

R, Quick start to data analysis

Alex Shlemov Anton Korobeynikov

18 октября 2014 г.

Содержание

1 Справка, workspaces, запуск скриптов, пакеты	2
1.1 Справка	2
1.2 Переменные, рабочие пространства (workspaces), история команд, выход	2
1.3 Запуск скриптов	3
1.4 Пакеты	3
2 Вектора, матрицы, массивы	4
2.1 Вектора, основные операции	4
2.2 Вектора, доступ к элементам (subscripting, индексная техника)	8
2.2.1 Числовой вектор индексов	8
2.2.2 Логический вектор-маска	9
2.2.3 Строковый вектор имен	9
2.3 Матрицы и массивы	9
2.3.1 Создание и размерность матриц	9
2.3.2 Операции с матрицами	10
2.3.3 Функции для работы с матрицами	11
2.3.4 Многомерные массивы	13
2.4 Матрицы и массивы, доступ к элементам	13
2.4.1 Обращение как к вектору	13
2.4.2 Обращение к декартовому произведению измерений	13
2.4.3 Обращение по многомерному индексу	15
3 Списки	15
3.1 Создание списка, склейка, повторение	15
3.2 Обращение к элементам	16
3.2.1 Взятие подсписка	16
3.2.2 Взятие элемента	17
4 Материалы с занятия 3 октября	19
4.1 Toothgrowth	19
4.2 Графики residuals-vs-fitted	25
4.3 Линейная регрессия (Университеты)	34
4.3.1 Advertising, окончательный результат	47

5	Материалы с занятия 10 октября	55
5.1	LDA и tune	57
5.2	Default	62
5.3	Smarket	66
5.4	banknote	73
6	Материалы с занятия 17 октября	81
6.1	Bootstrap and CV	81
7	Рисование	106
7.1	Оценка двумерной плотности (kde2)	115
7.2	Boxplots, stripcharts and dotcharts (draft)	120
7.3	Densityplots, ecdfplots, qqmath (draft)	135

1 Справка, workspaces, запуск скриптов, пакеты

1.1 Справка

```
help(package = package_name) # Справка по пакету
> help(package = lattice)

?function_name # Справка по функции
> ?ls

?"keyword" # Справка по ключевому слову
> ?"for"
> ?"+"
> ?"["
> ?"[[<-
??pattern # Поиск по справке
> ??glm
apropos("pattern") # Возвращает найденные имена функций, подходящие под
шаблон
> apropos("GLM")
```

1.2 Переменные, рабочие пространства (workspaces), история команд, выход

```
ls() # Возвращает вектор из имен переменных в текущем scope, если запущен в
терминале, то возвращает имена переменных из рабочего workspace
ls(all.values = TRUE) # Возвращает все имена переменных текущего scope
(включая начинающиеся с .)

rm(varname) # Удаляет переменную. Имя без кавычек
rm(list = ls(all.values = TRUE)) # При вызове из терминала -- чистит
workspace
```

```

save.image(file = "workspace_file_name.rda") # сохраняет workspace (проще
    говоря, все переменные) в файл
load.image(file = "workspace_file_name.rda") # Загружает workspace из файла

history(max.show = Inf) # Показывает историю команд
savehistory(file = "history_file_name.R") # Сохраняет историю команд в файл

q() # Выход из R
q("no") # Выход из R без сохранения workspace (предпочтительнее)

```

1.3 Запуск скриптов

```
source("script_file_name.R") # Выполняет скрипт из файла
```

Также есть утилита `Rscript`, которая позволяет выполнить R-файл прямо из командной строки:

```
> Rscript script.R
```

Можно включить ее в shabang и сделать скрипт исполняемым файлом (в Unix):

```
script.R
```

```
#!/usr/bin/Rscript
```

```
args <- commandArgs(TRUE) # Получить аргументы командной строки в виде
    вектора строк
print(args)
```

после чего:

```
> chmod +x script.R
> ./script.R just command line args 3 14 15
[1] "just"      "command"   "line"      "args"      "3"        "14"        "15"
```

Если есть необходимость в детальном разборе аргументов командной строки, не нужно писать свой велюенид парсер, есть пакеты `getopt` и `optparse`.

1.4 Пакеты

```
library("package_name") # Подключает установленный пакет.
```

Кавычки можно опустить:

```
> library(lattice)
```

```
install.package("package_name") # Устанавливает пакет с зеркала CRAN
> install.packages("latticeExtra")
```

При первом запуске в сессии R предложит выбрать зеркало CRAN, достаточно выбрать “Cloud” (первое в списке). Обратите внимание, что в Unix пакеты скачиваются в виде исходников и собираются у Вас на машине, поэтому должен быть установлен компилятор C/C++/fortran и необходимые библиотеки (причем девелоперские версии, в пакетном менеджере они обычно имеют суффикс “-dev”, например “libfftw3-dev”). Под Windows пакеты скачиваются уже собранными.

При необходимости, можно установить сторонние пакеты из исходников. Для этого удобно пользоваться пакетом `devtools`:

```

install.packages("devtools")
library(devtools)
install_github("asl/rss")
# Аналогично:
install_git(...); install_bitbucket(...); install_url(...); install_local
(...)

```

Здесь пакет в любом случае будет собираться из исходников, под Windows нужно устанавливать и настраивать весь toolchain (msys + mingw + девелоперские либы). Под Unix могут понадобится некоторые стандартные утилиты типа curl (как правило, они уже установлены).

2 Вектора, матрицы, массивы

2.1 Вектора, основные операции

Начнем с того, что в R нет “скалярных” значений, любое скалярное значение (число, строка) это вектор длины 1. Вектора бывают следующих типов: `numeric`, `complex`, `logical`, `character`, т.е. числовые, комплексные, булевские и строковые. Числовые вектора делятся на `integer` и `double`, но это деление исключительно внутреннее — при делении или выходе из диапазона целые числа автоматически приводятся к вещественному типу.

Создание и простейшая работа с векторами:

```

> v <- 1:10
> print(v)
[1] 1 2 3 4 5 6 7 8 9 10
> 10:1
[1] 10 9 8 7 6 5 4 3 2 1
> seq(1, 10, 2)
[1] 1 3 5 7 9
> seq(from = 10, by = 5, length.out = 6)
[1] 10 15 20 25 30 35

# Создание “пустых” векторов
> v <- numeric(10)
> v
[1] 0 0 0 0 0 0 0 0 0 0
> b <- logical(10)
> b
[1] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
> cplx <- complex(10)
> cplx
[1] 0+0i 0+0i 0+0i 0+0i 0+0i 0+0i 0+0i 0+0i 0+0i 0+0i
> ch <- character(10)
> ch
[1] "" "" "" "" "" "" "" "" ""

# Прочитать элемент
> v[2]

```

```

[1] 0
> ch[3]
[1] ""
> b[4]
[1] FALSE
> cplx[5]
[1] 0+0i

# Записать элемент
> v[6] <- 42
> ch[7] <- "Hello"
> b[8] <- TRUE
> i <- 9
> 4i + 3 -> cplx[i]

# Повторения
> rep(1:3, 5) # Последовательная склейка
[1] 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3
> rep(1:3, each = 5) # И повтор каждого элемента
[1] 1 1 1 1 1 2 2 2 2 2 3 3 3 3 3
```

Конкатенация (склейка)

```
> c(1:5, 5:1, 3:4)
[1] 1 2 3 4 5 5 4 3 2 1 3 4
```

Немного служебных операций. Вывод:

```
> print(ch)
[1] ""      ""      ""      ""      ""      ""      "Hello"  ""      ""
[10] "
```

Если Вы работаете в командной сессии, то выводится результат каждой выполненной команды. Но если Вы проводите какие-то действия в цикле, в функции, в вызываемом по `source()` или `Rscript` скрипте, то желаемый вывод необходимо делать явно.

Кстати говоря, если Вы работаете в командной строке, то переменная `.Last.value` всегда содержит результат последней команды:

```
> 2 + 2
[1] 4
> print(.Last.value)
[1] 4
```

Summary:

```
> summary(1:10)
   Min. 1st Qu. Median     Mean 3rd Qu.    Max.
   1.00    3.25    5.50    5.50    7.75   10.00
```

Вообще `summary()` (как, кстати, и `print()`) — это полиморфные функции, для каждого типа объекта они определены по-своему. Для числовых векторов `summary()` выводит квантили и среднее. Незамысловато, но бывает полезно.

Длина вектора:

```

> length(v)
[1] 10
> length(v) <- 5
> v
[1] 0 0 0 0 0
> length(v) <- 10
> v
[1] 0 0 0 0 0 NA NA NA NA NA

```

Функция `length()` работает и на присваивание. При попытке увеличить длину вектора новые элементы получают значение `NA`, т.е. пропущенное значение.

Тип вектора:

```

> mode(v) # Логический тип (mode)
[1] "numeric"
> storage.mode(v) # Хранимый тип. Нужен редко, в основном, если хочется
   передать указатель на объект "наружу"
[1] "double"

```

Обе функции работают на присваивание, изменяя тип объекта.

Также можно совершить приведение типа с помощью функций `as.whatever()`:

```

> as.character(10)
[1] "10"
> as.logical(10)
[1] TRUE
> as.numeric("33.5")
[1] 33.5
> as.integer("33.5")
[1] 33
> as.integer(33.5)
[1] 33

```

Все стандартные операции с векторами векторизованы, т.е. выполняются поэлементно:

```

> 1:10 + 10:1
[1] 11 11 11 11 11 11 11 11 11 11
> sin(1:10)
[1] 0.8414710 0.9092974 0.1411200 -0.7568025 -0.9589243 -0.2794155
[7] 0.6569866 0.9893582 0.4121185 -0.5440211

```

При этом если в бинарной операции встречаются вектора неодинаковой длины, то используются так называемое переписывание (recycling), вектор меньшей длины автоматически повторяется нужное число раз:

```

> 1:10 + 1:5
[1] 2 4 6 8 10 7 9 11 13 15

```

При этом, если длина меньшего вектора не является делителем длины большей, будет выведено предупреждение (warning):

```

> 1:10 + 1:3
[1] 2 4 6 5 7 9 8 10 12 11

```

Warning message:

In 1:10 + 1:3 :

longer object length is not a multiple of shorter object length

Обычно меньший вектор имеет длину 1 и такой проблемы не возникает:

```
> (1:10)^2  
[1] 1 4 9 16 25 36 49 64 81 100
```

Кстати, степень имеет более высокий приоритет, чем ::

```
> 1:3^2  
[1] 1 2 3 4 5 6 7 8 9
```

Полезные векторизованные функции:

```
a + b, a - b, a * b, a / b # 4 действия арифметики  
a ^ b # степень  
a %% b, a %/% b # целочисленное деление и взятие остатка
```

```
exp(x), log(x) # экспонента и логарифм
```

```
abs(x) # Модуль
```

```
Re(z), Im(z), Conj(z), Mod(z), Arg(z) # вещественная и мнимая часть,  
комплексное сопряжение, модуль и аргумент
```

```
cos(x), sin(x), tan(x), acos(x), asin(x), atan(x), atan2(y, x) #  
Тригонометрия
```

```
x == y, x != y, x > y, x >= y, etc # поэлементные сравнения  
> 1:10 > 5
```

```
[1] FALSE FALSE FALSE FALSE FALSE TRUE TRUE TRUE TRUE TRUE
```

```
x & y, x | y, xor(x, y) # булевские поэлементные операции  
x && y, x || y # булевские операции для векторов длины 1, вычисляемые по  
короткой схеме
```

Агрегирующие функции. Наряду с поэлементной векторизацией (любители функционального программирования назвали бы ее “map”) есть функции, сопоставляющие вектору единичное значение (любители ФП назвали бы это “reduce”). Вот примеры таких функций:

```
sum(x), prod(x) # Сумма и произведение всех элементов  
max(x), min(x), which.max(), which.min() # Максимум-минимум и индекс  
максимального и минимального элемента
```

```
mean(), sd(), cov(), cor(), median(), mad(), quantile() # Статистические  
функции
```

```
all(x), any(x) # Логические функции, возвращают TRUE, если все (или хотя бы  
один) из элементов вектора истина
```

2.2 Вектора, доступ к элементам (subscripting, индексная техника)

Доступ к элементам вектора осуществляется с помощью оператора “[” (“subscript”). Доступ работает как на чтение, так и на запись:

```
x[?]  
x[?] <- y
```

Чтение возвращает подвектор (возможно, что пустой). При записи подвектор перезаписывается значениями из вектора, стоящего в правой части (y). Если длины перезаписываемого подвектора и правой части не совпадают, применяется переписывание (если количество заменяемых значений не делится на количество новых, то выводится предупреждение).

Что может стоять внутри “[]”?

2.2.1 Числовой вектор индексов

Все нецелые значения приводятся к целым (отбрасывается дробная часть). Нули отбрасываются. Для положительных индексов возвращаются соответствующие элементы (нумерация от единицы!!!):

```
> v <- c("a", "b", "c", "d", "e", "f", "g", "h")  
> v[c(1, 3, 5.9)]  
[1] "a" "c" "e"  
> v[c(1, 3, 5.9)] <- "X"  
> v  
[1] "X" "b" "X" "d" "X" "f" "g" "h"
```

Для отрицательных возвращаются все элементы, кроме названных:

```
> v[-c(2, 4, 7.7)]  
[1] "X" "X" "X" "f" "h"  
> v[-c(2, 4, 7.7)] <- Y  
Error: object 'Y' not found  
> v[-c(2, 4, 7.7)] <- "Y"  
> v  
[1] "Y" "b" "Y" "d" "Y" "Y" "g" "Y"
```

Смешивать отрицательные и положительные индексы нельзя. На чтение положительные индексы можно дублировать:

```
> v  
[1] "Y" "b" "Y" "d" "Y" "Y" "g" "Y"  
> v[c(1, 1, 1, 2)]  
[1] "Y" "Y" "Y" "b"
```

На запись тоже можно, но в таком случае элемент с повторенным индексом будет перезаписан несколько раз и в итоге в нем окажется последний записанный элемент:¹

```
> v[c(1, 1, 1)] <- c("X", "Y", "Z")  
> v  
[1] "Z" "b" "c"
```

¹Лично я считаю, что использовать повторные индексы на запись — очень скверная идея.

2.2.2 Логический вектор-маска

Выбираются элементы, соответствующие TRUE. Если вектор недостаточной длины, используется переписывание (если длина маски не делит длину вектора, то выведется соответствующее предупреждение). Если вектор-маска больше длины вектора, то вектор удлиняется до необходимой длины и дополняется пропусками (NA).

```
> v <- 1:10
> v[c(TRUE, FALSE)] # Выбрать четные элементы
[1] 1 3 5 7 9
> v[c(TRUE, FALSE)] <- 42 # Заменить четные элементы
> v
[1] 42 2 42 4 42 6 42 8 42 10
```

В основном, в качестве логической маски используются выражения-“запросы”:

```
> v[v > 6] <- 0 # Заменить элементы > 6
> v
[1] 0 2 0 4 0 6 0 0 0 0
```

Тут нет никакой магии — `v > 6` возвращает логический вектор.

2.2.3 Строковый вектор имен

Для того, чтобы обращаться к элементам вектора по именам, необходимо эти имена назначить. У каждого вектора есть возможность установить атрибут `names` — строковый вектор такой же длины, как и сам вектор:

```
> v <- 1:3
> names(v) <- c("a", "b", "c")
> v
a b c
1 2 3
> names(v)
[1] "a" "b" "c"
```

Вектор стал именованным. Теперь если передать в качестве индекса строковый вектор, будут выбраны соответствующие элементы:

```
> v[c("a", "b")]
a b
1 2
> v[c("a", "b")] <- 42
> v
a b c
42 42 3
```

2.3 Матрицы и массивы

2.3.1 Создание и размерность матриц

Матрица создается с помощью одноименной команды:

```

> m <- matrix(1:9, 3, 3)
> m
     [,1] [,2] [,3]
[1,]    1    4    7
[2,]    2    5    8
[3,]    3    6    9

```

Матрицы в R представляют собой вектор (с разверткой FORTRAN-style, т.е. по столбцам) со специальным атрибутом размерности:

```

> dim(m)
[1] 3 3
> length(m)
[1] 9
> dim(m) <- c(1, 9)
> m
     [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9]
[1,]    1    2    3    4    5    6    7    8    9

```

Как видите, атрибут доступен на запись, единственное, необходимо, чтобы `prod(dim(x)) = length(x)`.

Также есть функции `nrow()` и `ncol()`, возвращают число строк и столбцов соответственно.

2.3.2 Операции с матрицами

Так как матрицы являются векторами, для них можно делать те же операции, что и для векторов; при этом размерность будет сохраняться:

```

> m <- matrix(1:9, 3, 3)
> sin(m)
     [,1]      [,2]      [,3]
[1,] 0.8414710 -0.7568025 0.6569866
[2,] 0.9092974 -0.9589243 0.9893582
[3,] 0.1411200 -0.2794155 0.4121185
> m + m
     [,1] [,2] [,3]
[1,]    2    8   14
[2,]    4   10   16
[3,]    6   12   18
> m ^ 2
     [,1] [,2] [,3]
[1,]    1   16   49
[2,]    4   25   64
[3,]    9   36   81
> m * 2
     [,1] [,2] [,3]
[1,]    2    8   14
[2,]    4   10   16
[3,]    6   12   18
> m * m

```

```

[,1] [,2] [,3]
[1,]    1    16   49
[2,]    4    25   64
[3,]    9    36   81
> m > 10
      [,1] [,2] [,3]
[1,] FALSE FALSE FALSE
[2,] FALSE FALSE FALSE
[3,] FALSE FALSE FALSE
> m > 5
      [,1] [,2] [,3]
[1,] FALSE FALSE TRUE
[2,] FALSE FALSE TRUE
[3,] FALSE  TRUE TRUE

```

Обратите внимание, что произведение матриц — поэлементное. Если мы хотим получить обычное операторное произведение, следует использовать `%*%`:

```

> m %*% m
      [,1] [,2] [,3]
[1,]    30   66  102
[2,]    36   81  126
[3,]    42   96  150

```

Вектор без атрибута размерности считается вектор-столбцом, но при умножении вектора на матрицу слева вектор автоматически транспонируется:

```

> m
      [,1] [,2] [,3]
[1,]    1    4    7
[2,]    2    5    8
[3,]    3    6    9
> m %*% 1:3
      [,1]
[1,]    30
[2,]    36
[3,]    42
> 1:3 %*% m
      [,1] [,2] [,3]
[1,]    14   32   50

```

Обратите внимание, что умножение `*` это поэлементное умножение каждого столбца на вектор:

```

> m * 1:3
      [,1] [,2] [,3]
[1,]    1    4    7
[2,]    4   10   16
[3,]    9   18   27

```

2.3.3 Функции для работы с матрицами

:

```

solve(m) # обратная матрица
solve(m, y) #  $m^{-1}y$ , но вычисляется устойчивее
t(m) # транспонирование

qr(m), eigen(m), svd(m), chol(m) # классические матричные разложения (QR,
EVD, SVD и разложение Холецкого)

crossprod(x, y) #  $x^T y$ , но вычисляется немного быстрее
tcrossprod(x, y) #  $xy^T$ , аналогично
crossprod(x), tcrossprod(x) # умножение саму на себя:  $x^T x$  и  $xx^T$ 

diag(m) # для матрицы, возвращает вектор главной диагонали, при этом
доступна на запись
> m <- matrix(1:9, 3, 3)
> diag(m)
[1] 1 5 9
> diag(m) <- -diag(m)
> m
 [,1] [,2] [,3]
[1,] -1     4     7
[2,]  2    -5     8
[3,]  3     6    -9
diag(n) # для числа -- возвращает единичную матрицу порядка n
> diag(3)
 [,1] [,2] [,3]
[1,] 1     0     0
[2,] 0     1     0
[3,] 0     0     1

```

Слейка матриц:

```

cbind(a, b, c, ...) # Слейка матриц по столбцам: [a : b : c : ...]
rbind(a, b, c, ...) # Слейка матриц по строкам (вертикально)

```

Если вектор (не матрицу) передать на вход `cbind()`, то он будет рассматриваться как столбец, а если `rbind()` — то как строка. При этом для векторов и матриц работают правила переписывания.

