Learning *R*

Carl James Schwarz

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

Split-Apply-Combine Paradigm

Advanced usage of the *dplyr* package

An extension of the *plyr* package only for data frames.

Table of Contents I

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 - *dplyr*

Why a new package

- Must faster than *plyr* for LARGE data frames
- Some additional functionality (not a huge amount)
- Piping

Why not use *dplyr*

- Not readily available for lists, but see *do()*.
- Conflicts with *plyr* package are a headache.
- Cannot parallelize, but see *multidplyr* package.

Equivalents

Action	*plyr*	*dplyr*	Notes
Simple Summary	*summarize()*	*summarize*	CAUTION
Sorting	none	*arrange()*	Base *order*
Filter	none	*filter*	Equivalent to df[...
Add columns	*mutate*	*mutate*	Add new variables
Group wise	*ddply, 'xxxxx', ...*	*group_by()*	
Select variables	none	*select()*	*grep()* and others.

Caution - load packages in right order if using both

```
1 library(plyr)
2 library(dplyr)
```

If you load in other direction

```
------------------------------------------------------------
You have loaded plyr after dplyr - this is likely to cause p
If you need functions from both plyr and dplyr, please load
library(plyr); library(dplyr)
------------------------------------------------------------
```

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

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

```
1  cereal <- read.csv('../sampledata/cereal.csv',
2                     header=TRUE, as.is=TRUE,
3                     strip.white=TRUE)
4  cereal[1:5,]
```

Summarizing by groups

```
1  library(plyr)
2  sumstats <- plyr::ddply(cereal, "shelf", plyr::summarize,
3              ncereal=length(name),
4              mean.calories=mean(calories))
5  sumstats
6
7
8  library(dplyr}
9  sumstats <- cereal %>%
10                group_by(shelf) %>%
11                  dplyr::summarize(
12                    ncereal=length(name),
13                    mean.calories=mean(calories))
14 sumstats
```

NOTE: ALWAYS qualify the *summarize*, i.e. plyr:: or dplyr::.
NOTE: In *plyr* grouping variables in quotes.
NOTE: In *dplyr* grouping variables NOT in quotes

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 - *dplyr+ 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 - *dplyr+ summarize()* Exercise I

```
1  library(plyr)
2  sumstats <- cereal %>%
3                group_by(shelf) %>%
4                 dplyr::summarize(
5                  std.calores=sd(calories),
6                  mean.fcal = mean(fat*9),
7                  mean.pcal.fat = mean( fat*9 / calories),
8                  mean.wt=mean(weight))
9  sumstats
```

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

Revise to account for missing values:

```
1  sumstats <- cereal %>%
2                group_by(shelf) %>%
3                 dplyr::summarize(
4                  std.calores=sd(calories),
5                  mean.fcal = mean(fat*9),
6                  mean.pcal.fat = mean( fat*9 / calories),
7                  mean.wt=mean(weight, na.rm=TRUE))
8  sumstats
```

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

```
1  sumstats <- cereal %>%
2              group_by(shelf) %>%
3               dplyr::summarize(
4                intercept=coef(lm(calories ~fat))[1],
5                slope    =coef(lm(calories ~fat))[2]
6              )
7  sumstats
```

```
> sumstats
  shelf intercept  slope
1     1     100.   0.370
2     2      96.8 10.8
3     3      91.4 11.8
```

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

Split - Apply - Combine - *dplyr*+ arbitrary function

Use the *do* function - it is rather awkward.

```
sumstats <- cereal %>% group_by(shelf) %>% do(
        (function(x){
         result <- lm(calories ~ fat, data=x) # notice use
         intercept <- coef(result)[1]
         slope    <-   coef(result)[2]
         res <- data.frame(intercept, slope, stringsAsFacto
         res
        })(.)
        )
sumstats
```

NOTE: Use of *do()* function.
NOTE: Anonymous functions enclosed by ()
NOTE: Additional (.) at the end.
NOTE: Must return a data frame.

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

```
1  # The accident data
2  accidents <- read.csv(...,
3                header=TRUE,
4                as.is=TRUE, strip.white=TRUE)
5  accidents[1:5,]
6  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 - *dplyr+ summarize()* Exercise IV

