

Plot number of accidents vs. mean weather conditions;

```
newplot <- ggplot(data=naccidents,
                 aes(x=mean.weather, y=freq))+
  geom_point( ) +
  geom_smooth(method="loess", color="red", se=FALSE) +
  ggtitle("Number of accidents by mean weather coded")
newplot
```

Split - Apply - Combine - Exercise VI



Split - Apply - Combine - Exercise VII

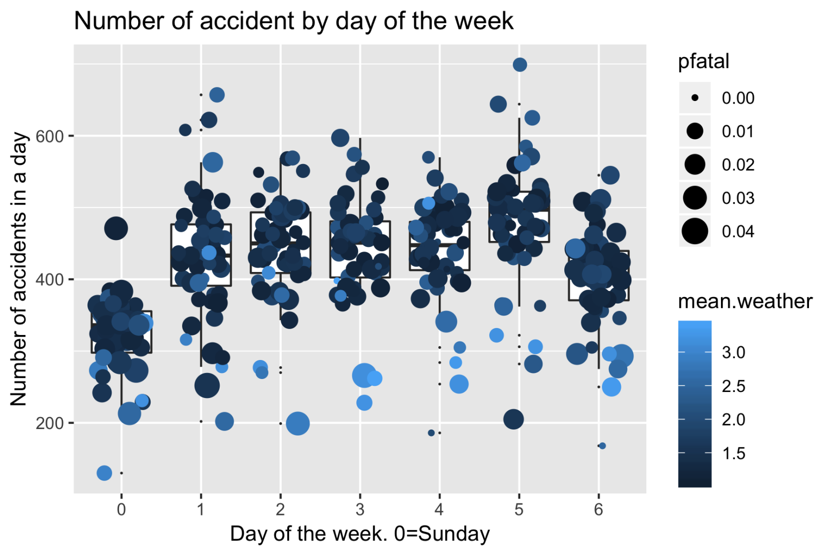
Accident data.

Make a box-plot of number of accident by day of the week coded using proportion of fatalities by the size of the symbol and the mean weather condition by a color gradient.

Split - Apply - Combine - Exercise VII

```
newplot <- ggplot(data=naccidents, aes(x=dow, y=freq))+
  geom_boxplot( ) +
  geom_jitter(aes(size=pfatal, color=mean.weather),
           position=position_jitter(w=.3, h=.0))+
  ggtitle("Number of accident by day of the week")+
  xlab("Day of the week. 0=Sunday") +
  ylab("Number of accidents in a day")
newplot
```

Split - Apply - Combine - Exercise VII



Split - Apply - Combine - Summary (Simple)

VERY COMMON PARADIGM IN *R*.

- Virtually unnecessary to use *for* loops in *R* if computations for each chunk are independent and do not depend on other chunks.
- Makes it easy to parallelize your work (routines are set up to use multiple machines)
- Most common usage is *ddply()*
- The *dplyr* package is specifically design for LARGE data frames and is much faster.

Split - Apply - Combine - Summary (Simple)

Most simple usage is with *ddply()* and *summarize()*

```
sumstat <- plyr::ddply( dataframe, "chunking variable",
                             plyr::summarize,
                             v1=....,
                             v2=....,
                             v3=...., .... )
```