Сумма и среднее по строкам и столбцам:

```

> m <- matrix(1:9, 3, 3)
> rowMeans(m)
[1] 4 5 6
> colMeans(m)
[1] 2 5 8
> rowSums(m)
[1] 12 15 18
> colSums(m)
[1] 6 15 24

```

2.3.4 Многомерные массивы

Также, кроме матриц присутствуют и многомерные массивы (тензоры) `array()`:

```
> a <- array(1:8, dim = c(2, 2, 2))
> a
, , 1
```

```
[,1] [,2]
[1,] 1 3
[2,] 2 4
```

```
, , 2
```

```
[,1] [,2]
[1,] 5 7
[2,] 6 8
```

2.4 Матрицы и массивы, доступ к элементам

Обсудим обращение к элементам матриц и многомерных массивов. Аналогично векторам, обращение возможно как на чтение, так и на запись

2.4.1 Обращение как к вектору

И матрица, и массив являются вектором, следовательно, для них работают те же методы индексирования, что и для векторов, при этом. напоминаю, матрица укладывается в вектор по столбцам. На практике, пожалуй, из этого может быть полезна только техника “логических запросов” типа:

```
m[m > 10]
m[m < 0] <- 0
```

2.4.2 Обращение к декартовому произведению измерений

Для обращения к матрице можно использовать двухиндексную технику (а для обращения к массивам — r -индексную, где r — количество измерений):

```
m[i, j]
a[i, j, k]
```

где i , j , k могут быть числовыми, логическими или строковыми векторами. Результатом будет подмассив той же структуры (подвыборка произойдет независимо по всем измерениям).

```
> m <- matrix(1:9, 3, 3)
> m
[,1] [,2] [,3]
[1,] 1 4 7
[2,] 2 5 8
[3,] 3 6 9
> m[c(TRUE, FALSE, TRUE), -1] # Выбрать 1 и 3 строки и отбросить 1 столбец
```

```
[,1] [,2]
[1,]    4    7
[2,]    6    9
```

Чтобы иметь возможность обращаться к строкам и столбцам матрицы по именам, нужно задать атрибуты `colnames` и `rownames` (а в случае массива — атрибут `dimnames`):

```
> m <- matrix(1:9, 3, 3)
> rownames(m) <- c("a", "b", "c")
> colnames(m) <- c("x", "y", "z")
> m[c("a", "c"), c("y", "y", "x")]
   y  y  x
a 4 4 1
c 6 6 3

> a <- array(1:8, dim = c(2, 2, 2))
> dimnames(a) <- list(c("a", "b"), c("i", "j"), c("x", "y"))
> a
, , x

   i  j
a 1 3
b 2 4

, , y

   i  j
a 5 7
b 6 8
> a["a", "j", "y"]
[1] 7
```

Нужно отметить две тонкости. Во-первых, один или несколько индексов можно опускать, это будет означать выбор всего диапазона. Во-вторых, если в результате выбора полученный массив будет иметь меньшую размерность, чем исходный (например, выбираем строку из матрицы), то вырожденные измерения автоматически “схлопнутся” (`drop`):

```
m <- matrix(1:9, 3, 3)
m

##      [,1] [,2] [,3]
## [1,]    1    4    7
## [2,]    2    5    8
## [3,]    3    6    9

m[1, ]
## [1] 1 4 7

m[, 1]
## [1] 1 2 3
```

В большинстве случаев это удобно: когда мы извлекаем строку или столбец, более приятно получать вектор, а не длинную матрицу. Но иногда требуется, чтобы подмассив имел строго такую же размерность, как и исходный массив. Чтобы избежать склонения и получить массив той же размерности нужно явно указать:

```
m <- matrix(1:9, 3, 3)
m[1, , drop = FALSE]

##      [,1] [,2] [,3]
## [1,]    1    4    7

m[, 1, drop = FALSE]

##      [,1]
## [1,]    1
## [2,]    2
## [3,]    3
```

2.4.3 Обращение по многомерному индексу

Можно передать в [] матрицу из r столбцов и n строк, где r — число измерений (2 для матрицы). В результате каждая строка будет рассматриваться как набор координат выбираемого элемента и результатом будет вектор длины n :

```
> m <- matrix(1:9, 3, 3)
> m[cbind(1:ncol(m), ncol(m):1)] # Антидиагональ
[1] 7 5 3
```

3 Списки

Список — это вектор, который может хранить элементы различных типов. В отличие от Python, нет возможности создать рекурсивный список (так как копирование всегда происходит по значению).

3.1 Создание списка, склейка, повторение

```
l <- list(a = 1, b = "string", f = q) # Может хранить объекты разных типов
l

## $a
## [1] 1
##
## $b
## [1] "string"
##
## $f
## function (save = "default", status = 0, runLast = TRUE)
```

```

## .Internal.quit(save, status, runLast)
## <bytecode: 0x232af8>
## <environment: namespace:base>

l <- list(a = 1, 2)  # Не обязательно все элементы должны иметь имена
l

## $a
## [1] 1
##
## [[2]]
## [1] 2

```

```

l <- as.list(1:3)
l
l1 <- list(1, "A")
l2 <- list("b", 10)
c(l1, l2)  # Списки можно склеивать
rep(l1, 5)  # И повторять

```

3.2 Обращение к элементам

3.2.1 Взятие подсписка

Для списков оператор [работает также, как и для векторов, только возвращается не подвектор, а подсписок:

```

l <- list(a = 1, b = "string", d = TRUE)
l[1:2]

## $a
## [1] 1
##
## $b
## [1] "string"

l[-2]

## $a
## [1] 1
##
## $d
## [1] TRUE

l[c("a", "b")]

```

```

## $a
## [1] 1
##
## $b
## [1] "string"

l[1:2] <- list(5, "char")
l

## $a
## [1] 5
##
## $b
## [1] "char"
##
## $d
## [1] TRUE

```

3.2.2 Взятие элемента

Оператор [[позволяет обратиться к элементу:

```

l[[1]]

## [1] 5

l[["d"]] <- list(42)
l

## $a
## [1] 5
##
## $b
## [1] "char"
##
## $d
## $d[[1]]
## [1] 42

```

Также к элементам списка можно обращаться через оператор \$:

```

ll <- list(a = 1, b = 2, "ccc")
ll$a

## [1] 1

ll$b <- 42
ll$c # При чтении достаточно уникального префикса

```

```

## NULL

l1

## $a
## [1] 1
##
## $b
## [1] 42
##
## [[3]]
## [1] "ccc"

l1$c <- 42 # А при записи будет создан элемент с переданным именем
l1

## $a
## [1] 1
##
## $b
## [1] 42
##
## [[3]]
## [1] "ccc"
##
## $c
## [1] 42

```

Присваивание элементу значения NULL удаляет элемент:

```

l

## $a
## [1] 5
##
## $b
## [1] "char"
##
## $d
## $d[[1]]
## [1] 42

l[[1]] <- NULL
l$d <- NULL
l

## $b
## [1] "char"

```

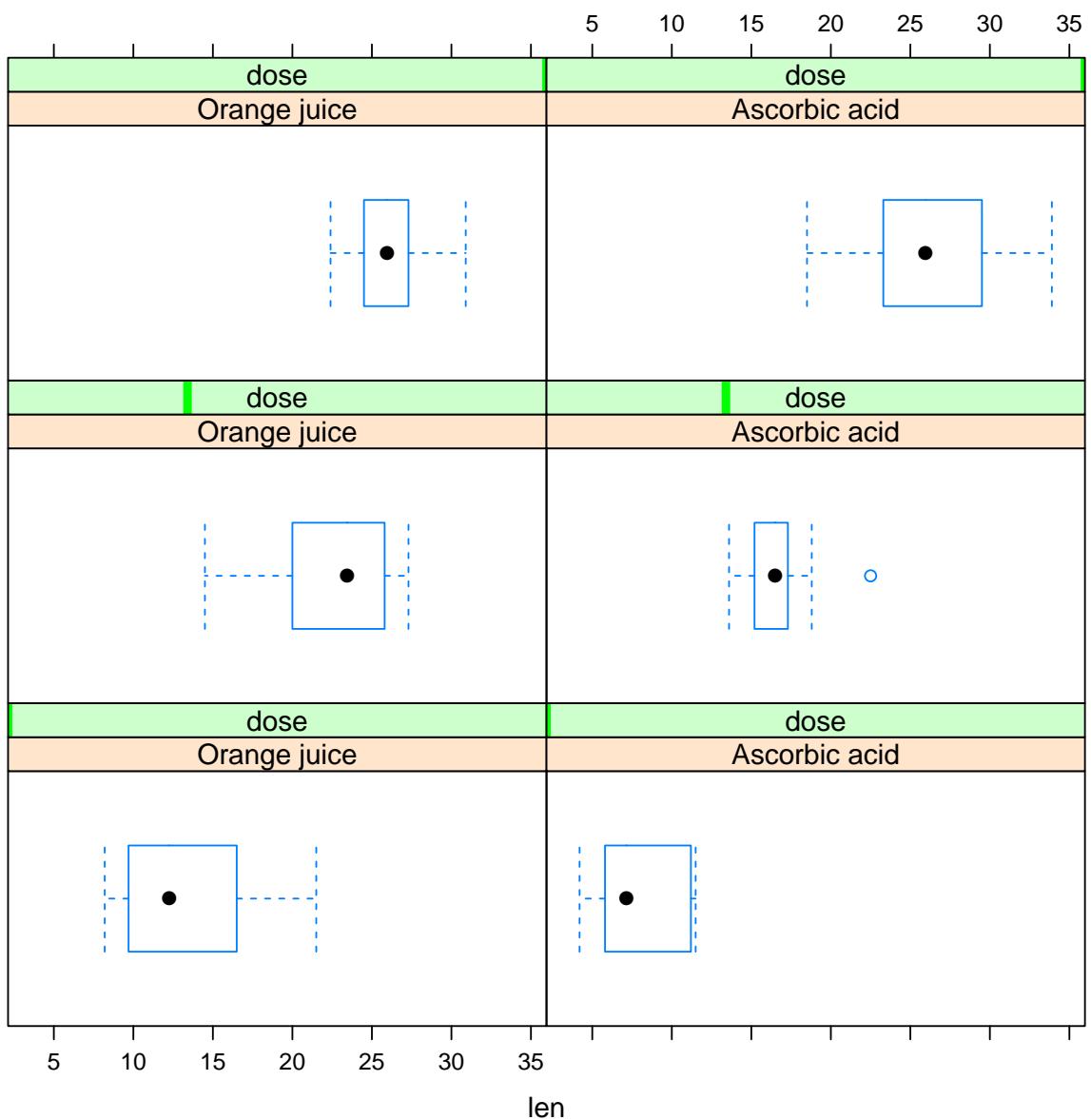
Если Вам по каким-то причинам надо положить NULL в список, то это делается так:

```
1 [1] <- list(NULL)
```

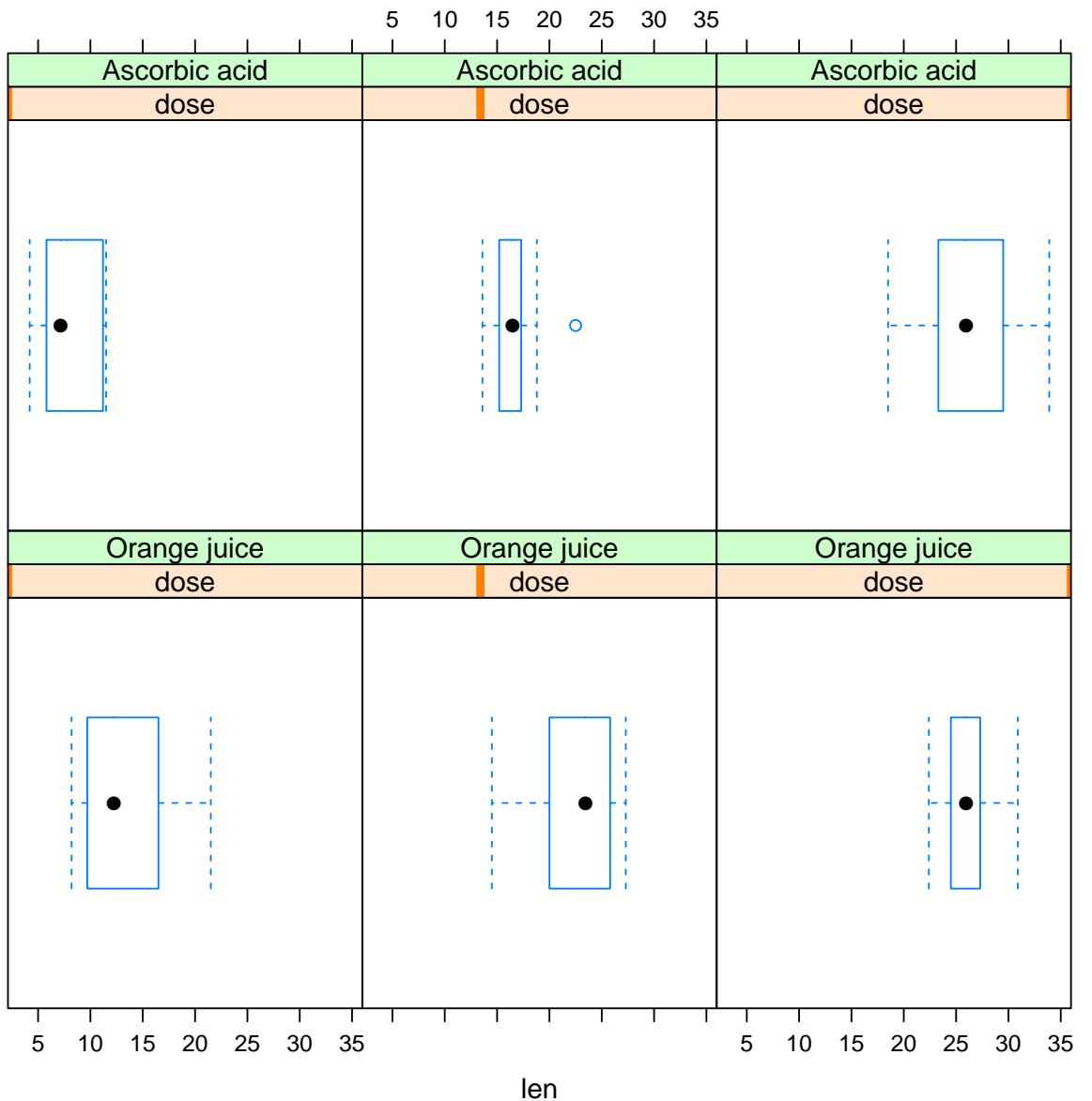
4 Материалы с занятия 3 октября

4.1 Toothgrowth

```
library(lattice)
library(latticeExtra)
library(MASS)
tooth <- read.table("toothgrowth.txt")
tooth$supp <- factor(tooth$supp, labels = c("Orange juice",
    "Ascorbic acid"))
tooth$supp <- factor(tooth$supp, levels = c("Orange juice",
    "Ascorbic acid"))
bwplot(~len | supp * dose, data = tooth)
```



```
bwplot(~len | dose * supp, data = tooth)
```



```

contrasts(tooth$supp)

##          Ascorbic acid
## Orange juice      0
## Ascorbic acid     1

contrasts(tooth$supp) <- contr.sum
contrasts(tooth$supp)

## [,1]
## Orange juice    1
## Ascorbic acid   -1

l <- lm(len ~ supp + dose, data = tooth)
summary(l)

```

```

## 
## Call:
## lm(formula = len ~ supp + dose, data = tooth)
## 
## Residuals:
##    Min     1Q Median     3Q    Max 
## -6.600 -3.700  0.373  2.116  8.800 
## 
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)    
## (Intercept) 7.4225     1.1599   6.399 3.17e-08 ***
## supp1       1.8500     0.5468   3.383  0.0013 **  
## dose        9.7636     0.8768  11.135 6.31e-16 ***
## ---        
## Signif. codes:
## 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1 
## 
## Residual standard error: 4.236 on 57 degrees of freedom
## Multiple R-squared:  0.7038, Adjusted R-squared:  0.6934 
## F-statistic: 67.72 on 2 and 57 DF,  p-value: 8.716e-16 

l <- lm(len ~ supp * dose, data = tooth)
summary(l)

## 
## Call:
## lm(formula = len ~ supp * dose, data = tooth)
## 
## Residuals:
##    Min     1Q Median     3Q    Max 
## -8.2264 -2.8462  0.0504  2.2893  7.9386 
## 
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)    
## (Intercept) 7.4225     1.1182   6.638 1.37e-08 *** 
## supp1       4.1275     1.1182   3.691 0.000507 ***  
## dose        9.7636     0.8453  11.551 < 2e-16 ***  
## supp1:dose -1.9521     0.8453  -2.309 0.024631 *   
## ---        
## Signif. codes:
## 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1 
## 
## Residual standard error: 4.083 on 56 degrees of freedom
## Multiple R-squared:  0.7296, Adjusted R-squared:  0.7151 
## F-statistic: 50.36 on 3 and 56 DF,  p-value: 6.521e-16 

tooth$dose <- factor(tooth$dose, ordered = TRUE)
contrasts(tooth$dose)