```
1
2  # Convert date to internal date format
3  accidents$mydate <- as.Date(accidents$Date,
4                format="%d/%m/%Y")
5  sum(is.na(accidents$mydate))
6  accidents[1:5,]
7  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
>
```

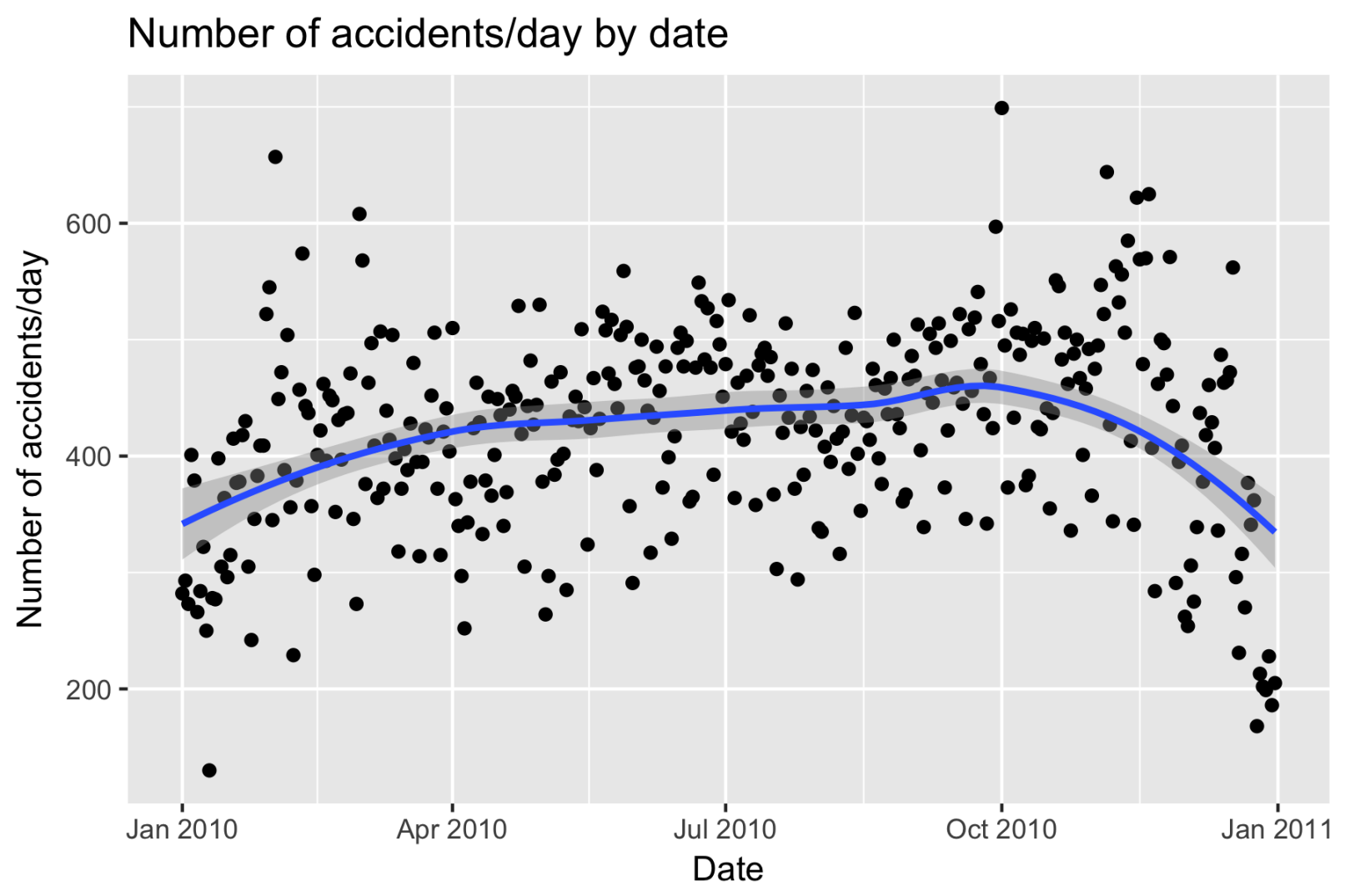
```
# Summarize number of accidents by date
naccidents <-
   accidents %>%
    group_by(mydate) %>%
       dplyr::summarize(
              freq=length(mydate),
              pfatal=mean(Fatality),
              mean.weather=mean(Weather_Conditions),
              dow=format(mydate, "%w")[1]
)
```

```
> 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 - *dplyr+ summarize()* Exercise IV

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

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



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

```
1  names(accidents)
2  unique(accidents$Accident_Severity)
3  library(car)
4  accidents$Fatality <- recode(accidents$Accident_Severity,
5                 ' 1=1; 2:hi=0')
6  accidents[1:5, c("Accident_Severity", "Fatality")]
7  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
```

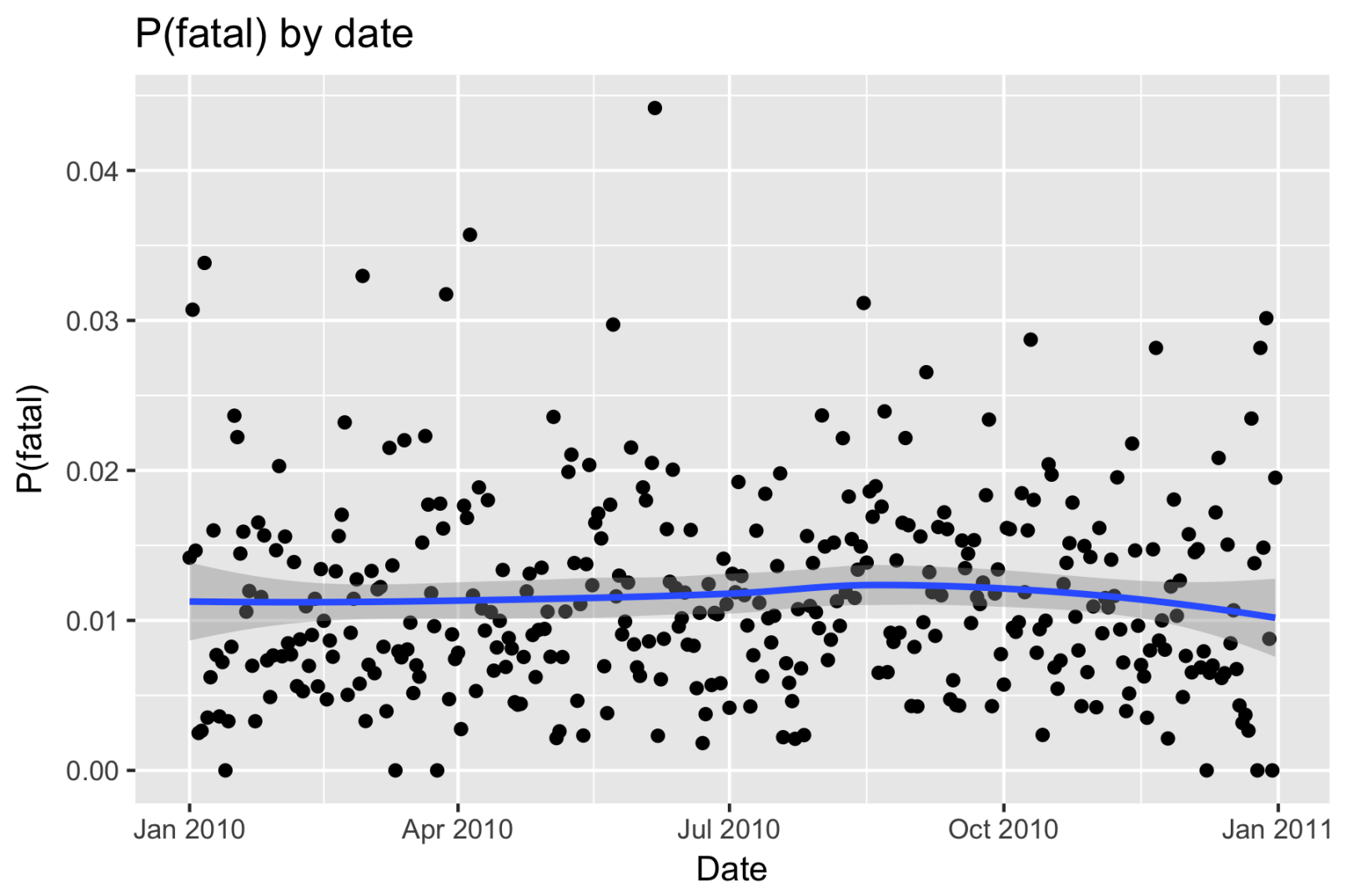
Split - Apply - Combine - *dplyr+ summarize()* Exercise V

```
naccidents <-
  accidents %>%
     group_by(mydate) %>%
       dplyr::summarize(
              freq=length(mydate),
              pfatal=mean(Fatality),
              mean.weather=mean(Weather_Conditions),
              dow=format(mydate, "%w")[1]
```