```

```

##          .L          .Q
## [1,] -7.071068e-01  0.4082483
## [2,] -7.850462e-17 -0.8164966
## [3,]  7.071068e-01  0.4082483

contrasts(tooth$dose) <- contr.helmert
contrasts(tooth$dose)

##      [,1] [,2]
## 0.5   -1   -1
## 1     1   -1
## 2     0    2

l <- lm(len ~ supp * dose, data = tooth)
summary(l)

##
## Call:
## lm(formula = len ~ supp * dose, data = tooth)
##
## Residuals:
##    Min     1Q Median     3Q    Max
## -8.20  -2.72 -0.27  2.65  8.27
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) 18.8133    0.4688  40.130 < 2e-16 ***
## supp1       1.8500    0.4688   3.946 0.000231 ***
## dose1       4.5650    0.5742   7.951 1.19e-10 ***
## dose2       3.6433    0.3315  10.990 2.17e-15 ***
## supp1:dose1 0.1700    0.5742   0.296 0.768308
## supp1:dose2 -0.9450    0.3315  -2.851 0.006166 **
## ---
## Signif. codes:
## 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 3.631 on 54 degrees of freedom
## Multiple R-squared:  0.7937, Adjusted R-squared:  0.7746
## F-statistic: 41.56 on 5 and 54 DF,  p-value: < 2.2e-16

stepAIC(l)

## Start:  AIC=160.43
## len ~ supp * dose
##
##          Df Sum of Sq    RSS    AIC
## <none>              712.11 160.43
## - supp:dose  2     108.32 820.43 164.93
## 
```

```

## Call:
## lm(formula = len ~ supp * dose, data = tooth)
##
## Coefficients:
## (Intercept)      supp1      dose1      dose2
##           18.813      1.850      4.565      3.643
## supp1:dose1  supp1:dose2
##           0.170     -0.945

l.lin <- lm(len ~ supp + dose, data = tooth)
# the smaller AIC/BIC, the better the fit
AIC(l, l.lin)

##          df      AIC
## l       7 332.7056
## l.lin  5 337.2013

BIC(l, l.lin)

##          df      BIC
## l       7 347.366
## l.lin  5 347.673

anova(l, l.lin)

## Analysis of Variance Table
##
## Model 1: len ~ supp * dose
## Model 2: len ~ supp + dose
##   Res.Df   RSS Df Sum of Sq    F  Pr(>F)
## 1     54 712.11
## 2     56 820.43 -2   -108.32 4.107 0.02186 *
## ---
## Signif. codes:
## 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

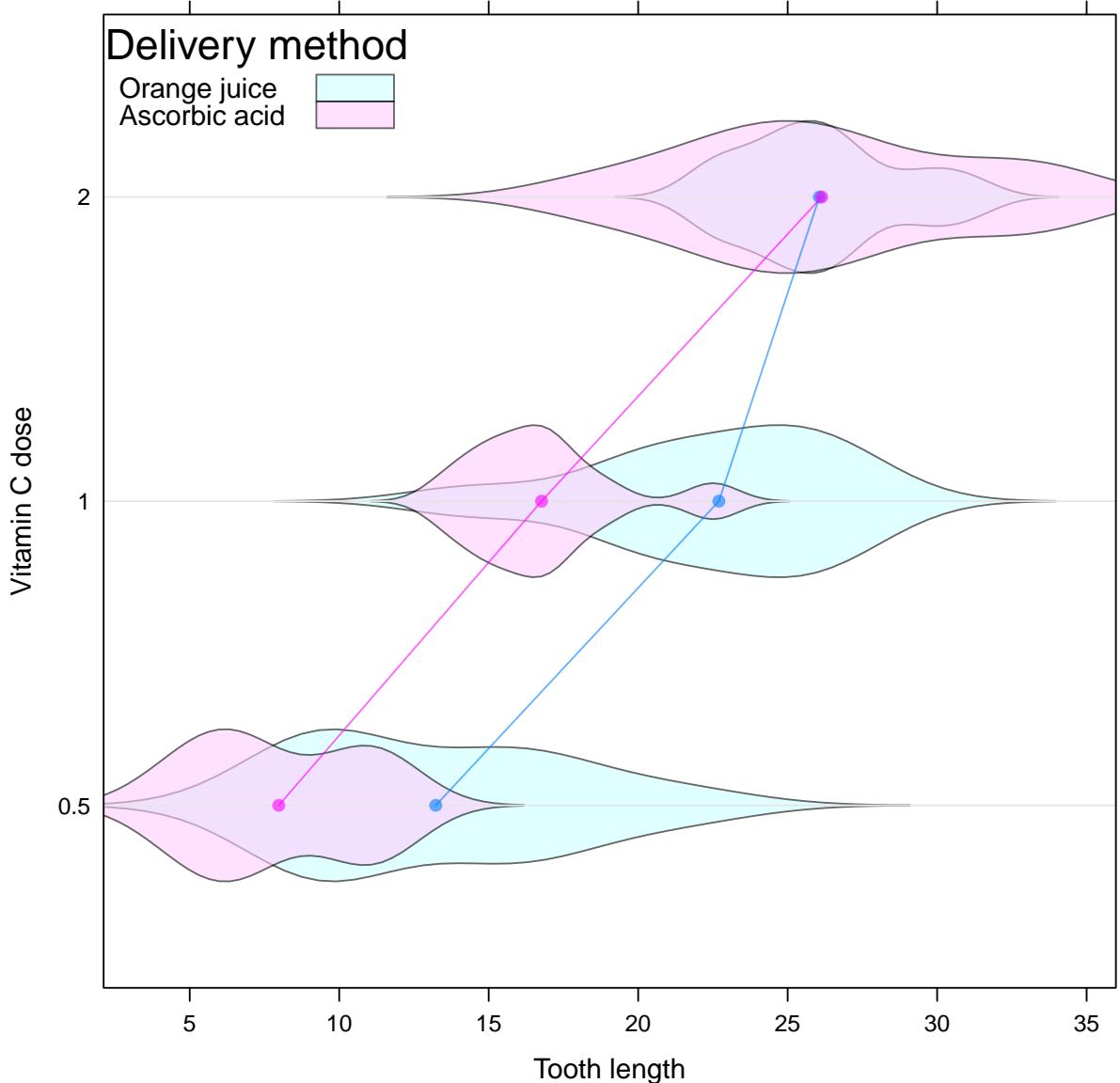
tooth.agg <- aggregate(subset(tooth, select = len),
  list(supp = tooth$supp, dose = tooth$dose),
  mean)
dp <- dotplot(factor(dose) ~ len, groups = supp,
  data = tooth.agg, auto.key = list(title = "Delivery",
    corner = c(0, 1)), type = "b", xlab = "mean(tooth length)",
    ylab = "Vitamin C dose", par.settings = simpleTheme(pch = 19))
vp <- bwplot(factor(dose) ~ len, groups = supp,
  data = tooth, panel = function(...) {
    panel.superpose(..., col = trellis.par.get("superpose.polygon")$col,
      panel.groups = panel.violin)
  }, auto.key = list(title = "Delivery method",
    corner = c(0, 1), points = FALSE,

```

```

    lines = FALSE, rectangles = TRUE),
xlab = "Tooth length", ylab = "Vitamin C dose",
par.settings = simpleTheme(alpha = 0.6,
    pch = 19))
vp + dp

```



4.2 Графики residuals-vs-fitted

```

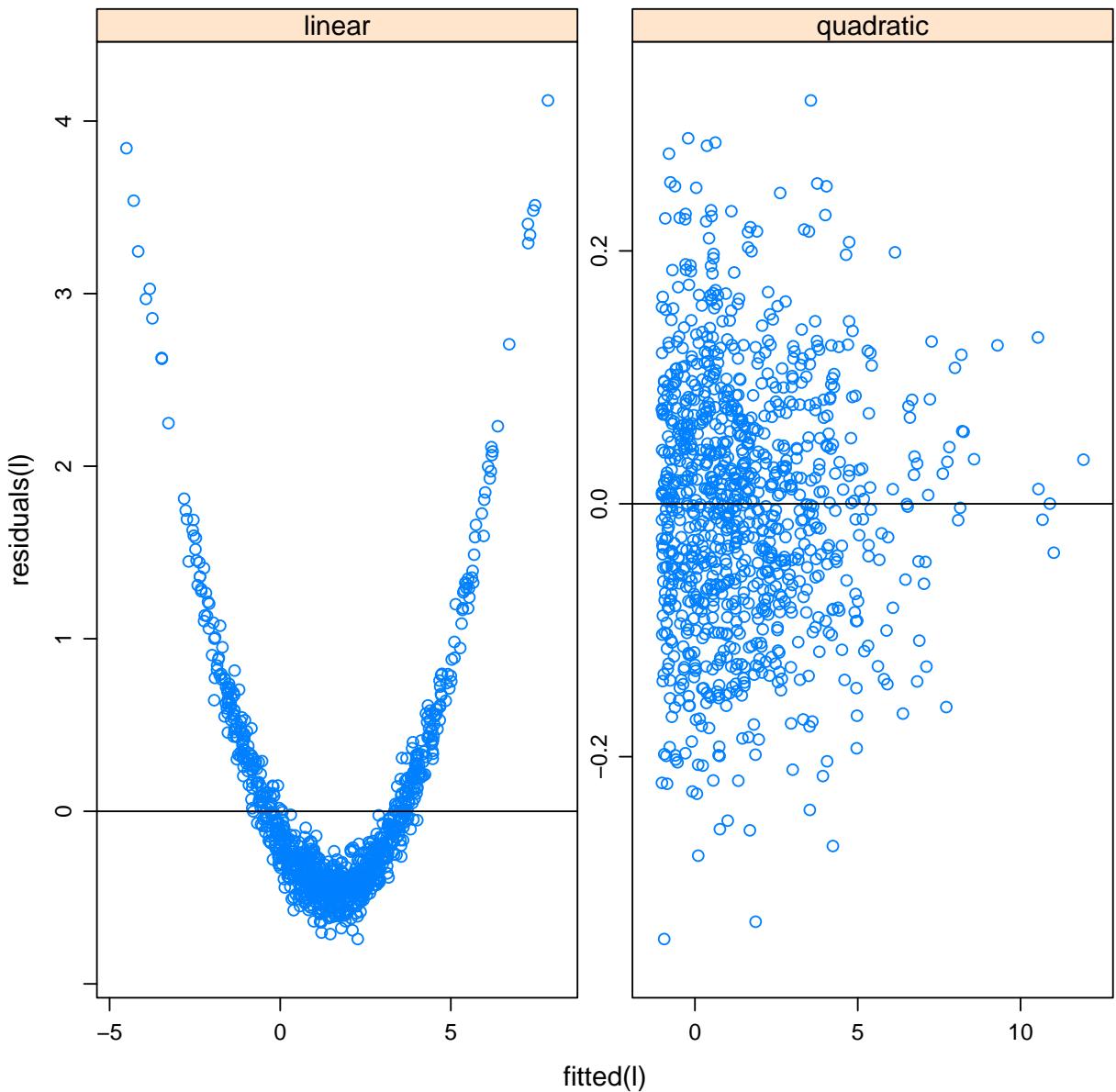
library(lattice)
library(latticeExtra)
library(MASS)
panel <- function(...) {
  panel.xyplot(...)
}

```

```
    panel.lmline(...)

}

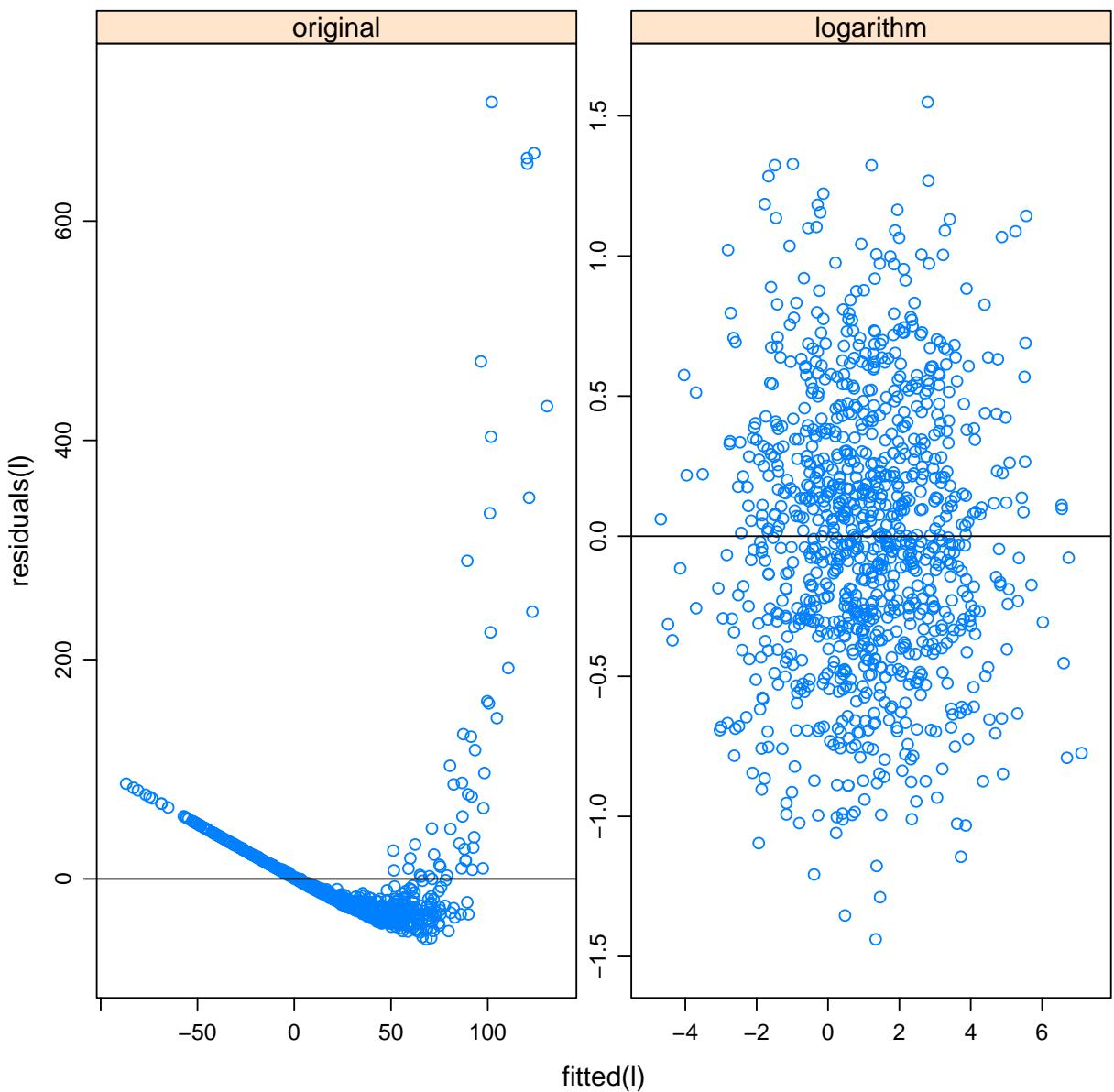
N <- 1000
x <- rnorm(N)
beta0 <- 1
beta1 <- 2
beta1_2 <- 0.5
y <- beta0 + beta1 * x + beta1_2 * x^2 +
    rnorm(N, sd = 0.1)
df <- data.frame(y = y, x = x)
l <- lm(y ~ x, data = df)
l2 <- lm(y ~ poly(x, degree = 2), data = df)
p1 <- xyplot(residuals(l) ~ fitted(l), panel = panel)
p2 <- xyplot(residuals(l2) ~ fitted(l2),
    panel = panel)
plot(c(linear = p1, quadratic = p2))
```



```

y <- exp(beta0 + beta1 * x + rnorm(N, sd = 0.5))
df <- data.frame(y = y, x = x)
l <- lm(y ~ x, data = df)
l2 <- lm(log(y) ~ x, data = df)
p1 <- xyplot(residuals(l) ~ fitted(l), panel = panel)
p2 <- xyplot(residuals(l2) ~ fitted(l2),
  panel = panel)
plot(c(original = p1, logarithm = p2))

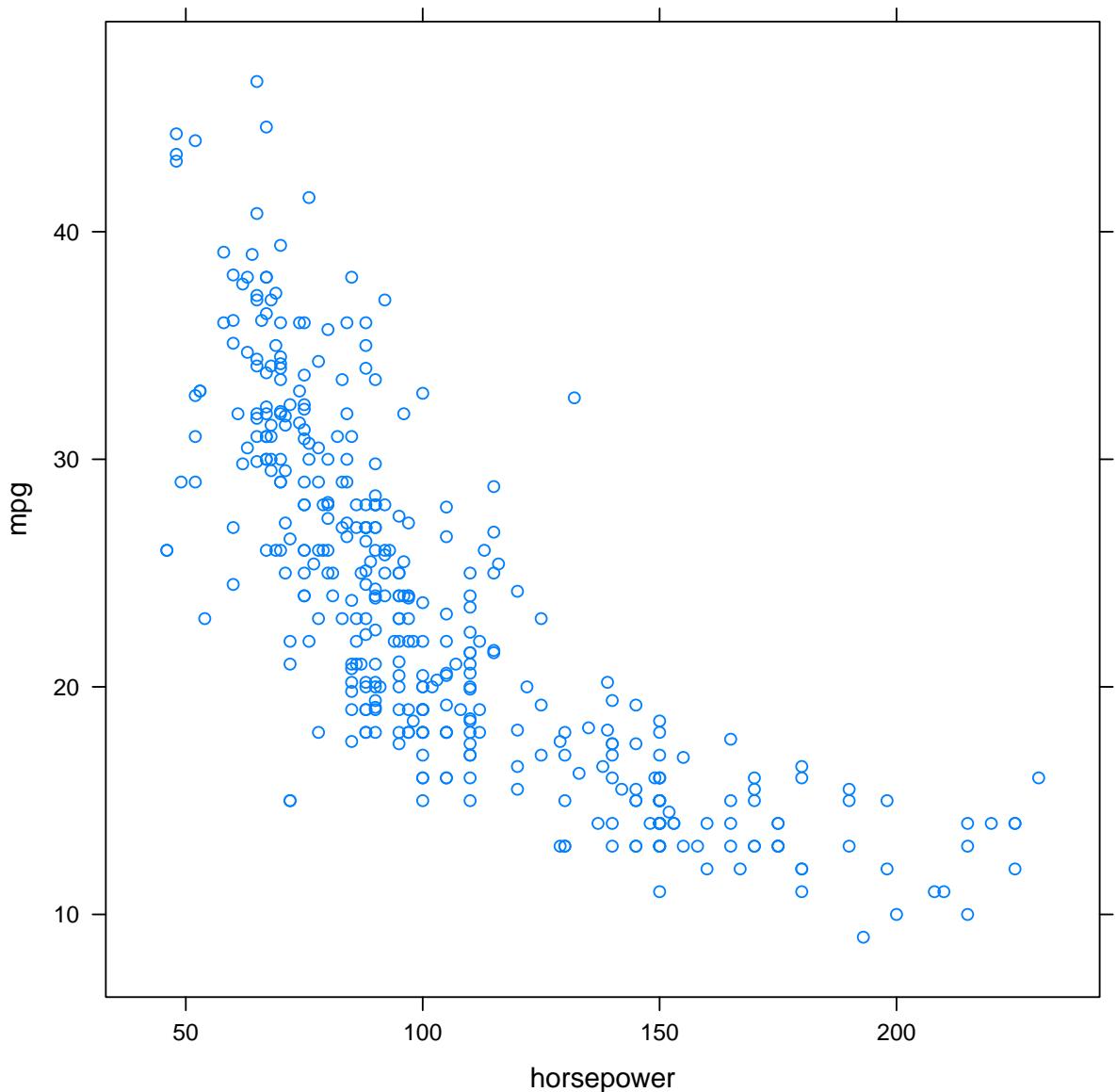
```