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

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

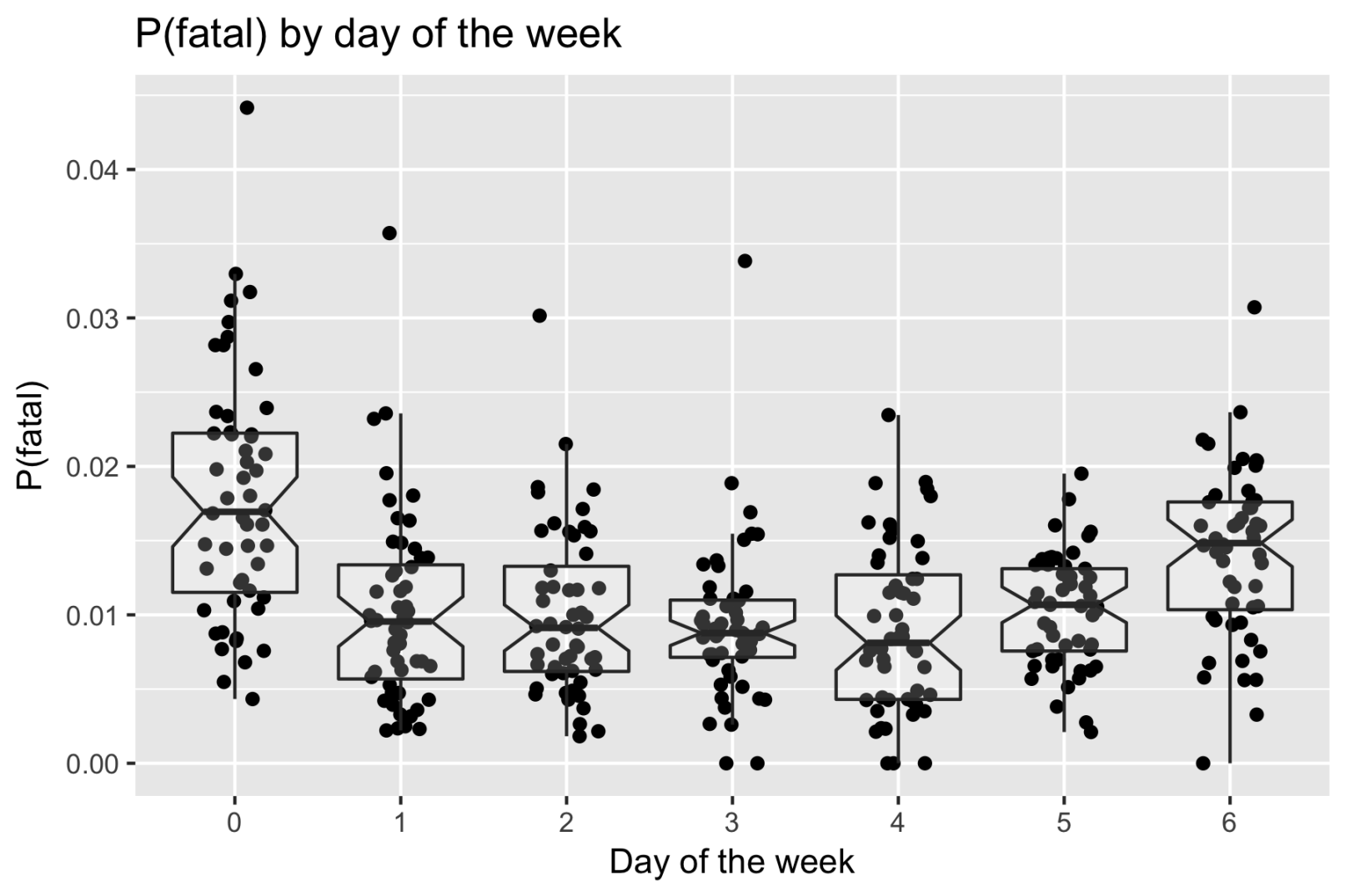
Split - Apply - Combine - *dplyr+ summarize()* Exercise V



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

```
1  # Extract day of the week - leave as character
2  pfatal.df$weekday <- format(pfatal$mydate, format="%w") # l
3  pfatal.df[1:10,]
4
5  plotpfatal2 <- ggplot(data=pfatal.df, aes(x=weekday, y=pfata
6    ggtitle("P(fatal) by day of the week")+
7    xlab("Day of the week")+ylab("P(fatal)")+
8    geom_point(position=position_jitter(w=0.2))+
9    geom_boxplot(notch=TRUE, alpha=0.2)
10 plotpfatal2
```

Split - Apply - Combine - *dplyr+ 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 *dplyr::summarize*

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

Split - Apply - Combine - Exercise

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

Split - Apply - Combine - Exercise VI

Using *dplyr* and *summarize()*

```
1  naccidents <- accidents %>% group_by(mydate) %>% do(
2              (function(x){
3                 freq <- nrow(x)
4                 mean.weather <- mean(x$Weather_Conditions)
5                 pfatal <- mean(x$Fatality)
6                 dow=format(x$mydate, "%w")[1]
7                 res <- data.frame(freq, mean.weather, pfatal,
8                 return(res)
9                 })(.)
10 )
```

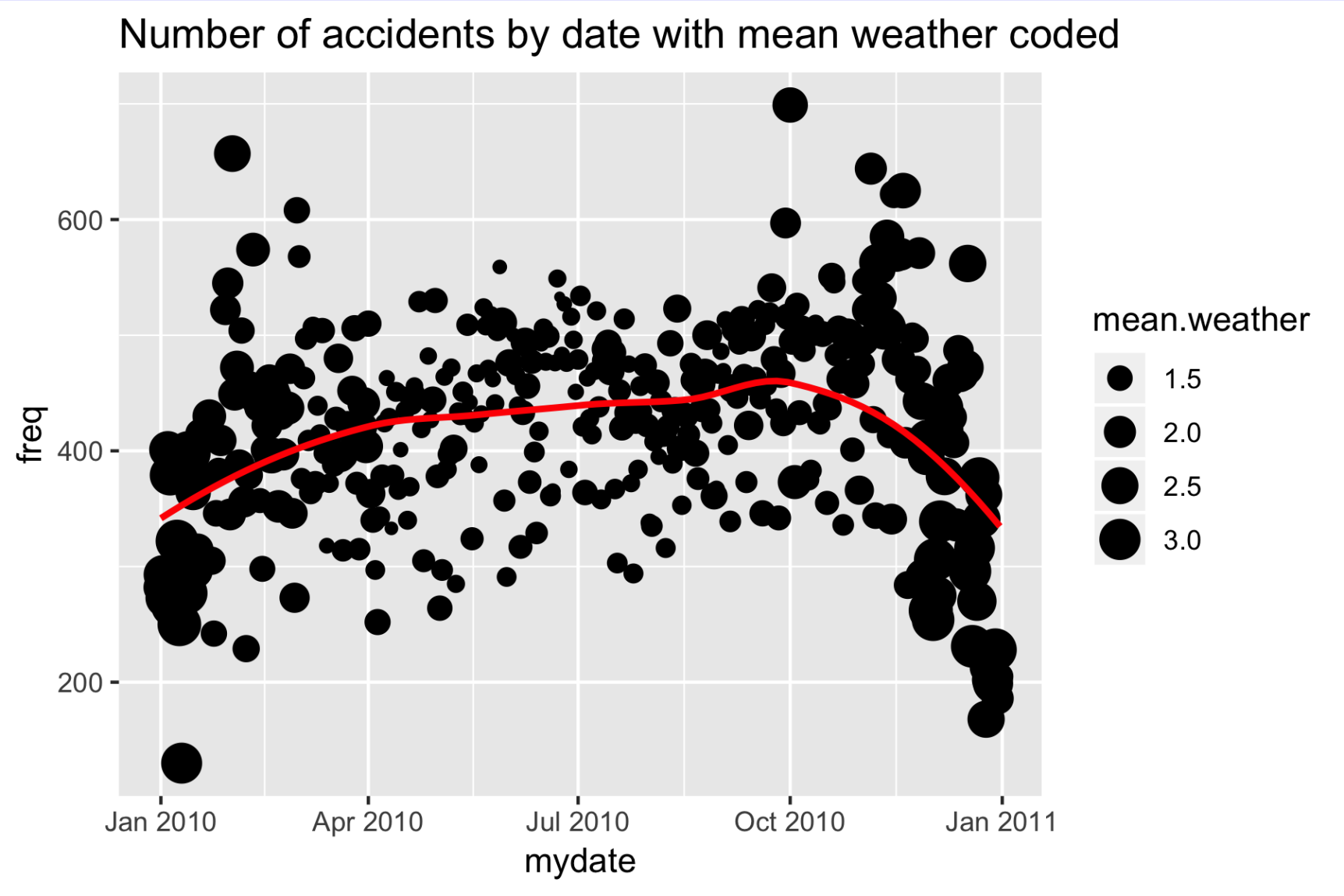
```
> naccidents[1:5,]
      mydate freq      pfatal mean.weather dow
1 2010-01-01  282 0.014184397     2.262411   5
2 2010-01-02  293 0.030716724     2.740614   6
3 2010-01-03  273 0.014652015     2.857143   0
4 2010-01-04  401 0.002493766     2.518703   1
5 2010-01-05  379 0.002638522     2.936675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

Split - Apply - Combine - Exercise VI

Make the plots

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

Split - Apply - Combine - Exercise VI



Split - Apply - Combine - Exercise VI

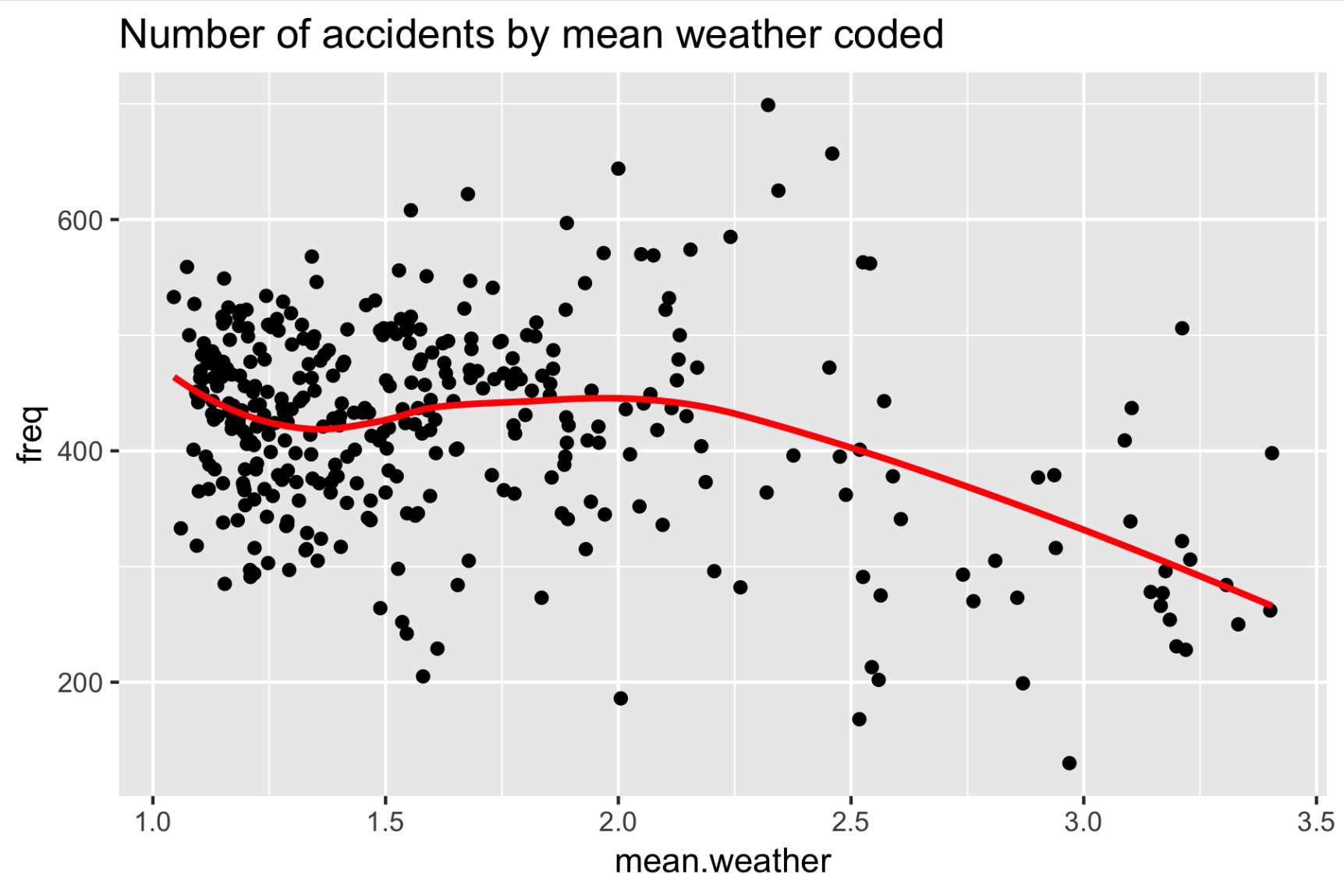