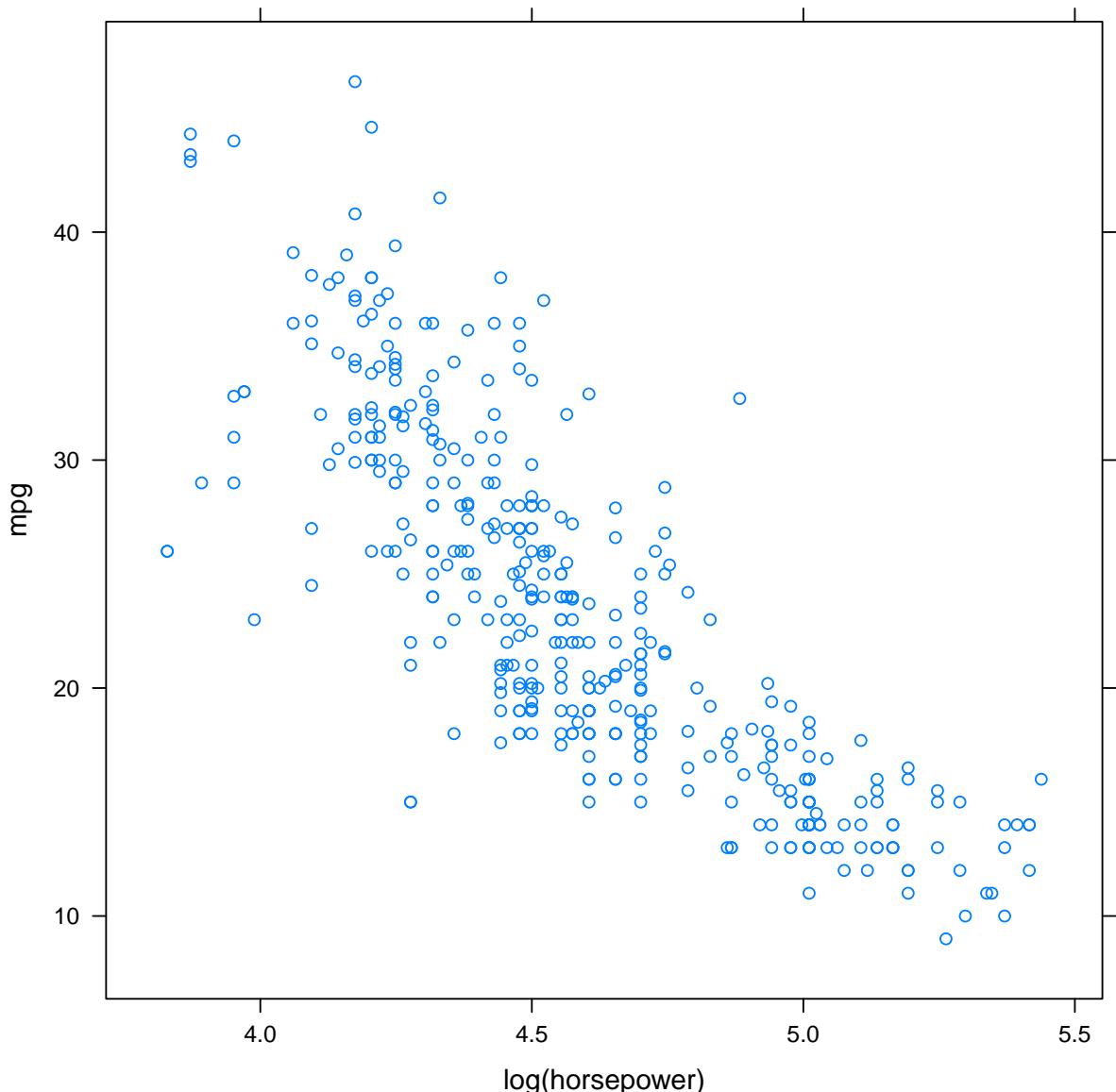
```

Auto <- read.table("Auto.data", header = TRUE,
  na.strings = "?")
Auto <- na.omit(Auto)
xyplot(mpg ~ horsepower, data = Auto)

```



```
xyplot(mpg ~ log(horsepower), data = Auto)
```



```
11 <- lm(mpg ~ horsepower, data = Auto)
12 <- lm(mpg ~ poly(horsepower, degree = 2),
      data = Auto)
13 <- lm(mpg ~ log(horsepower), data = Auto)
14 <- lm(mpg ~ poly(horsepower, degree = 5),
      data = Auto)
15 <- lm(mpg ~ poly(horsepower, degree = 6),
      data = Auto)
summary(11)

##
## Call:
## lm(formula = mpg ~ horsepower, data = Auto)
## 
## Residuals:
```

```

##      Min      1Q Median      3Q      Max
## -13.5710 -3.2592 -0.3435  2.7630 16.9240
##
## Coefficients:
##                 Estimate Std. Error t value Pr(>|t|)
## (Intercept) 39.935861   0.717499  55.66 <2e-16 ***
## horsepower -0.157845   0.006446 -24.49 <2e-16 ***
## ---
## Signif. codes:
## 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 4.906 on 390 degrees of freedom
## Multiple R-squared:  0.6059, Adjusted R-squared:  0.6049
## F-statistic: 599.7 on 1 and 390 DF,  p-value: < 2.2e-16

summary(llog)

##
## Call:
## lm(formula = mpg ~ log(horsepower), data = Auto)
##
## Residuals:
##      Min      1Q Median      3Q      Max
## -14.2299 -2.7818 -0.2322  2.6661 15.4695
##
## Coefficients:
##                 Estimate Std. Error t value Pr(>|t|)
## (Intercept) 108.6997    3.0496  35.64 <2e-16 ***
## log(horsepower) -18.5822    0.6629 -28.03 <2e-16 ***
## ---
## Signif. codes:
## 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 4.501 on 390 degrees of freedom
## Multiple R-squared:  0.6683, Adjusted R-squared:  0.6675
## F-statistic: 785.9 on 1 and 390 DF,  p-value: < 2.2e-16

summary(15)

##
## Call:
## lm(formula = mpg ~ poly(horsepower, degree = 5), data = Auto)
##
## Residuals:
##      Min      1Q Median      3Q      Max
## -15.4326 -2.5285 -0.2925  2.1750 15.9730
##
## Coefficients:
##                               Estimate Std. Error t value

```

```

## (Intercept)           23.4459   0.2185 107.308
## poly(horsepower, degree = 5)1 -120.1377  4.3259 -27.772
## poly(horsepower, degree = 5)2   44.0895  4.3259 10.192
## poly(horsepower, degree = 5)3  -3.9488  4.3259 -0.913
## poly(horsepower, degree = 5)4  -5.1878  4.3259 -1.199
## poly(horsepower, degree = 5)5   13.2722  4.3259  3.068
##                                     Pr(>|t|)
## (Intercept)           < 2e-16 ***
## poly(horsepower, degree = 5)1 < 2e-16 ***
## poly(horsepower, degree = 5)2 < 2e-16 ***
## poly(horsepower, degree = 5)3  0.36190
## poly(horsepower, degree = 5)4  0.23117
## poly(horsepower, degree = 5)5  0.00231 **
## ---
## Signif. codes:
## 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 4.326 on 386 degrees of freedom
## Multiple R-squared:  0.6967, Adjusted R-squared:  0.6928
## F-statistic: 177.4 on 5 and 386 DF,  p-value: < 2.2e-16

summary(16)

##
## Call:
## lm(formula = mpg ~ poly(horsepower, degree = 6), data = Auto)
##
## Residuals:
##    Min     1Q Median     3Q    Max
## -15.595 -2.571 -0.269  2.209 15.362
##
## Coefficients:
##                               Estimate Std. Error t value
## (Intercept)           23.4459   0.2177 107.715
## poly(horsepower, degree = 6)1 -120.1377  4.3096 -27.877
## poly(horsepower, degree = 6)2   44.0895  4.3096 10.231
## poly(horsepower, degree = 6)3  -3.9488  4.3096 -0.916
## poly(horsepower, degree = 6)4  -5.1878  4.3096 -1.204
## poly(horsepower, degree = 6)5   13.2722  4.3096  3.080
## poly(horsepower, degree = 6)6  -8.5462  4.3096 -1.983
##                               Pr(>|t|)
## (Intercept)           < 2e-16 ***
## poly(horsepower, degree = 6)1 < 2e-16 ***
## poly(horsepower, degree = 6)2 < 2e-16 ***
## poly(horsepower, degree = 6)3  0.36008
## poly(horsepower, degree = 6)4  0.22941
## poly(horsepower, degree = 6)5  0.00222 **
## poly(horsepower, degree = 6)6  0.04807 *

```

```

## ---
## Signif. codes:
## 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 4.31 on 385 degrees of freedom
## Multiple R-squared:  0.6998, Adjusted R-squared:  0.6951
## F-statistic: 149.6 on 6 and 385 DF,  p-value: < 2.2e-16

AIC(l1, l2, llog, l5, l6)

##      df      AIC
## l1     3 2363.324
## l2     4 2274.354
## llog   3 2295.760
## l5     7 2268.663
## l6     8 2266.680

BIC(l1, l2, llog, l5, l6)

##      df      BIC
## l1     3 2375.237
## l2     4 2290.239
## llog   3 2307.674
## l5     7 2296.462
## l6     8 2298.450

anova(l1, l2)

## Analysis of Variance Table
##
## Model 1: mpg ~ horsepower
## Model 2: mpg ~ poly(horsepower, degree = 2)
##   Res.Df   RSS Df Sum of Sq    F    Pr(>F)
## 1     390 9385.9
## 2     389 7442.0  1    1943.9 101.61 < 2.2e-16 ***
## ---
## Signif. codes:
## 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

anova(l1, l5)

## Analysis of Variance Table
##
## Model 1: mpg ~ horsepower
## Model 2: mpg ~ poly(horsepower, degree = 5)
##   Res.Df   RSS Df Sum of Sq    F    Pr(>F)
## 1     390 9385.9
## 2     386 7223.4  4    2162.5 28.89 < 2.2e-16 ***
## ---
## Signif. codes:
## 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

```

```

anova(l1, l6)

## Analysis of Variance Table
##
## Model 1: mpg ~ horsepower
## Model 2: mpg ~ poly(horsepower, degree = 6)
##   Res.Df   RSS Df Sum of Sq    F    Pr(>F)
## 1     390 9385.9
## 2     385 7150.3  5    2235.6 24.074 < 2.2e-16 ***
## ---
## Signif. codes:
## 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

anova(l5, l6)

## Analysis of Variance Table
##
## Model 1: mpg ~ poly(horsepower, degree = 5)
## Model 2: mpg ~ poly(horsepower, degree = 6)
##   Res.Df   RSS Df Sum of Sq    F    Pr(>F)
## 1     386 7223.4
## 2     385 7150.3  1    73.038 3.9326 0.04807 *
## ---
## Signif. codes:
## 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

```

4.3 Линейная регрессия (Университеты)

```

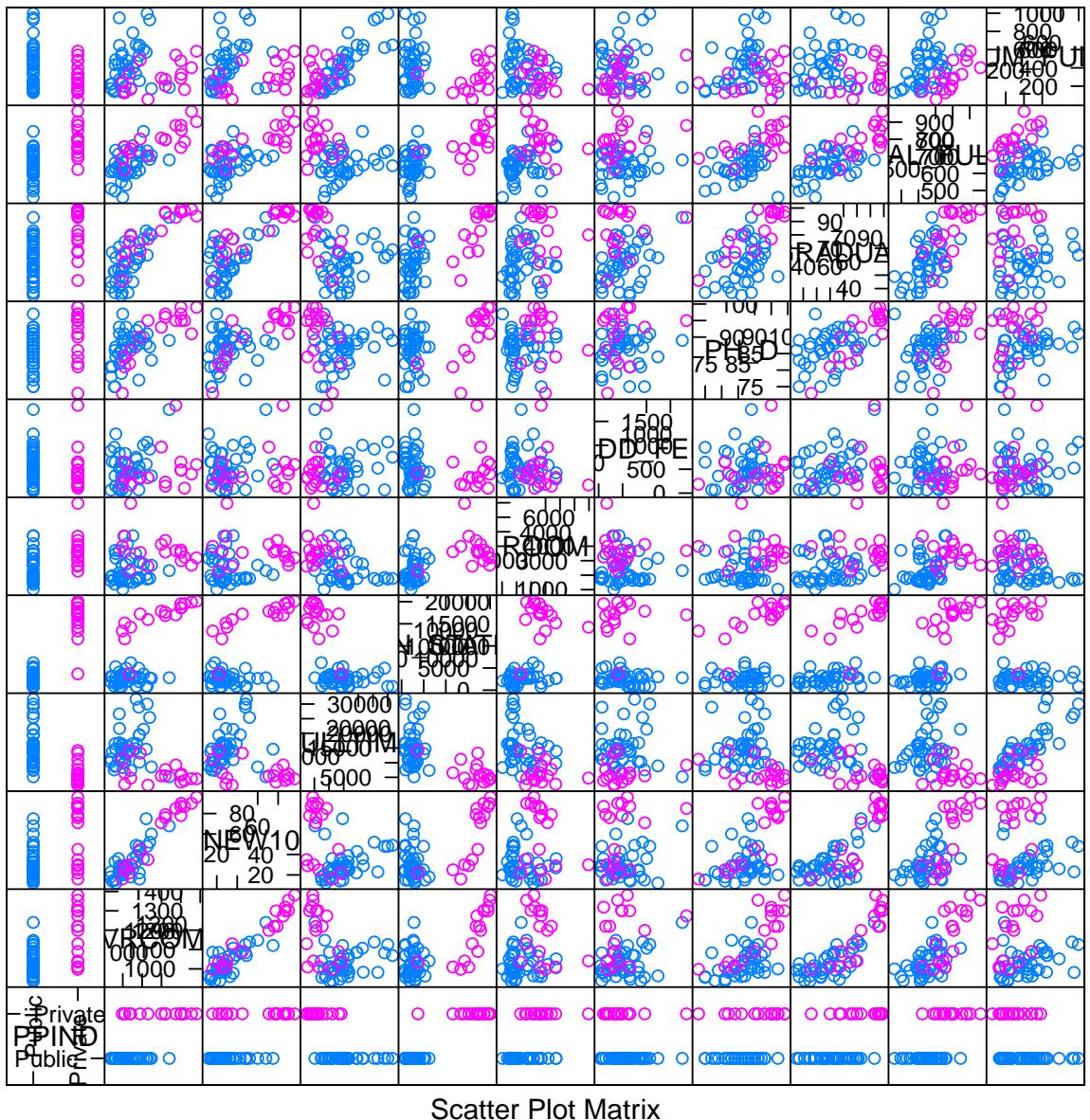
library(MASS)
library(lattice)
library(latticeExtra)
library(latticist)
df <- read.csv2(file = "I.csv")
# Я установил пакет latticist, чтобы
# доволь налюбоваться на университеты Я
# решил, что переменных слишком много и
# оставил только по одной из каждого
# класса Например, количество вечерников
# и очников --- явно характеристики
# одного и того же, поэтому оставим
# только одну из них Мы же помним, что
# брать сильно коррелированные признаки в
# модель --- дурной тон?) Итакаааак,
# барабанная дробь, я решил оставить:
# PPIND - фактор, 1 - Государственный, 2
# - Частный университет. AVRCOMB -

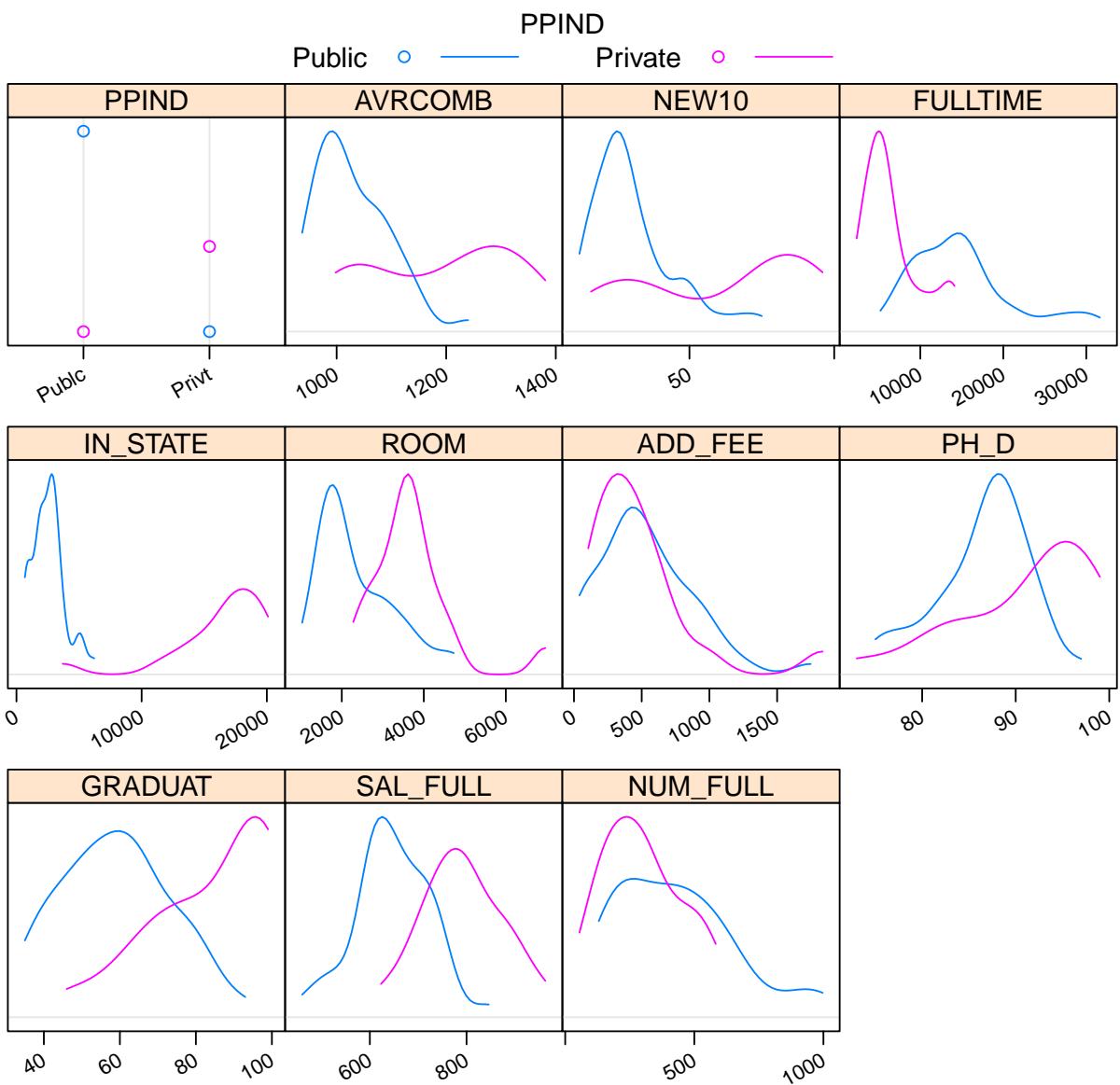
```

```

# средний средний балл на вступительных
# экзаменах (SAT, вроде нашего ЕГЭ).
# NEW10 - Это то, что будем
# аппроксимировать, задание у нас такое
# Это процент свежезачисленных
# студентов-отличников (Процент среди
# поступивших, тех, кто в Н.С. входил в
# 10% лучших) В оригинале ''Pct. new
# students from top 10% of H.S. class''
# FULLTIME - Количество студентов-очников
# IN_STATE - Плата за обучение для
# местных ROOM - Плата за койку в общаге
# ADD_FEE - Дополнительные поборы (сверх
# платы за обучение, койку и учебные
# материалы) PH_D - Процент кандидатов
# наук среди педагогического состава.
# GRADUAT - Процент выпускавшихся.
# Гы-гы=) SAL_FULL - Средняя зарплата
# полного профессора (full professor).
# NUM_FULL - Количество этих самых полных
# профессоров Теперь нам надо обрезать и
# подправить исходный датафрейм и
# скормить его latticist.
# latticist(df)
# Отобрали признаки
df <- subset(df, select = c(PPIND, AVRCOMB,
    NEW10, FULLTIME, IN_STATE, ROOM, ADD_FEE,
    PH_D, GRADUAT, SAL_FULL, NUM_FULL))
# Сконвертировали тип Университета в
# фактор, так и вывод красивее, и в
# модели будет удобнее интерпретировать
df$PPIND <- factor(df$PPIND, labels = c("Public",
    "Private"))
df <- na.exclude(df)
splom(df, groups = df$PPIND)