Plot number of accidents vs. mean weather conditions;
Add loess curve

Split - Apply - Combine - Exercise VI

Plot number of accidents vs. mean weather conditions;

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

Split - Apply - Combine - Exercise VI



Split - Apply - Combine - Exercise VII

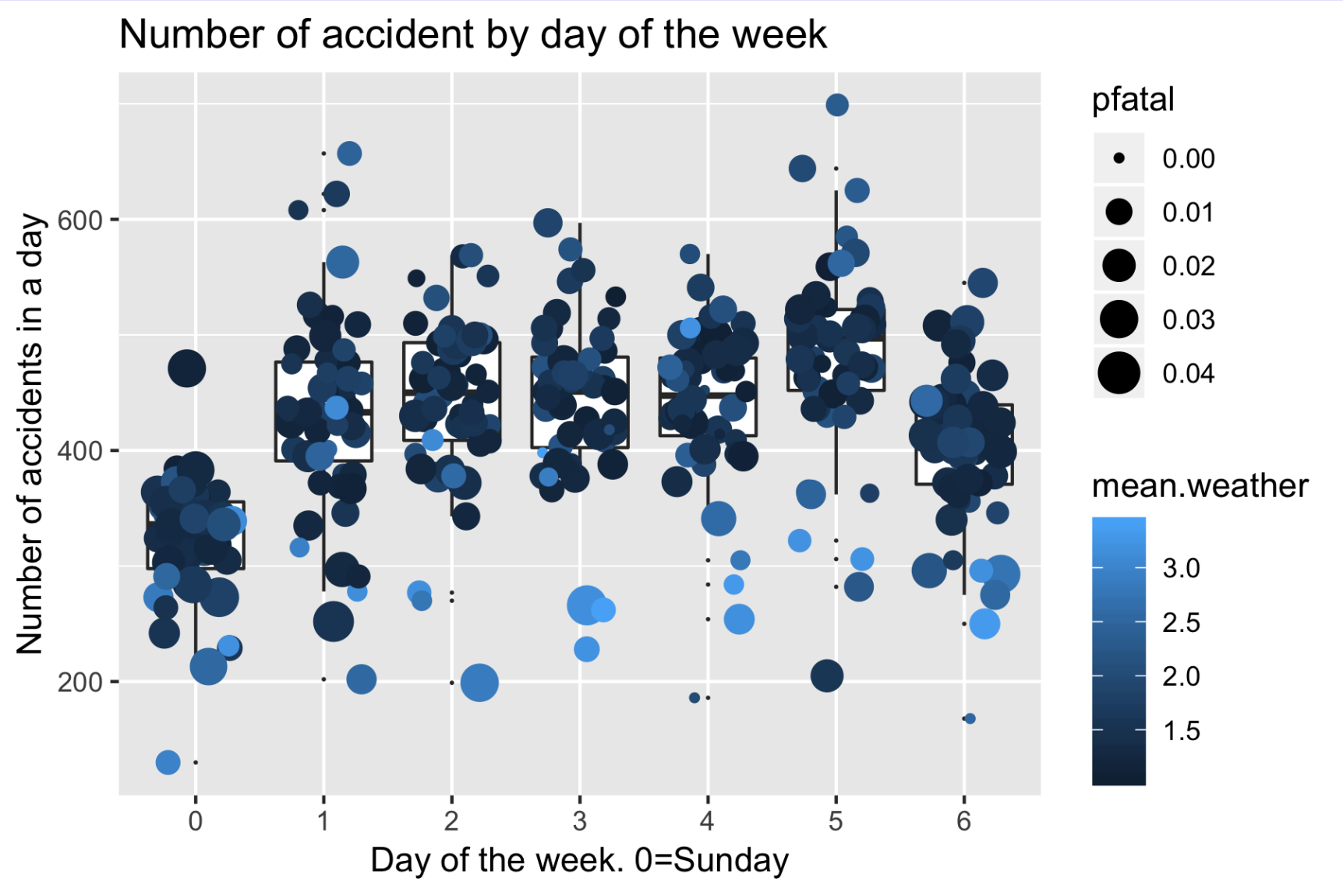
Accident data.

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

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

Split - Apply - Combine - Exercise VII



Passing the name of the variable to analyse. This is problematic in the *dplyr* package because use of non-standard evaluation, i.e. variable names not in quotes.

```
> dplyr::summarize(cereal, mean(fat))
  mean(fat)
1  1.012987
> dplyr::summarize(cereal, mean(var))
  mean(var)
1        NA
Warning message:
In mean.default(var) : argument is not numeric or logical: r
```

See the *Programming with dplyr* for the full gory details!

Passing the name of the variable to analyse. This is problematic in the *dplyr* package because use of non-standard evaluation, i.e. variable names not in quotes. Solution:

```
# Need to use the quo() and !! functions
var <- quo(fat)
var

dplyr::summarize(cereal, mean(!!var))
```

See the *Programming with dplyr* for the full gory details at `https://dplyr.tidyverse.org/articles/programming.html`!

Split - Apply - Combine - passing arguments to lowest level function

Example:Refer back to the cereal dataset. For each shelf group (and for an "arbitrary" variable), compute

- Number of observations
- Number of observations with missing values
- Mean of the variable
- SD of the variable

Split - Apply - Combine - Advanced

```
1  sumstat <- function(x, var){
2    # Compute some summary statistics for a data frame
3    #browser()
4    values <- x[,var,drop=TRUE]  # extract the variable value
5    n <- length(values)
6    nmiss <- sum(is.na(values))
7    mean.val  <- mean(values, na.rm=TRUE)
8    sd.val    <- sd(values, na.rm=TRUE)
9    data.frame(n,nmiss,mean=mean.val,sd=sd.val,
10                 stringsAsFactors=FALSE)
11 } # end of sumstat
12
13 sumstat(cereal, "calories")
14 sumstat(cereal, "weight")
```

NOTE: Use of *drop=* argument to deal with tibbles vs. data.frames

But how are the second (and additional arguments passed to *do()*?

```
1  cereal %>%
2     group_by(shelf) %>%
3         do( sumstatfun(., var='calories'))
4
5  cereal %>%
6     group_by(shelf) %>%
7         do( sumstatfun(., var='fat'))
```

NOTE: Use of *do()*
NOTE: How to specify additional arguments to the function
NOTE: Use of *drop=* argument with tibbles and data frames.

Write a function that takes a data frame and a variable name and

- Find the sample size, number of missing values, mean, its se, and a 95% normal-based confidence interval.
- Bonus - allow for different size of confidence limits, e.g. 90% confidence interval.

Either use the *t.test()* or *lm(y ~1)* or code yourself using sample size and t-distribution

Split - Apply - Combine - Advanced - Exercise

Sample output:

```
> my.simple.summary(cereal, "fat")
  variable  n nmiss     mean       sd        se conflevel
1      fat 77     0 1.012987 1.006473 0.1146982      0.95 0
> my.simple.summary(cereal, "fat", conflevel=.90)
  variable  n nmiss     mean       sd        se conflevel
1      fat 77     0 1.012987 1.006473 0.1146982       0.9 0
```

```
> cereal %>% group_by(shelf) %>% do(  my.simple.summary(., v
  shelf variable     n nmiss  mean    sd    se conflevel   l
1     1 fat         20     0  0.6  0.754 0.169      0.95 0.2
2     2 fat         21     0  1    0.775 0.169      0.95 0.6
3     3 fat         36     0  1.25 1.18  0.197      0.95 0.8
```

```
> cereal %>% group_by(shelf) %>% do(  my.simple.summary(., v
  shelf variable     n nmiss  mean    sd    se conflevel   l
1     1 fat         20     0  0.6  0.754 0.169       0.9 0.3
2     2 fat         21     0  1    0.775 0.169       0.9 0.7
```

Split - Apply - Combine - Advanced usage of *do()*

Equivalence with *dlply()* and *ldply()*

- It is convenient to do ALL computations for a chunk, return a list, and then extract from the list a needed rather than having several different function.
- Some output (like plots) cannot be stored in data frames.

Example: Refer back to the cereal dataset. For each shelf group (and for two "arbitrary" variable), compute

- Plot of Y vs. X (use *aes_string()* in *ggplot()*
- Regression of Y on Y. Use *lm(x[, Yvar] ~ x[, Xvar])*
- Return both in a list

```
sumstat(cereal, "calories", "fat")
```

should return a list with 2 elements.

Equivalence with *dlply()* and *ldply()*

```
1  sumstat <- function(x, Yvar, Xvar){
2     # Do the plot (use aes_string)
3     plot <- ggplot(data=x, aes_string(x=Xvar, y=Yvar))+
4        ggtitle(paste("Scatterplot of ", Yvar, " vs. ", Xvar,s
5        geom_point(position=position_jitter(h=.1, w=.1))+
6        geom_smooth(method="lm", se=FALSE)
7     fit <- lm( x[,Yvar] ~ x[, Xvar], data=x)
8     list(plot=plot, fit=fit)
9  }
10
11 res<- sumstat(cereal, "calories", "fat")
12 length(res)
```

Split - Apply - Combine - Advanced usage of *do()* I

Equivalence with *dlply()* and *ldply()*

- Now use *do()* to make a list of lists, once for each shelf

```
1  library(plyr)
2  res <- plyr::dlply(cereal, "shelf", sumstat,
3            Yvar="calories", Xvar="fat")
4
5  library(dplyr)
6  res.b <- cereal %>% group_by(shelf) %>% do( res=sumstat(., Y
```

NOTE: you need to NAME the returned value, e./g. (*res*)
NOTE: you get a linked list structure

Split - Apply - Combine - Advanced usage of *do()* II

```
> # you get a paired set of lists
> names(res.b)
[1] "shelf" "res"
> length(res.b$res)
[1] 3
> names(res.b$res[[1]])
[1] "plot" "fit"
> res.b$res[[1]]$fit

Call:
lm(formula = x[, Yvar] ~ x[, Xvar], data = x)

Coefficients:
(Intercept)    x[, Xvar]
   100.2778        0.3704
```

Split - Apply - Combine - Advanced usage of *do()*

Equivalence with *dlply()* and *ldply()*

- Now use *do()* to make a data frame of slope and se

```
1 report <- res.b %>%
2     do( data.frame(shelf=.$shelf,
3                    t(summary(.$res$fit)$coefficients[2,])) )
4 report
```