```

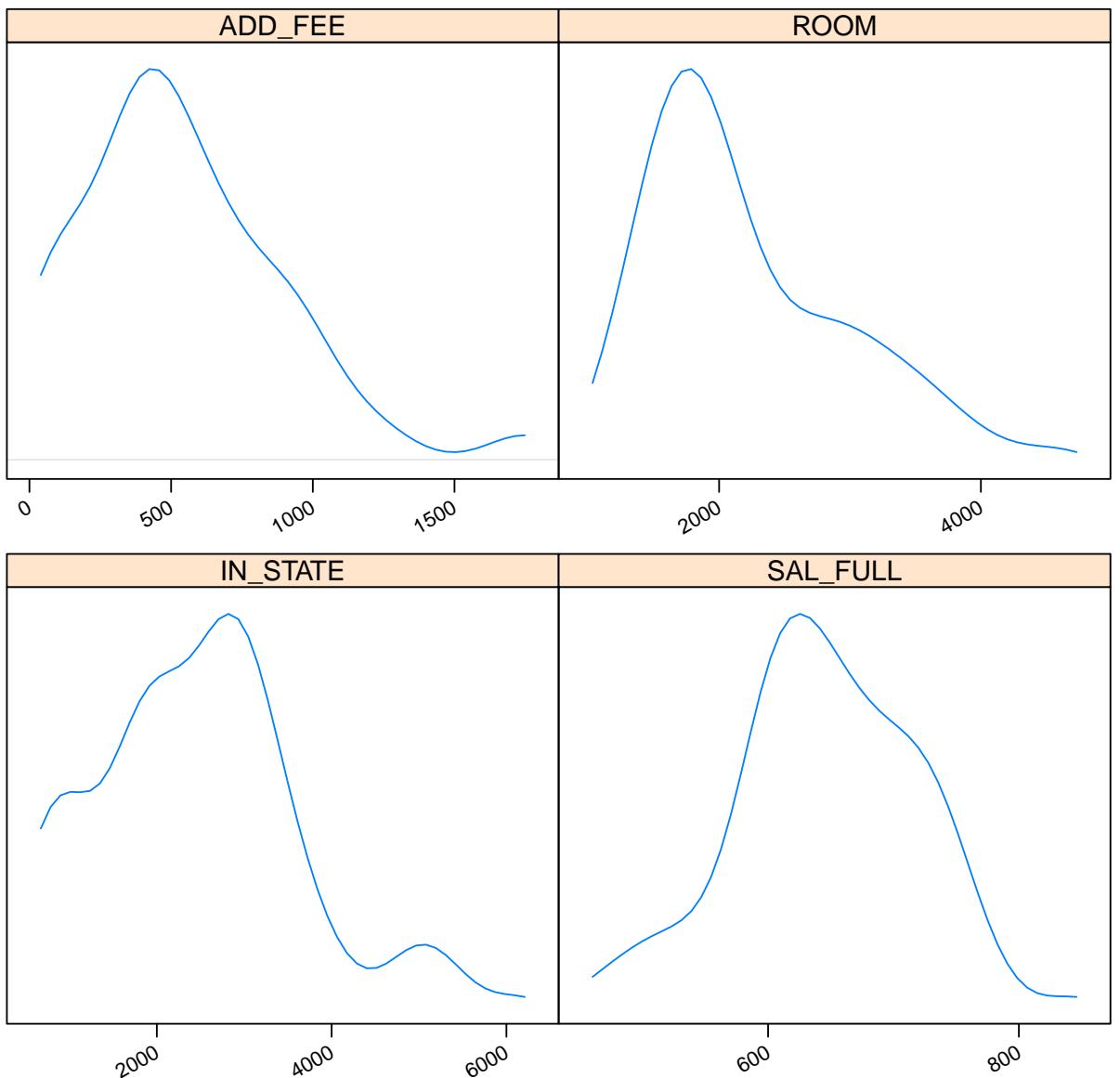




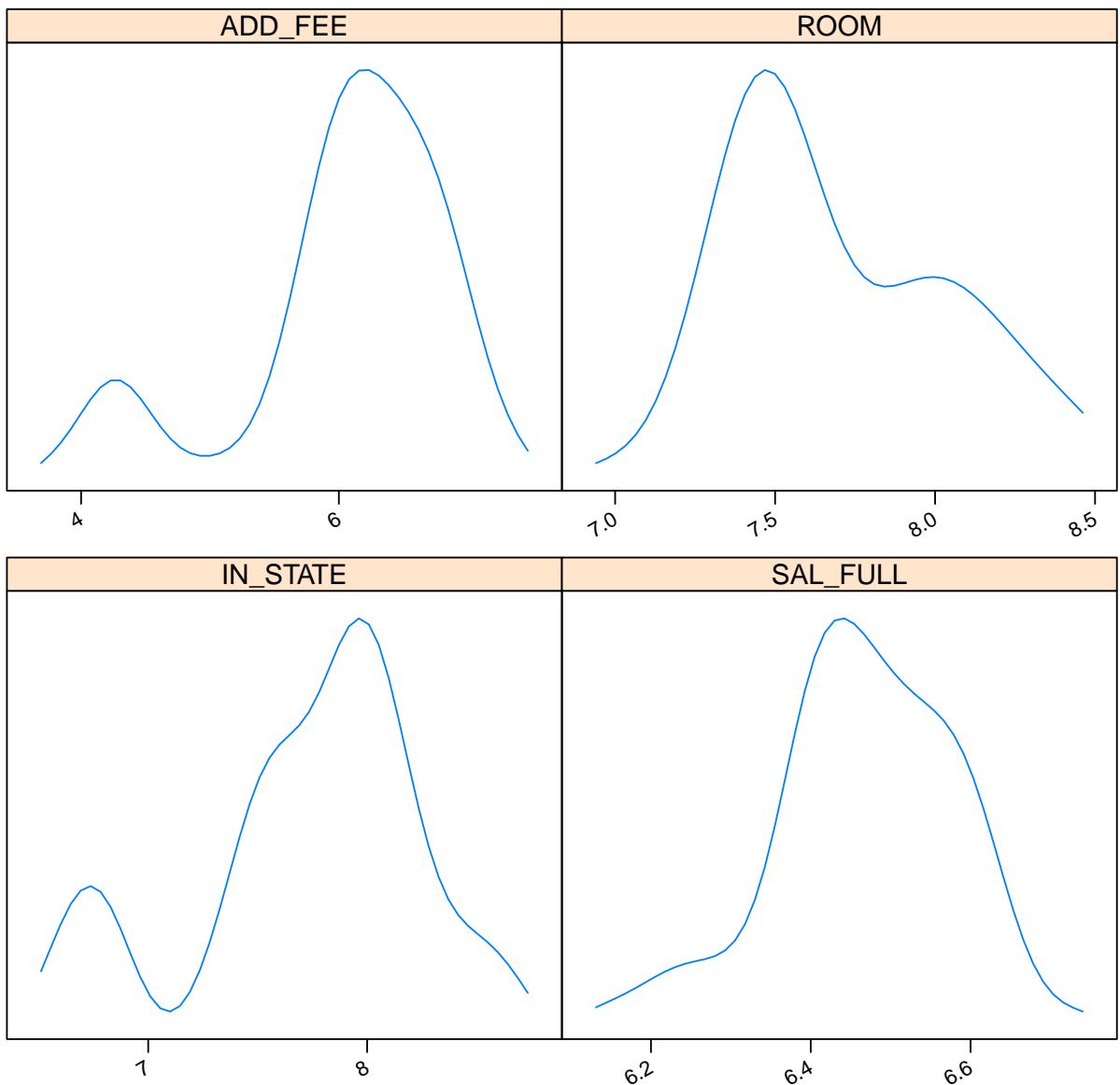
```

df.pub <- subset(df, PPIND == "Public")
# Теперь посмотрим на нормальность,
# может, что-то стоит
# прологарифмировать?...
# latticist(df.pub) Подозрение падает на
# денежные признаки. Они часто
# логнормальны.
marginal.plot(subset(df.pub, select = c(ADD_FEE,
    ROOM, IN_STATE, SAL_FULL)))

```



```
marginal.plot(log(subset(df.pub, select = c(ADD_FEE,  
ROOM, IN_STATE, SAL_FULL))))
```



```

# После логарифмирования появилась
# мультидисперсионность, хотя распределения
# стали на вид немного более
# симметричными. Я все-таки хочу
# оставить логарифмирование, потому что
# это денежные признаки Но потом мы
# проверим и без него
# Итого
fit1 <- lm(NEW10 ~ AVRCOMB + FULLTIME + log(IN_STATE) +
  log(ROOM) + log(ADD_FEE) + log(SAL_FULL) +
  PH_D + GRADUAT + NUM_FULL, data = df.pub)
summary(fit1)

##
## Call:

```

```

## lm(formula = NEW10 ~ AVRCOMB + FULLTIME + log(IN_STATE) + log(ROOM) +
##      log(ADD_FEE) + log(SAL_FULL) + PH_D + GRADUAT + NUM_FULL,
##      data = df.pub)
##
## Residuals:
##    Min     1Q Median     3Q    Max
## -22.770 -3.407  1.111  3.588 15.071
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept) -4.397e+01 8.490e+01 -0.518 0.60828
## AVRCOMB      1.469e-01 2.823e-02  5.203 1.31e-05 ***
## FULLTIME     -2.458e-04 3.797e-04 -0.647 0.52227
## log(IN_STATE) -9.057e+00 2.646e+00 -3.422 0.00181 **
## log(ROOM)     -4.011e+00 4.082e+00 -0.982 0.33375
## log(ADD_FEE)   -3.651e+00 1.628e+00 -2.242 0.03251 *
## log(SAL_FULL)  1.114e+01 1.459e+01  0.764 0.45090
## PH_D          -4.984e-01 2.842e-01 -1.754 0.08965 .
## GRADUAT        2.645e-01 1.469e-01  1.801 0.08173 .
## NUM_FULL       1.225e-02 1.172e-02  1.045 0.30431
## ---
## Signif. codes:
## 0 *** 0.001 ** 0.01 * 0.05 . 0.1 ' ' 1
##
## Residual standard error: 7.755 on 30 degrees of freedom
## Multiple R-squared:  0.7874, Adjusted R-squared:  0.7236
## F-statistic: 12.35 on 9 and 30 DF, p-value: 6.861e-08

# Имеем --- отличники прекрасно сдают
# экзамены и поступают туда, где меньше
# надо платить, меньше поборов, меньше
# кандидатов (sic!) и больше процент
# успешно закончивших. На самом деле,
# это несодержательно. AVRCOMB ---
# абсолютно не нужен нам. И так понятно,
# что отличники там, где отличники. Если
# мы хотим получить действительно
# информативную модель и нетривиальные
# выводы, то из предикторов AVRCOMB имеет
# смысл убрать, иначе трактовка регрессии
# будет паэтомологией

fit2 <- lm(NEW10 ~ FULLTIME + log(IN_STATE) +
  log(ROOM) + log(ADD_FEE) + log(SAL_FULL) +
  PH_D + GRADUAT + NUM_FULL, data = df.pub)
summary(fit2)

##
## Call:

```

```

## lm(formula = NEW10 ~ FULLTIME + log(IN_STATE) + log(ROOM) + log(ADD_FEE) +
##      log(SAL_FULL) + PH_D + GRADUAT + NUM_FULL, data = df.pub)
##
## Residuals:
##    Min      1Q  Median      3Q     Max
## -22.749  -5.841   0.533   5.365  23.062
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept) 4.531e+01 1.128e+02  0.402 0.690693
## FULLTIME   -3.024e-04 5.150e-04 -0.587 0.561361
## log(IN_STATE) -1.313e+01 3.430e+00 -3.829 0.000586 ***
## log(ROOM)   -2.312e+00 5.521e+00 -0.419 0.678299
## log(ADD_FEE) -4.263e+00 2.204e+00 -1.934 0.062256 .
## log(SAL_FULL) 1.671e+01 1.974e+01  0.846 0.403798
## PH_D       -2.902e-01 3.818e-01 -0.760 0.452957
## GRADUAT    7.369e-01 1.566e-01  4.705  5e-05 ***
## NUM_FULL   1.917e-02 1.580e-02  1.213 0.234185
## ---
## Signif. codes:
## 0 *** 0.001 ** 0.01 * 0.05 . 0.1 ' ' 1
##
## Residual standard error: 10.52 on 31 degrees of freedom
## Multiple R-squared:  0.5956, Adjusted R-squared:  0.4912
## F-statistic: 5.706 on 8 and 31 DF, p-value: 0.0001718

# Уже на что-то похоже. Поступают туда,
# где дешевле, где больше шанс
# выпуститься и меньше поборов.
# Попробуем уменьшить число признаков.
# Вручную по t-test и по Акаике
fit2.manual <- lm(NEW10 ~ log(IN_STATE) +
  log(ADD_FEE) + GRADUAT, data = df.pub)
summary(fit2.manual)

##
## Call:
## lm(formula = NEW10 ~ log(IN_STATE) + log(ADD_FEE) + GRADUAT,
##      data = df.pub)
##
## Residuals:
##    Min      1Q  Median      3Q     Max
## -26.2271  -5.1813  -0.4403   6.7982  19.5760
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept) 113.0619    28.2689   4.000 0.000302 ***
## log(IN_STATE) -13.1875    3.2280  -4.085 0.000235 ***

```

```

## log(ADD_FEE) -4.8304    2.1489  -2.248 0.030801 *
## GRADUAT      0.8187    0.1376   5.949 8.14e-07 ***
## ---
## Signif. codes:
## 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 10.53 on 36 degrees of freedom
## Multiple R-squared:  0.53, Adjusted R-squared:  0.4908
## F-statistic: 13.53 on 3 and 36 DF,  p-value: 4.559e-06

fit2.aic <- stepAIC(fit2)

## Start: AIC=196.08
## NEW10 ~ FULLTIME + log(IN_STATE) + log(ROOM) + log(ADD_FEE) +
##       log(SAL_FULL) + PH_D + GRADUAT + NUM_FULL
##
##          Df Sum of Sq   RSS   AIC
## - log(ROOM)     1    19.41 3451.4 194.31
## - FULLTIME     1    38.17 3470.2 194.52
## - PH_D         1    63.96 3496.0 194.82
## - log(SAL_FULL) 1    79.32 3511.3 195.00
## - NUM_FULL     1   162.97 3595.0 195.94
## <none>           3432.0 196.08
## - log(ADD_FEE)  1   414.20 3846.2 198.64
## - log(IN_STATE) 1   1623.10 5055.1 209.57
## - GRADUAT       1   2450.49 5882.5 215.63
##
## Step: AIC=194.31
## NEW10 ~ FULLTIME + log(IN_STATE) + log(ADD_FEE) + log(SAL_FULL) +
##       PH_D + GRADUAT + NUM_FULL
##
##          Df Sum of Sq   RSS   AIC
## - FULLTIME     1    36.92 3488.4 192.73
## - log(SAL_FULL) 1    60.03 3511.5 193.00
## - PH_D         1    61.34 3512.8 193.01
## <none>           3451.4 194.31
## - NUM_FULL     1   193.54 3645.0 194.49
## - log(ADD_FEE)  1   451.12 3902.6 197.22
## - log(IN_STATE) 1   1729.29 5180.7 208.55
## - GRADUAT       1   2629.10 6080.5 214.96
##
## Step: AIC=192.73
## NEW10 ~ log(IN_STATE) + log(ADD_FEE) + log(SAL_FULL) + PH_D +
##       GRADUAT + NUM_FULL
##
##          Df Sum of Sq   RSS   AIC
## - PH_D         1    42.66 3531.0 191.22
## - log(SAL_FULL) 1    65.23 3553.6 191.47

```

```

## <none>                      3488.4 192.73
## - NUM_FULL                  1     250.29 3738.6 193.50
## - log(ADD_FEE)               1     433.03 3921.4 195.41
## - log(IN_STATE)              1    1702.06 5190.4 206.63
## - GRADUAT                   1    2592.19 6080.5 212.96
##
## Step: AIC=191.22
## NEW10 ~ log(IN_STATE) + log(ADD_FEE) + log(SAL_FULL) + GRADUAT +
##       NUM_FULL
##
##                               Df Sum of Sq   RSS   AIC
## - log(SAL_FULL)            1      59.98 3591.0 189.89
## <none>                      3531.0 191.22
## - NUM_FULL                 1     212.33 3743.3 191.55
## - log(ADD_FEE)              1     458.20 3989.2 194.10
## - log(IN_STATE)             1    1724.25 5255.3 205.12
## - GRADUAT                  1    2550.89 6081.9 210.97
##
## Step: AIC=189.89
## NEW10 ~ log(IN_STATE) + log(ADD_FEE) + GRADUAT + NUM_FULL
##
##                               Df Sum of Sq   RSS   AIC
## <none>                      3591.0 189.89
## - NUM_FULL                  1     397.4 3988.4 192.09
## - log(ADD_FEE)               1     446.4 4037.4 192.58
## - log(IN_STATE)              1     1664.3 5255.3 203.12
## - GRADUAT                   1     3291.4 6882.4 213.91

summary(fit2.aic)

##
## Call:
## lm(formula = NEW10 ~ log(IN_STATE) + log(ADD_FEE) + GRADUAT +
##      NUM_FULL, data = df.pub)
##
## Residuals:
##    Min      1Q   Median      3Q      Max
## -22.9871 -5.6649 -0.5082  5.3076 23.9919
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)    
## (Intercept) 102.452608  27.733017  3.694 0.000748 ***
## log(IN_STATE) -12.574064   3.121998 -4.028 0.000288 ***
## log(ADD_FEE)  -4.344059   2.082633 -2.086 0.044348 *  
## GRADUAT        0.765564   0.135166  5.664 2.14e-06 ***
## NUM_FULL       0.014231   0.007231  1.968 0.057016 .  
## ---
## Signif. codes:
```

```

## 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 '' 1
##
## Residual standard error: 10.13 on 35 degrees of freedom
## Multiple R-squared:  0.5768, Adjusted R-squared:  0.5285
## F-statistic: 11.93 on 4 and 35 DF,  p-value: 3.23e-06

# Акаике предлагает оставить признак
# NUM_FULL. Отличники поступают туда
# еще, где народу побольше.
# Посравниваю-ка я модели...
AIC(fit2.manual, fit2.aic)

##           df      AIC
## fit2.manual 5 307.6059
## fit2.aic     6 305.4073

# Неудивительно, потому что fit.aic
# построена по stepAIC()
anova(fit2.manual, fit2.aic)

## Analysis of Variance Table
##
## Model 1: NEW10 ~ log(IN_STATE) + log(ADD_FEE) + GRADUAT
## Model 2: NEW10 ~ log(IN_STATE) + log(ADD_FEE) + GRADUAT + NUM_FULL
##   Res.Df   RSS Df Sum of Sq    F  Pr(>F)
## 1     36 3988.4
## 2     35 3591.0  1    397.42 3.8735 0.05702 .
## ---
## Signif. codes:
## 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 '' 1

# Посмотрим на корреляции признаков,
# возможно, некоторые признаки захочется
# удалить, потому что они сильно
# коррелируют с другими
cor(fit2.aic$model)

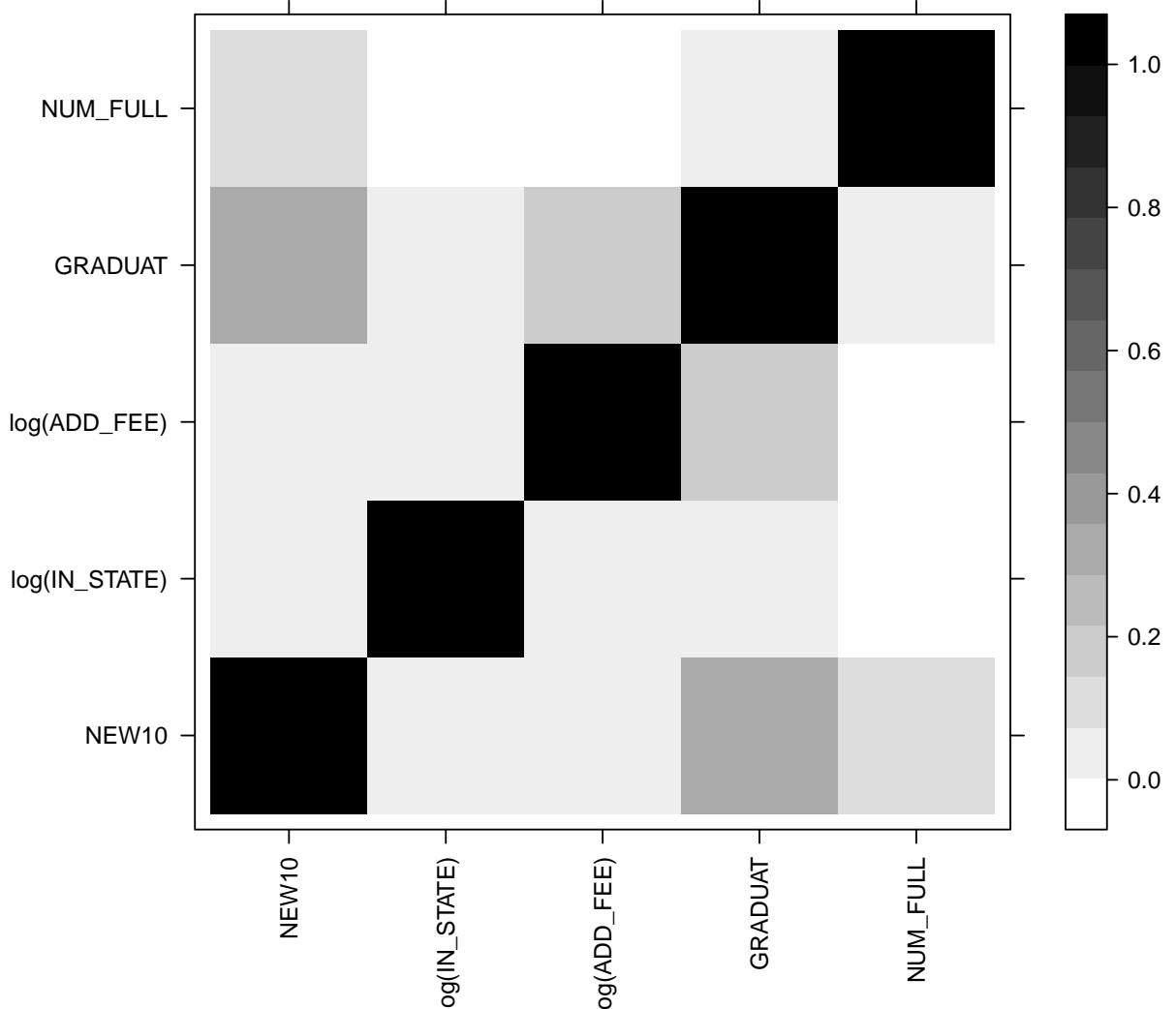
##          NEW10 log(IN_STATE) log(ADD_FEE)
## NEW10        1.0000000 -0.25000604  0.11170368
## log(IN_STATE) -0.2500060  1.00000000 -0.15425172
## log(ADD_FEE)   0.1117037 -0.15425172  1.00000000
## GRADUAT       0.5449545  0.26450468  0.41064855
## NUM_FULL      0.3516683 -0.02384413 -0.02151739
##             GRADUAT    NUM_FULL
## NEW10        0.5449545  0.35166826
## log(IN_STATE) 0.2645047 -0.02384413
## log(ADD_FEE)  0.4106486 -0.02151739
## GRADUAT      1.0000000  0.15156033
## NUM_FULL     0.1515603  1.00000000

```

```

levelplot(cor(fit2.aic$model)^2, par.settings = list(regions = list(col = colorRamp
  scales = list(x = list(rot = 90)), xlab = "",
  ylab = ""))

```



```

summary(fit2.aic)

##
## Call:
## lm(formula = NEW10 ~ log(IN_STATE) + log(ADD_FEE) + GRADUAT +
##     NUM_FULL, data = df.pub)
##
## Residuals:
##      Min       1Q   Median       3Q      Max 
## -22.9871  -5.6649  -0.5082  5.3076 23.9919 
## 
```

```

## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept) 102.452608  27.733017   3.694 0.000748 ***
## log(IN_STATE) -12.574064   3.121998  -4.028 0.000288 ***
## log(ADD_FEE)  -4.344059   2.082633  -2.086 0.044348 *
## GRADUAT        0.765564   0.135166   5.664 2.14e-06 ***
## NUM_FULL       0.014231   0.007231   1.968 0.057016 .
## ---
## Signif. codes:
## 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 10.13 on 35 degrees of freedom
## Multiple R-squared:  0.5768, Adjusted R-squared:  0.5285
## F-statistic: 11.93 on 4 and 35 DF,  p-value: 3.23e-06

# ADD_FEE и NUM_FULL у меня на большом
# подозрении, особенно первый. Они
# малозначимы и сильно коррелируют с
# GRADUAT, велика вероятность, что они
# мусорные Используем CV сравнение
# train-test слишком неустойчиво себя
# ведет

l <- update(fit2.aic, . ~ . - log(ADD_FEE) -
             NUM_FULL)
library(e1071)
tune(lm, fit2.aic$call$formula, data = df.pub,
      tunecontrol = tune.control(sampling = "cross",
                                  cross = 35))

##
## Error estimation of 'lm' using 35-fold cross validation: 135.5141

tune(lm, l$call$formula, data = df.pub, tunecontrol = tune.control(sampling = "cross",
                                                                     cross = 35))

##
## Error estimation of 'lm' using 35-fold cross validation: 130.0471

# Вывод --- все-таки выкидывать не стоило
# Попробуем нелогарифмировать признаки и
# сравним модели
fit.nolog <- lm(NEW10 ~ IN_STATE + ADD_FEE +
                 GRADUAT + NUM_FULL, data = df.pub)
summary(fit.nolog)

##
## Call:
## lm(formula = NEW10 ~ IN_STATE + ADD_FEE + GRADUAT + NUM_FULL,
##      data = df.pub)

```

```

## 
## Residuals:
##      Min       1Q   Median      3Q     Max
## -24.0914  -4.6808   0.5468  3.5763 24.7934
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept) -1.866977  7.509751 -0.249 0.805118
## IN_STATE    -0.005280  0.001417 -3.726 0.000685 ***
## ADD_FEE     -0.005985  0.005486 -1.091 0.282694
## GRADUAT     0.726085  0.144096  5.039 1.43e-05 ***
## NUM_FULL    0.015331  0.007459  2.055 0.047376 *
## ---
## Signif. codes:
## 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 10.45 on 35 degrees of freedom
## Multiple R-squared:  0.5498, Adjusted R-squared:  0.4984
## F-statistic: 10.69 on 4 and 35 DF, p-value: 9.126e-06

AIC(fit.nolog, fit2.aic)

##          df      AIC
## fit.nolog 6 307.8804
## fit2.aic   6 305.4073

# Как видно, прологарифмировали мы
# все-таки не зря

```

4.3.1 Advertising, окончательный результат

```

library(lattice)
library(latticeExtra)
library(MASS)
library(e1071)
Advertising <- read.csv("Advertising.csv")
Advertising$X <- NULL
l <- lm(Sales ~ TV + Radio + Newspaper, data = Advertising)
li <- lm(Sales ~ (TV + Radio + Newspaper)^2,
          data = Advertising)
ltvradio <- lm(Sales ~ TV + Radio + Newspaper +
                 TV:Radio, data = Advertising)
laic <- stepAIC(li)

## Start: AIC=-18.59
## Sales ~ (TV + Radio + Newspaper)^2
## 

```

```

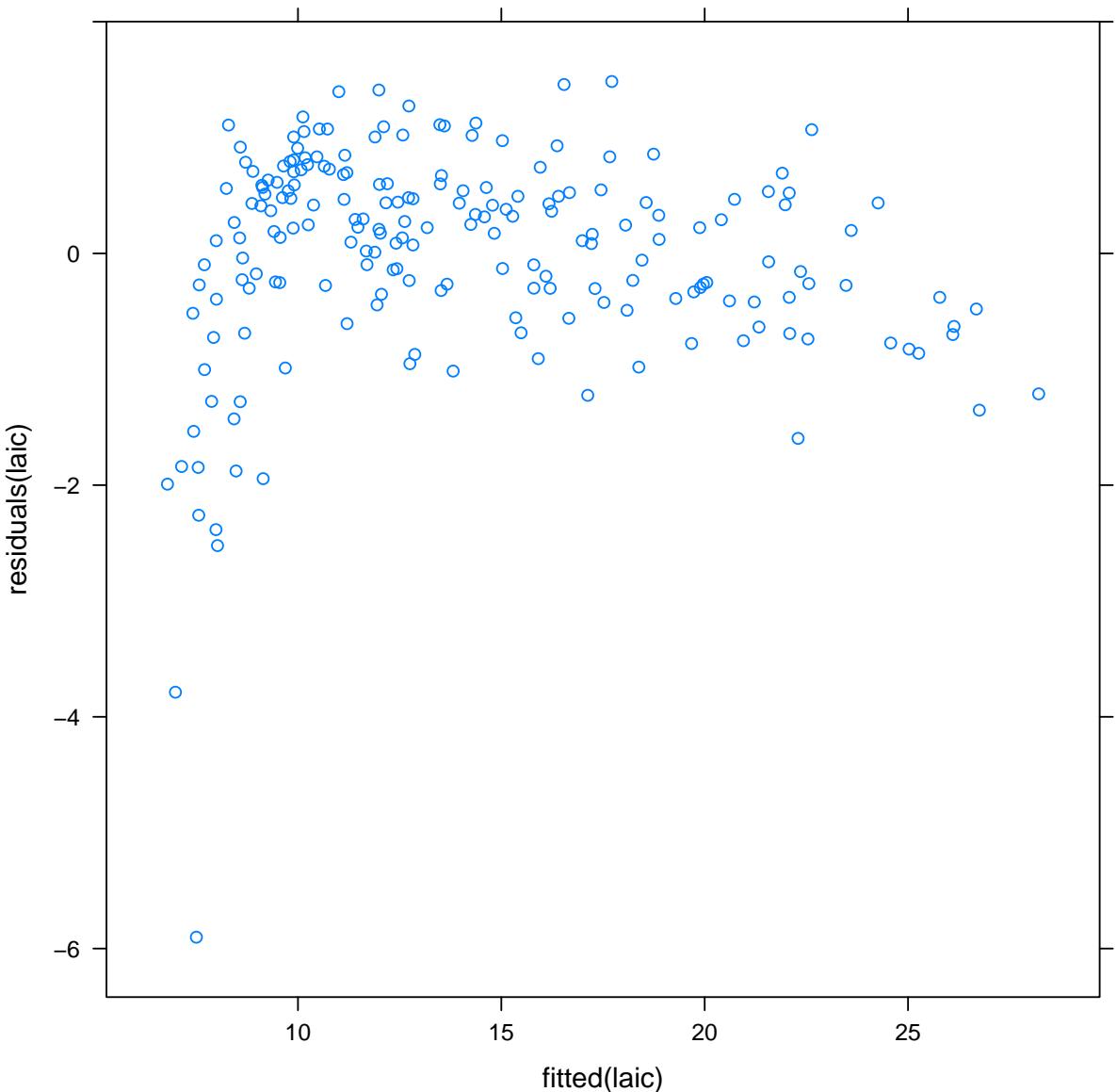
##                                     Df Sum of Sq    RSS      AIC
## - Radio:Newspaper     1      0.19 170.12 -20.363
## <none>                               169.93 -18.586
## - TV:Newspaper        1      4.37 174.30 -15.511
## - TV:Radio             1     349.71 519.64 202.965
##
## Step:  AIC=-20.36
## Sales ~ TV + Radio + Newspaper + TV:Radio + TV:Newspaper
##
##                                     Df Sum of Sq    RSS      AIC
## <none>                               170.12 -20.363
## - TV:Newspaper     1      4.19 174.31 -17.494
## - TV:Radio         1     352.83 522.95 202.234

summary(laic)

##
## Call:
## lm(formula = Sales ~ TV + Radio + Newspaper + TV:Radio + TV:Newspaper,
##      data = Advertising)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -5.9019 -0.3818  0.1937  0.5741  1.4839
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)    
## (Intercept) 6.541e+00 2.652e-01 24.668 <2e-16 ***
## TV          2.035e-02 1.605e-03 12.675 <2e-16 ***
## Radio       2.018e-02 9.734e-03  2.073 0.0395 *  
## Newspaper   1.342e-02 6.377e-03  2.105 0.0366 *  
## TV:Radio    1.136e-03 5.664e-05 20.059 <2e-16 ***
## TV:Newspaper -7.719e-05 3.531e-05 -2.187 0.0300 *  
## ---
## Signif. codes:
## 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.9364 on 194 degrees of freedom
## Multiple R-squared:  0.9686, Adjusted R-squared:  0.9678 
## F-statistic: 1197 on 5 and 194 DF,  p-value: < 2.2e-16

xyplot(residuals(laic) ~ fitted(laic))