```
> report
  shelf Estimate Std..Error t.value Pr...t..
1     1    0.370       3.58   0.103 0.919
2     2   10.8         2.63   4.12  0.000576
3     3   11.8         3.70   3.19  0.00306
>
```

Split - Apply - Combine - Advanced - Exercise VIII

Refer back to the accident dataset.

- For each month, compute the number of days, weekdays and weekends. Hint: Use the *unique()* function on the dates within each month. Why do I want all three values?
- For each month, compute the total number of accidents with injury, those on weekends, and those on weekdays. Again, why do I want all three values?
- For each month, find the ratio of the number of accidents on weekday to weekends.
- Plot these over the year
- Add a suitable comparison line if accidents were uniformly spread over the days of the week. Note that the number of weekends and weekdays differs among months.

Split - Apply - Combine - Advanced - Exercise VIII

```
1  ... read accident data ....
2  ... convert dates to internal R format ...
3
4  # get the month for each accident date
5  accidents$month <- as.numeric(format(
6          accidents$mydate, "%m"))
```

Split - Apply - Combine - Advanced - Exercise VIII

```
1  mysummary <- function(accidents){
2  # Compute the number of weekend and weekdays in the month (
3  # Compute the number of accidents on weekend/weekdays
4  # Report the two ratio.
5      DaysOfMonth <- unique(accidents$mydate)
6      DaysOfWeeks <- format(DaysOfMonth, "%w")
7      nDays       <- length(DaysOfMonth)
8      nWeekdays <- sum(DaysOfWeeks %in% 1:5)
9      nWeekends <- sum(DaysOfWeeks %in% c(0,6))
10
11     AccDaysOfWeek <- format(accidents$mydate, "%w")
12     nAccTotal     <- length(accidents$mydate)
13     nAccWeekdays <- sum(AccDaysOfWeek %in% 1:5)
14     nAccWeekends <- sum(AccDaysOfWeek %in% c(0,6))
15
16     rAccWdWe <- nAccWeekdays/nAccWeekends
17     rDaysWdWe <- nWeekdays/nWeekends
18
19     res <- data.frame(nDays,
20               nWeekdays,
```

Split - Apply - Combine - Advanced - Exercise VIII

```
1  testdata <- subset(accidents, accidents$month == 1)
2  dim(testdata)
3
4  mysummary(testdata)
5
6  > mysummary(testdata)
7         nDays    nWeekdays    nWeekends    nAccTotal nAccWeel
8     31.000000    21.000000    10.000000 10637.000000  7643.00
```

Split - Apply - Combine - Advanced - Exercise VIII

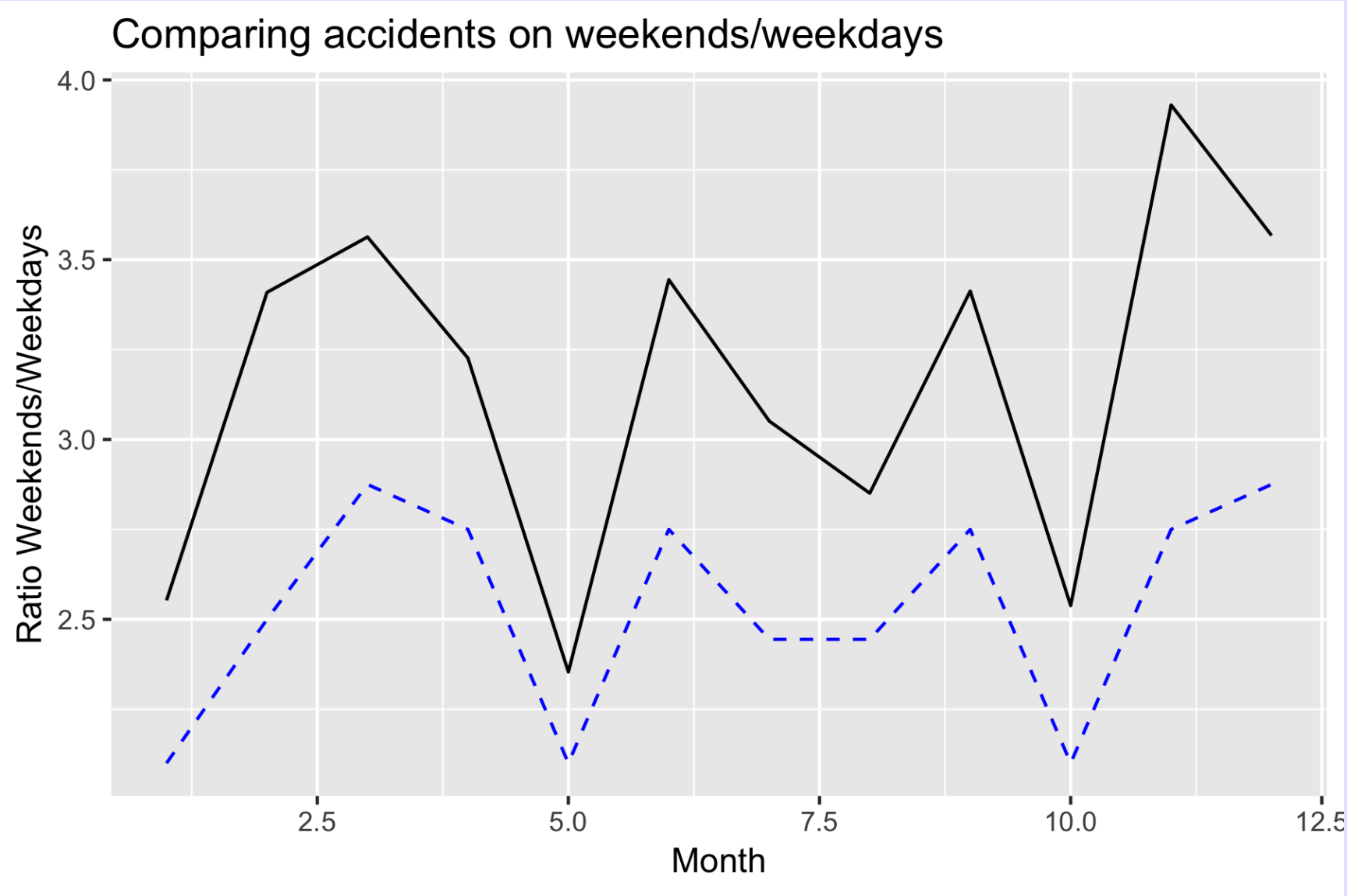
```
# using dplyr
results <-
   accidents %>%
    group_by(month) %>%
      do( mysummary(.) )
results
```