```



```

lsq <- lm(Sales ~ poly(TV, Radio, Newspaper,
  degree = 2), data = Advertising)
summary(lsq)

##
## Call:
## lm(formula = Sales ~ poly(TV, Radio, Newspaper, degree = 2),
##     data = Advertising)
##
## Residuals:
##     Min      1Q  Median      3Q     Max 
## -4.6507 -0.2941 -0.0060  0.3830  1.4378 
## 
## Coefficients:
##                               Estimate
## 
```

```

## (Intercept) 13.94105
## poly(TV, Radio, Newspaper, degree = 2)1.0.0 53.73032
## poly(TV, Radio, Newspaper, degree = 2)2.0.0 -9.97989
## poly(TV, Radio, Newspaper, degree = 2)0.1.0 40.10484
## poly(TV, Radio, Newspaper, degree = 2)1.1.0 280.35751
## poly(TV, Radio, Newspaper, degree = 2)0.2.0 0.29727
## poly(TV, Radio, Newspaper, degree = 2)0.0.1 0.94373
## poly(TV, Radio, Newspaper, degree = 2)1.0.1 -16.93577
## poly(TV, Radio, Newspaper, degree = 2)0.1.1 5.31882
## poly(TV, Radio, Newspaper, degree = 2)0.0.2 0.10711
##
# Std. Error
## (Intercept) 0.04806
## poly(TV, Radio, Newspaper, degree = 2)1.0.0 0.62580
## poly(TV, Radio, Newspaper, degree = 2)2.0.0 0.63251
## poly(TV, Radio, Newspaper, degree = 2)0.1.0 0.67044
## poly(TV, Radio, Newspaper, degree = 2)1.1.0 9.65318
## poly(TV, Radio, Newspaper, degree = 2)0.2.0 0.64859
## poly(TV, Radio, Newspaper, degree = 2)0.0.1 0.74141
## poly(TV, Radio, Newspaper, degree = 2)1.0.1 8.84414
## poly(TV, Radio, Newspaper, degree = 2)0.1.1 10.95328
## poly(TV, Radio, Newspaper, degree = 2)0.0.2 0.65997
##
# t value
## (Intercept) 290.083
## poly(TV, Radio, Newspaper, degree = 2)1.0.0 85.858
## poly(TV, Radio, Newspaper, degree = 2)2.0.0 -15.778
## poly(TV, Radio, Newspaper, degree = 2)0.1.0 59.819
## poly(TV, Radio, Newspaper, degree = 2)1.1.0 29.043
## poly(TV, Radio, Newspaper, degree = 2)0.2.0 0.458
## poly(TV, Radio, Newspaper, degree = 2)0.0.1 1.273
## poly(TV, Radio, Newspaper, degree = 2)1.0.1 -1.915
## poly(TV, Radio, Newspaper, degree = 2)0.1.1 0.486
## poly(TV, Radio, Newspaper, degree = 2)0.0.2 0.162
##
# Pr(>|t|)
## (Intercept) <2e-16 ***
## poly(TV, Radio, Newspaper, degree = 2)1.0.0 <2e-16 ***
## poly(TV, Radio, Newspaper, degree = 2)2.0.0 <2e-16 ***
## poly(TV, Radio, Newspaper, degree = 2)0.1.0 <2e-16 ***
## poly(TV, Radio, Newspaper, degree = 2)1.1.0 <2e-16 ***
## poly(TV, Radio, Newspaper, degree = 2)0.2.0 0.647
## poly(TV, Radio, Newspaper, degree = 2)0.0.1 0.205
## poly(TV, Radio, Newspaper, degree = 2)1.0.1 0.057 .
## poly(TV, Radio, Newspaper, degree = 2)0.1.1 0.628
## poly(TV, Radio, Newspaper, degree = 2)0.0.2 0.871
## ---
## Signif. codes:
## 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.6203 on 190 degrees of freedom

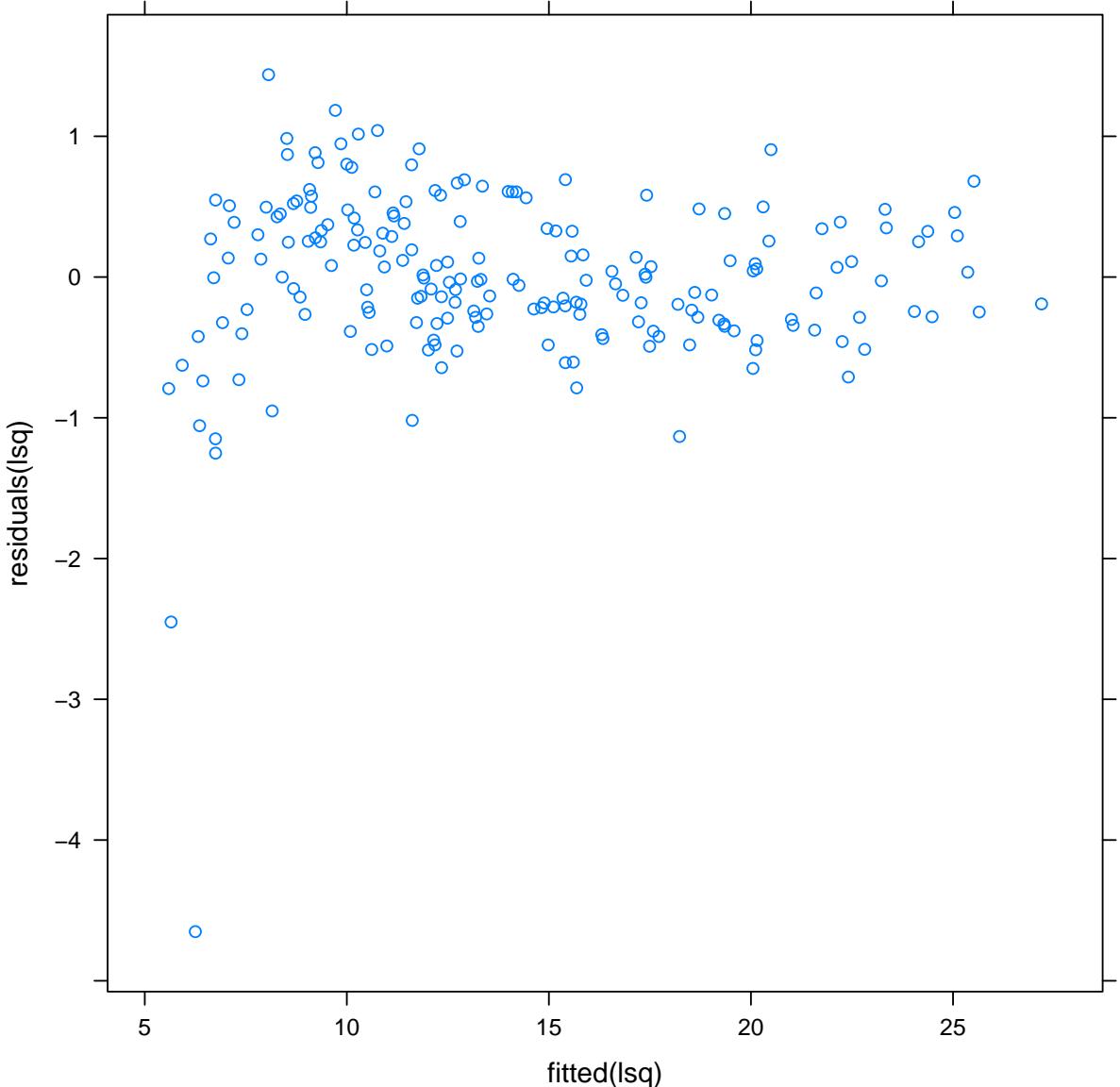
```

```

## Multiple R-squared:  0.9865, Adjusted R-squared:  0.9859
## F-statistic:  1543 on 9 and 190 DF,  p-value: < 2.2e-16

xyplot(residuals(lsq) ~ fitted(lsq))

```



```

lsqaic <- stepAIC(lsq)

## Start:  AIC=-181.3
## Sales ~ poly(TV, Radio, Newspaper, degree = 2)
##
##                                     Df Sum of Sq
## <none>
## - poly(TV, Radio, Newspaper, degree = 2)  9      5344
##                                         RSS   AIC
## <none>                           73.1 -181.3

```

```

## - poly(TV, Radio, Newspaper, degree = 2) 5417.1 661.8

summary(lsqaic)

##
## Call:
## lm(formula = Sales ~ poly(TV, Radio, Newspaper, degree = 2),
##      data = Advertising)
##
## Residuals:
##       Min     1Q Median     3Q    Max 
## -4.6507 -0.2941 -0.0060  0.3830  1.4378 
##
## Coefficients:
##                               Estimate
## (Intercept)                  13.94105
## poly(TV, Radio, Newspaper, degree = 2)1.0.0  53.73032
## poly(TV, Radio, Newspaper, degree = 2)2.0.0  -9.97989
## poly(TV, Radio, Newspaper, degree = 2)0.1.0  40.10484
## poly(TV, Radio, Newspaper, degree = 2)1.1.0  280.35751
## poly(TV, Radio, Newspaper, degree = 2)0.2.0   0.29727
## poly(TV, Radio, Newspaper, degree = 2)0.0.1   0.94373
## poly(TV, Radio, Newspaper, degree = 2)1.0.1  -16.93577
## poly(TV, Radio, Newspaper, degree = 2)0.1.1   5.31882
## poly(TV, Radio, Newspaper, degree = 2)0.0.2   0.10711
##                               Std. Error
## (Intercept)                  0.04806
## poly(TV, Radio, Newspaper, degree = 2)1.0.0   0.62580
## poly(TV, Radio, Newspaper, degree = 2)2.0.0   0.63251
## poly(TV, Radio, Newspaper, degree = 2)0.1.0   0.67044
## poly(TV, Radio, Newspaper, degree = 2)1.1.0   9.65318
## poly(TV, Radio, Newspaper, degree = 2)0.2.0   0.64859
## poly(TV, Radio, Newspaper, degree = 2)0.0.1   0.74141
## poly(TV, Radio, Newspaper, degree = 2)1.0.1   8.84414
## poly(TV, Radio, Newspaper, degree = 2)0.1.1  10.95328
## poly(TV, Radio, Newspaper, degree = 2)0.0.2   0.65997
##                               t value
## (Intercept)                290.083
## poly(TV, Radio, Newspaper, degree = 2)1.0.0   85.858
## poly(TV, Radio, Newspaper, degree = 2)2.0.0  -15.778
## poly(TV, Radio, Newspaper, degree = 2)0.1.0   59.819
## poly(TV, Radio, Newspaper, degree = 2)1.1.0   29.043
## poly(TV, Radio, Newspaper, degree = 2)0.2.0   0.458
## poly(TV, Radio, Newspaper, degree = 2)0.0.1   1.273
## poly(TV, Radio, Newspaper, degree = 2)1.0.1  -1.915
## poly(TV, Radio, Newspaper, degree = 2)0.1.1   0.486
## poly(TV, Radio, Newspaper, degree = 2)0.0.2   0.162
##                               Pr(>|t|)
```

```

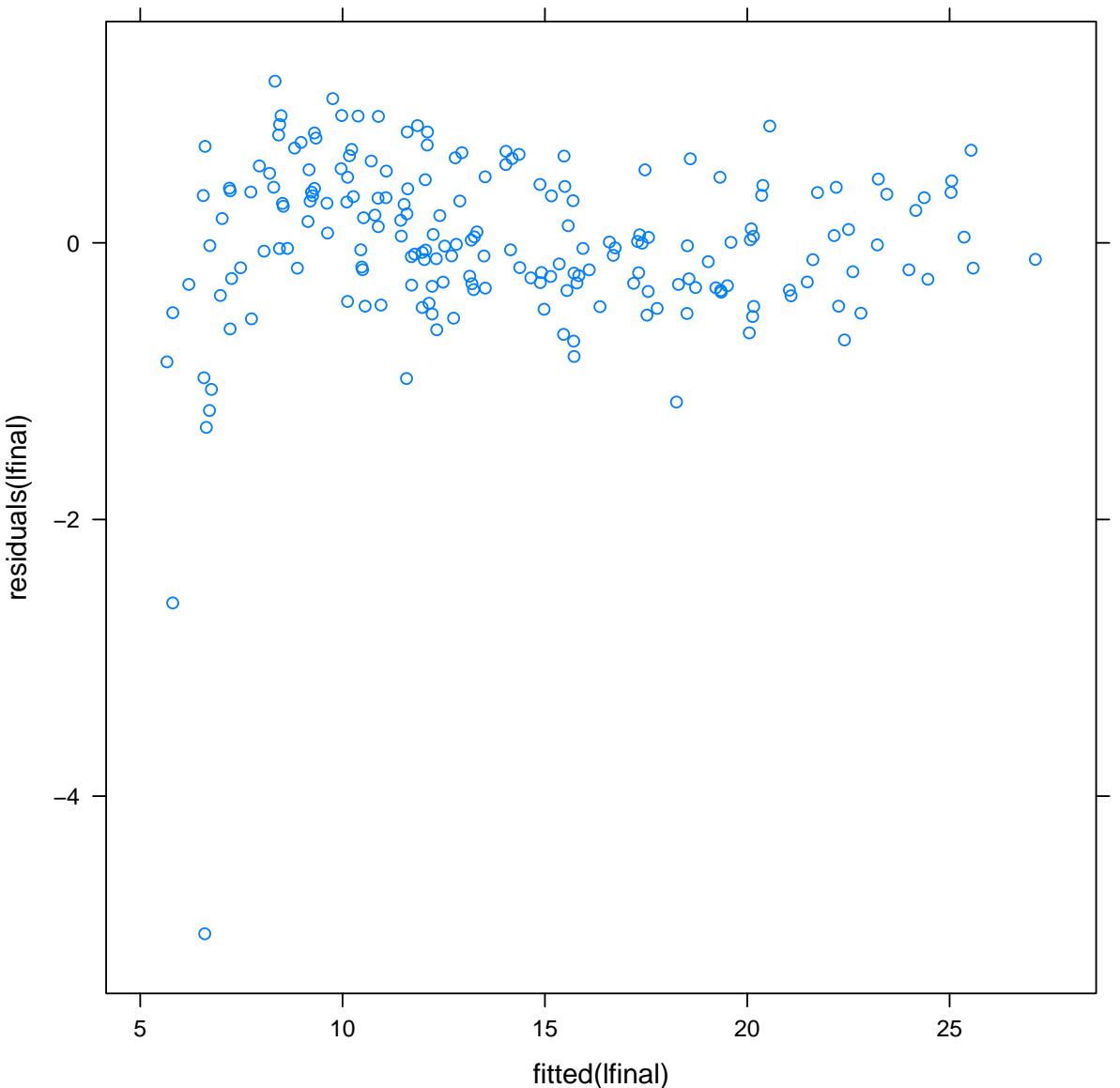
## (Intercept) <2e-16 ***
## poly(TV, Radio, Newspaper, degree = 2)1.0.0 <2e-16 ***
## poly(TV, Radio, Newspaper, degree = 2)2.0.0 <2e-16 ***
## poly(TV, Radio, Newspaper, degree = 2)0.1.0 <2e-16 ***
## poly(TV, Radio, Newspaper, degree = 2)1.1.0 <2e-16 ***
## poly(TV, Radio, Newspaper, degree = 2)0.2.0 0.647
## poly(TV, Radio, Newspaper, degree = 2)0.0.1 0.205
## poly(TV, Radio, Newspaper, degree = 2)1.0.1 0.057 .
## poly(TV, Radio, Newspaper, degree = 2)0.1.1 0.628
## poly(TV, Radio, Newspaper, degree = 2)0.0.2 0.871
## ---
## Signif. codes:
## 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.6203 on 190 degrees of freedom
## Multiple R-squared:  0.9865, Adjusted R-squared:  0.9859
## F-statistic:  1543 on 9 and 190 DF,  p-value: < 2.2e-16

lfinal <- lm(Sales ~ (TV + Radio)^2 + I(TV^2),
  data = Advertising)
summary(lfinal)

##
## Call:
## lm(formula = Sales ~ (TV + Radio)^2 + I(TV^2), data = Advertising)
##
## Residuals:
##      Min       1Q   Median       3Q      Max 
## -4.9949 -0.2969 -0.0066  0.3798  1.1686 
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)    
## (Intercept) 5.137e+00 1.927e-01 26.663 < 2e-16 ***
## TV          5.092e-02 2.232e-03 22.810 < 2e-16 ***
## Radio        3.516e-02 5.901e-03  5.959 1.17e-08 ***
## I(TV^2)     -1.097e-04 6.893e-06 -15.920 < 2e-16 ***
## TV:Radio    1.077e-03 3.466e-05 31.061 < 2e-16 ***
## ---
## Signif. codes:
## 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.6238 on 195 degrees of freedom
## Multiple R-squared:  0.986, Adjusted R-squared:  0.9857
## F-statistic:  3432 on 4 and 195 DF,  p-value: < 2.2e-16

xyplot(residuals(lfinal) ~ fitted(lfinal))

```



```
tune(lm, lfinal$call$formula, data = Advertising,
  tunecontrol = tune.control(sampling = "fix"))

##
## Error estimation of 'lm' using fixed training/validation set: 0.2684496

tune(lm, l$call$formula, data = Advertising,
  tunecontrol = tune.control(sampling = "fix"))

##
## Error estimation of 'lm' using fixed training/validation set: 1.965847

tune(lm, ltvradio$call$formula, data = Advertising,
  tunecontrol = tune.control(sampling = "fix"))
```

```

## 
## Error estimation of 'lm' using fixed training/validation set: 1.061326

tune(lm, li$call$formula, data = Advertising,
      tunecontrol = tune.control(sampling = "fix"))

## 
## Error estimation of 'lm' using fixed training/validation set: 0.7586386

```

5 Материалы с занятия 10 октября

Вспомогательный код для классификации

```

library(MASS)  # AIC(), BIC(), lda(), qda()
library(lattice) # xyplot(), densityplot()
library(latticeExtra) # layer()

## Loading required package: RColorBrewer

library(ROCR) # performance(), prediction()

## Loading required package: gplots
##
## Attaching package: 'gplots'
##
## The following object is masked from 'package:stats':
## 
##     lowess
##
## Loading required package: methods

# library(caret) # specificity(),
# sensitivity()
library(nnet) # multinom()
library(e1071) # naiveBayes(), tune()
specificity <- caret:::specificity
sensitivity <- caret:::sensitivity
ROC <- function(predicted, actual, ...) {
  pred <- prediction(predicted, as.numeric(actual))
  roc <- performance(pred, measure = "tpr",
    x.measure = "fpr", ...)
  roc
}
xyplot.performance <- function(x, ...) {
  xyplot(x@y.values[[1]] ~ x@x.values[[1]],
    xlab = x@x.name, ylab = x@y.name,
    type = "l", ...) + layer_(abline(a = 0,
    b = 1, col = "red"))
}

```

```

}

AUC <- function(predicted, actual, ...) {
  pred <- prediction(predicted, as.numeric(actual))
  perf <- performance(pred, measure = "auc",
    ...)
  perf@y.values[[1]]
}

roc.opt <- function(predicted, actual, cutoff = NULL,
  measure = c("mean", "max", "err")) {
  pred <- prediction(predicted, as.numeric(actual))
  perf <- performance(pred, measure = "fpr",
    x.measure = "fnr")
  measure <- match.arg(measure)
  fpr <- perf@y.values[[1]]
  fnr <- perf@x.values[[1]]
  npos <- pred@n.pos[[1]]
  nneg <- pred@n.neg[[1]]
  err <- (fpr * nneg + fnr * npos)/(npos +
    nneg)
  error.rate <- switch(measure, mean = (fpr +
    fnr)/2, max = pmax(fpr, fnr), err = err)
  if (is.null(cutoff)) {
    i <- which.min(error.rate)
  } else {
    i <- which.min(abs(perf@alpha.values[[1]] -
      cutoff))
  }
  list(cutoff = perf@alpha.values[[1]][i],
    fpr = fpr[i], fnr = fnr[i], err = err[i],
    error.rate = error.rate[i])
}

simple.predict.glm <- function(x, newdata,
  ...) {
  response <- predict(x, newdata, type = "response",
    ...)
  factor(levels(x$model[, 1])[1 + as.integer(response >
    0.5)])
}

my.predict.glm <- function(x, newdata = x$data,
  ..., measure = "max") {
  opt <- roc.opt(fitted(x), as.numeric(x$model[, 1]),
    measure = measure)
  cutoff <- opt$cutoff
  factor(as.integer(predict(x, newdata = newdata,
    type = "response") > cutoff), labels = levels(x$model[, 1]))
}

error.fun.max <- function(true, predicted) {

```

```

    1 - min(sensitivity(predicted, true),
             specificity(predicted, true))
}
error.fun.mean <- function(true, predicted) {
  1 - mean(sensitivity(predicted, true),
            specificity(predicted, true))
}
my.lda <- function(x, data, ...) {
  out <- lda(x, data, ...)
  out$data <- data
  out
}
my.qda <- function(x, data, ...) {
  out <- qda(x, data, ...)
  out$data <- data
  out
}
simple.predict.da <- function(...) predict(...)$class
my.predict.da <- function(x, newdata, cutoff.data = x$data,
  ..., measure = "max") {
  response <- model.frame(x$terms, cutoff.data)[,
    1]
  opt <- roc.opt(predict(x, cutoff.data)$posterior[, 2],
    as.numeric(response), measure = measure)
  cutoff <- opt$cutoff
  factor(as.integer(predict(x, newdata = newdata)$posterior[, 2] > cutoff),
    labels = levels(response))
}