```
   month nDays nWeekdays nWeekends nAccTotal nAccWeekdays nA
1      1    31        21         10     10637         7643
2      2    28        20          8     11724         9065
3      3    31        23          8     13165        10280
4      4    30        22          8     12248         9350
5      5    31        21         10     13220         9278
6      6    30        22          8     13644        10574
7      7    31        22          9     13527        10188
8      8    31        22          9     13027         9644
9      9    30        22          8     13904        10753
10    10    31        21         10     14429        10351
11    11    30        22          8     14544        11594
12    12    31        23          8     10345         8080
```

Split - Apply - Combine - Advanced - Exercise VIII

```
1  newplot <- ggplot(data=results, aes(x=month, y=rDaysWdWe))+
2    ggtitle("Comparing accidens on weekends/weekdays")+
3    xlab("Month")+ylab("Ratio Weekends/Weekdays")+
4    geom_line(group=1, color="blue", linetype=2)+
5    geom_line(aes(y=rAccWdWe,group=1))
6  newplot
```

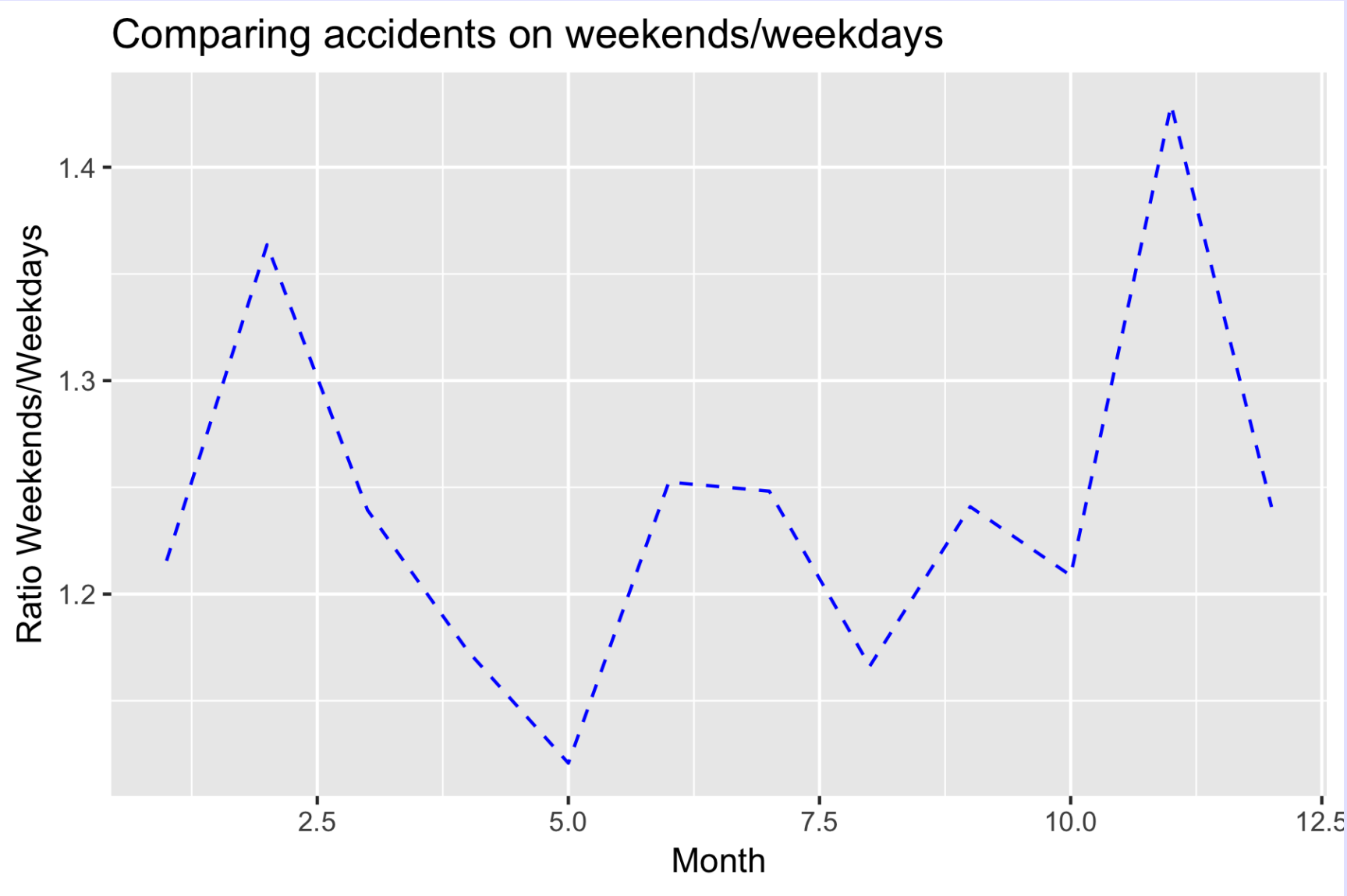
Split - Apply - Combine - Advanced - Exercise VIII



Split - Apply - Combine - Advanced - Exercise VIII

```
1  newplot <- ggplot(data=results, aes(x=month, y=rAccWdWe/rDay
2    ggtitle("Comparing accidens on weekends/weekdays")+
3    xlab("Month")+ylab("Ratio Weekends/Weekdays")+
4    geom_line(group=1, color="blue", linetype=2)
5  newplot
```

Split - Apply - Combine - Advanced - Exercise VIII



Comparing the two packages.

- Pro: for simple actions, some people find use of pipes to be more understandable
- Pro: must faster for very large data frames
- Pro: works same with directly access to data bases (see help)
- Con: non-standard evaluation will trip you up!
- Con: parallelizaton is more complicated than in the *plyr*
- Con: conflict between *plyr* and *dplyr*, esp. with *summarize()*