```

5.1 LDA и tune

```

library(MASS)
library(lattice)
library(latticeExtra)
library(ROCR)
library(e1071)
ld <- lda(Species ~ ., data = iris)
ld

## Call:
## lda(Species ~ ., data = iris)
##
## Prior probabilities of groups:
##   setosa versicolor virginica
## 0.3333333 0.3333333 0.3333333
##
```

```

## Group means:
##           Sepal.Length Sepal.Width Petal.Length
## setosa      5.006       3.428     1.462
## versicolor  5.936       2.770     4.260
## virginica   6.588       2.974     5.552
##           Petal.Width
## setosa      0.246
## versicolor  1.326
## virginica   2.026
##
## Coefficients of linear discriminants:
##           LD1        LD2
## Sepal.Length 0.8293776 0.02410215
## Sepal.Width   1.5344731 2.16452123
## Petal.Length -2.2012117 -0.93192121
## Petal.Width  -2.8104603 2.83918785
##
## Proportion of trace:
##    LD1     LD2
## 0.9912 0.0088

train.idx <- sample(nrow(iris), size = nrow(iris) *
  0.6)
iris.train <- iris[train.idx, ]
iris.test <- iris[-train.idx, ]
ld <- lda(Species ~ ., data = iris.train,
  prior = c(1/3, 1/3, 1/3))
ld

## Call:
## lda(Species ~ ., data = iris.train, prior = c(1/3, 1/3, 1/3))
##
## Prior probabilities of groups:
##    setosa versicolor virginica
## 0.3333333 0.3333333 0.3333333
##
## Group means:
##           Sepal.Length Sepal.Width Petal.Length
## setosa      4.993548  3.429032  1.441935
## versicolor  5.951724  2.762069  4.275862
## virginica   6.396667  2.880000  5.403333
##           Petal.Width
## setosa      0.2451613
## versicolor  1.3137931
## virginica   2.0133333
##
## Coefficients of linear discriminants:
##           LD1        LD2

```

```

## Sepal.Length  1.084260  0.570387
## Sepal.Width   1.163914 -1.784660
## Petal.Length -2.579978  1.225039
## Petal.Width  -2.420225 -3.933850
##
## Proportion of trace:
##    LD1     LD2
## 0.9903 0.0097



```

```

##  

## Error estimation of 'lda' using bootstrapping: 0.0296425  

# leave-one-out  

tune(lda, Species ~ ., prior = c(1/3, 1/3,  

  1/3), data = iris, predict.func = simple.predict.da,  

  tunecontrol = tune.control(sampling = "cross",  

  cross = nrow(iris)))  

##  

## Error estimation of 'lda' using leave-one-out: 0.02  

# cross-validation (default)  

tn <- tune(lda, Species ~ ., prior = c(1/3,  

  1/3, 1/3), data = iris, predict.func = simple.predict.da,  

  tunecontrol = tune.control(sampling = "cross",  

  cross = 10))  

tn$best.model  

## Call:  

## best.tune(lda, train.x = Species ~ ., data = iris, predict.func = simple.predict  

##   tunecontrol = tune.control(sampling = "cross", cross = 10),  

##   prior = c(1/3, 1/3, 1/3))  

##  

## Prior probabilities of groups:  

##   setosa versicolor virginica  

## 0.3333333 0.3333333 0.3333333  

##  

## Group means:  

##           Sepal.Length Sepal.Width Petal.Length  

## setosa          5.006      3.428      1.462  

## versicolor       5.936      2.770      4.260  

## virginica        6.588      2.974      5.552  

##           Petal.Width  

## setosa          0.246  

## versicolor       1.326  

## virginica        2.026  

##  

## Coefficients of linear discriminants:  

##           LD1         LD2  

## Sepal.Length  0.8293776  0.02410215  

## Sepal.Width   1.5344731  2.16452123  

## Petal.Length -2.2012117 -0.93192121  

## Petal.Width   -2.8104603  2.83918785  

##  

## Proportion of trace:  

##    LD1     LD2  

## 0.9912 0.0088

```

```

tn$performance

##   dummpyparameter error dispersion
## 1          0  0.02 0.03220306

# tn$train.ind
# Naive Bayes
nb <- naiveBayes(Species ~ ., data = iris)
tn.nb <- tune(naiveBayes, Species ~ ., data = iris)
# multinomial regression
mln <- multinom(Species ~ ., data = iris,
  trace = FALSE)
tn.mln <- tune(multinom, Species ~ ., data = iris,
  trace = FALSE)
summary(mln)

## Call:
## multinom(formula = Species ~ ., data = iris, trace = FALSE)
##
## Coefficients:
##             (Intercept) Sepal.Length Sepal.Width
## versicolor    18.69037     -5.458424   -8.707401
## virginica    -23.83628      -7.923634  -15.370769
##             Petal.Length Petal.Width
## versicolor     14.24477     -3.097684
## virginica      23.65978     15.135301
##
## Std. Errors:
##             (Intercept) Sepal.Length Sepal.Width
## versicolor     34.97116      89.89215    157.0415
## virginica      35.76649      89.91153    157.1196
##             Petal.Length Petal.Width
## versicolor     60.19170      45.48852
## virginica      60.46753      45.93406
##
## Residual Deviance: 11.89973
## AIC: 31.89973

mln.aic <- stepAIC(mln)

## Start:  AIC=31.9
## Species ~ Sepal.Length + Sepal.Width + Petal.Length + Petal.Width
##
##             Df   AIC
## - Sepal.Length  2 29.267
## - Sepal.Width   2 31.498
## <none>           31.900
## - Petal.Width   2 39.773
## - Petal.Length  2 41.915

```

```

## 
## Step: AIC=29.27
## Species ~ Sepal.Width + Petal.Length + Petal.Width
## 
##          Df   AIC
## <none>      29.267
## - Sepal.Width  2 32.579
## - Petal.Length 2 39.399
## - Petal.Width  2 43.516

summary(mln.aic)

## Call:
## multinom(formula = Species ~ Sepal.Width + Petal.Length + Petal.Width,
##           data = iris, trace = FALSE)
## 
## Coefficients:
##             (Intercept) Sepal.Width Petal.Length Petal.Width
## versicolor    14.15646   -17.32240     14.09906   -2.695628
## virginica    -36.44078   -25.70717     21.98210    18.765796
## 
## Std. Errors:
##             (Intercept) Sepal.Width Petal.Length Petal.Width
## versicolor    29.66211    47.48205    68.57820    39.08345
## virginica     32.18618    48.00257    68.76678    39.75433
## 
## Residual Deviance: 13.26653
## AIC: 29.26653

```

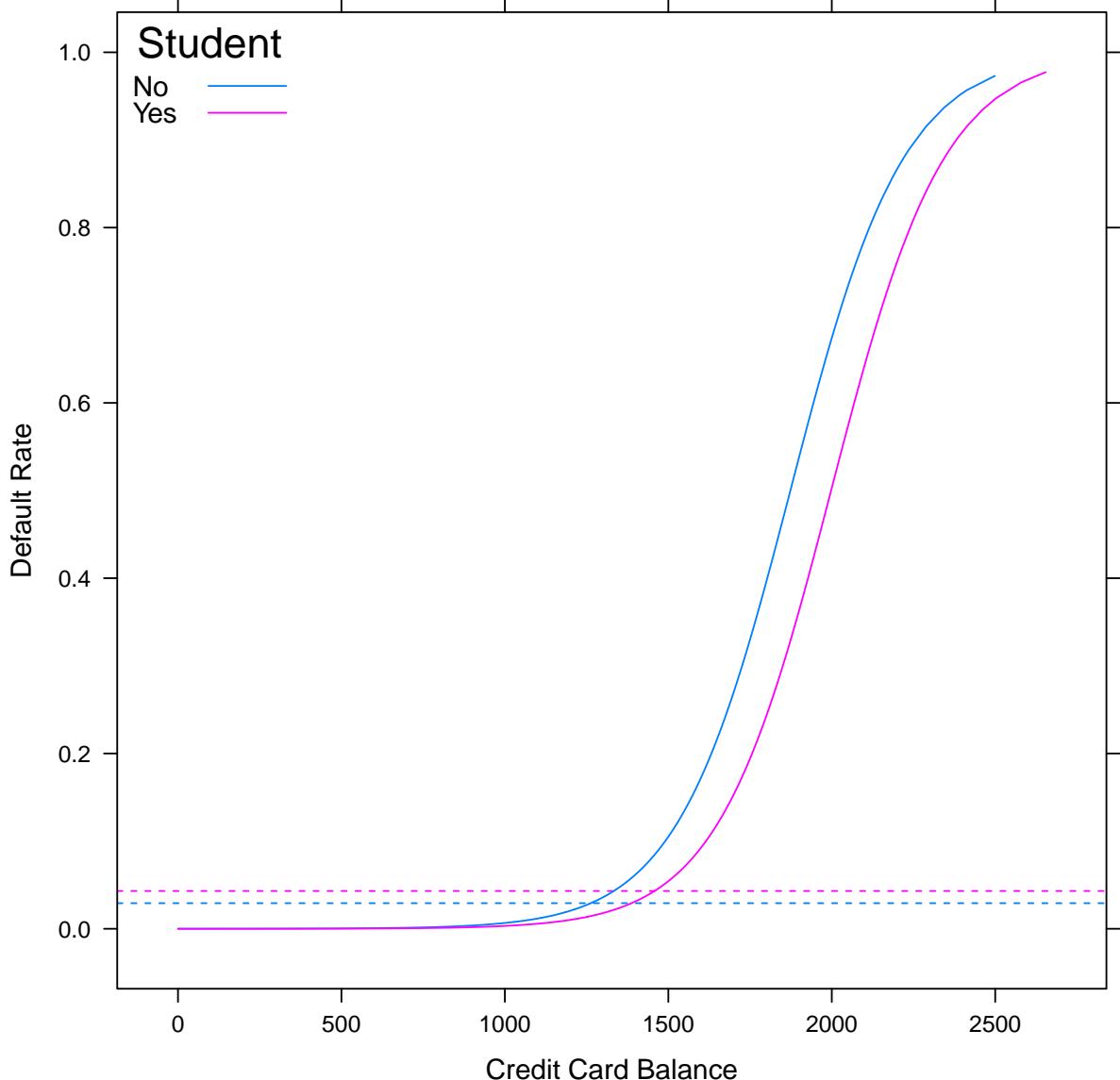
5.2 Default

```

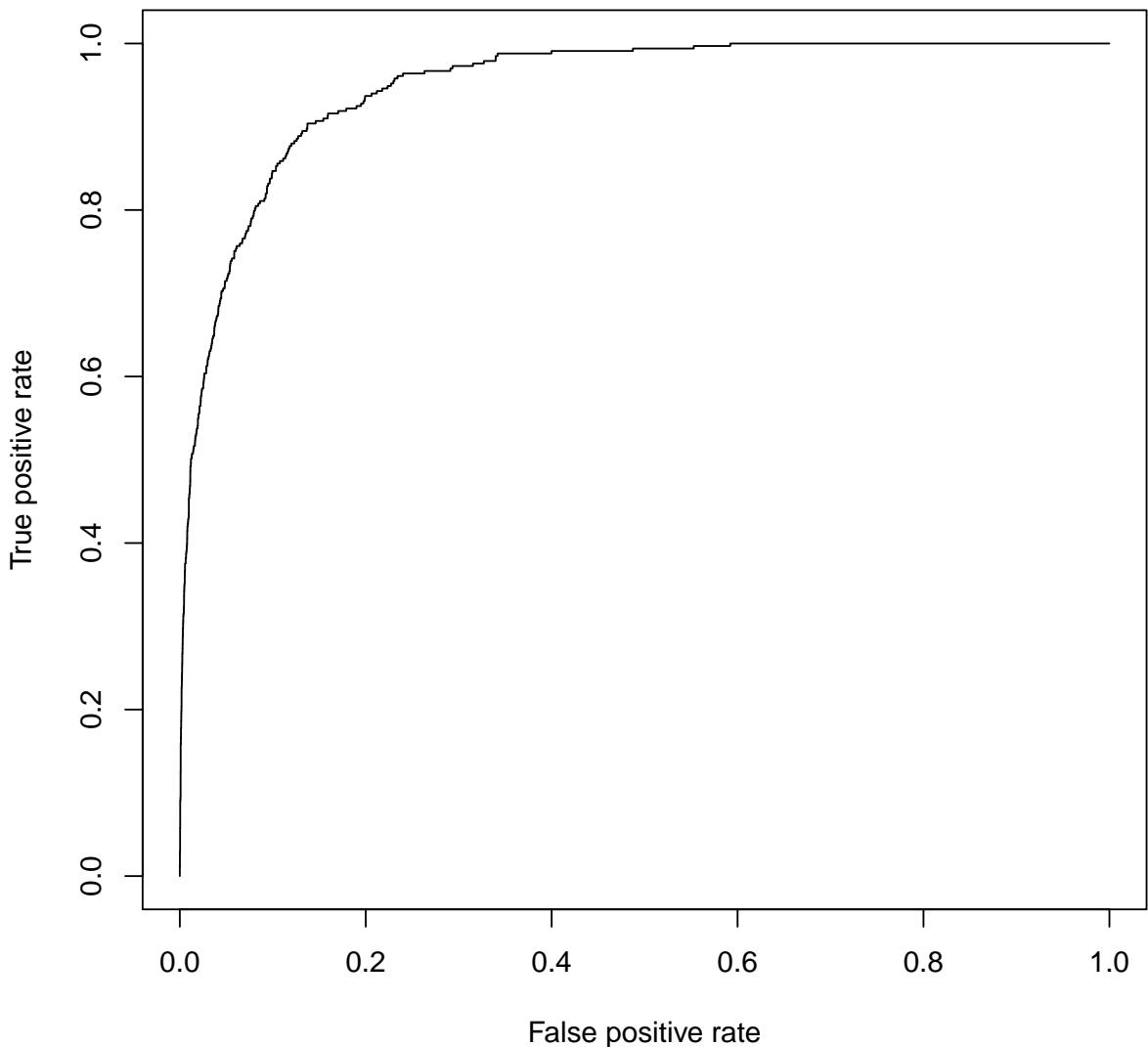
library(MASS)
library(lattice)
library(latticeExtra)
library(ISLR)
source("class.R")
data(Default)
gl <- glm(default ~ balance + student, data = Default,
           family = binomial(link = "logit"))
Default.sorted <- Default[order(Default$balance),
]
xyplot(predict(gl, Default.sorted, type = "response") ~
  balance, groups = student, data = Default.sorted,
  type = "l", auto.key = list(corner = c(0,
  1), title = "Student", lines = TRUE,
  points = FALSE), xlab = "Credit Card Balance",

```

```
ylab = "Default Rate") + layer_(panel.superpose(x = Default.sorted$default ==  
"Yes", panel.groups = function(x, y,  
...) panel.abline(h = mean(x), ...),  
lty = "dashed", ...))
```



```
roc <- ROC(predict(gl, Default), Default$default)  
plot(roc)
```



```
AUC(predict(glm, Default), Default$default)

## [1] 0.9495476

error.fun.auc <- function(true, predicted) {
  -AUC(predicted, true)
}
tune(glm, default ~ ., data = Default, family = binomial(link = "logit"),
  tunecontrol = tune.control(error.fun = error.fun.auc))

##
## Error estimation of 'glm' using 10-fold cross validation: -0.9502396

table(predicted = my.predict.glm(glm, measure = "max"),
  actual = Default$default)
```

```

##           actual
## predicted   No  Yes
##          No 8507   41
##          Yes 1160  292



```

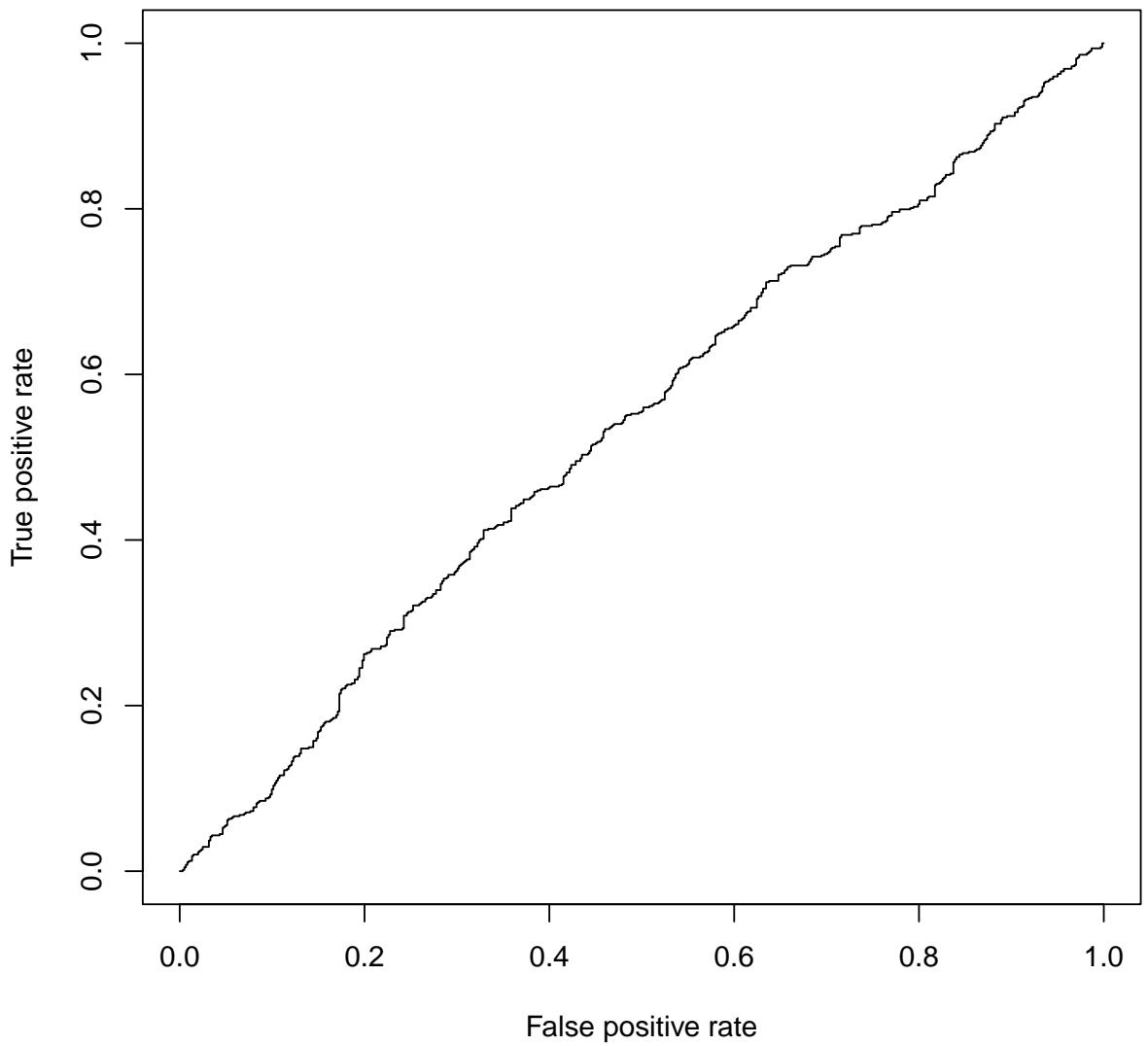
5.3 Smarket

```
library(ISLR)
library(MASS)
data(Smarket)

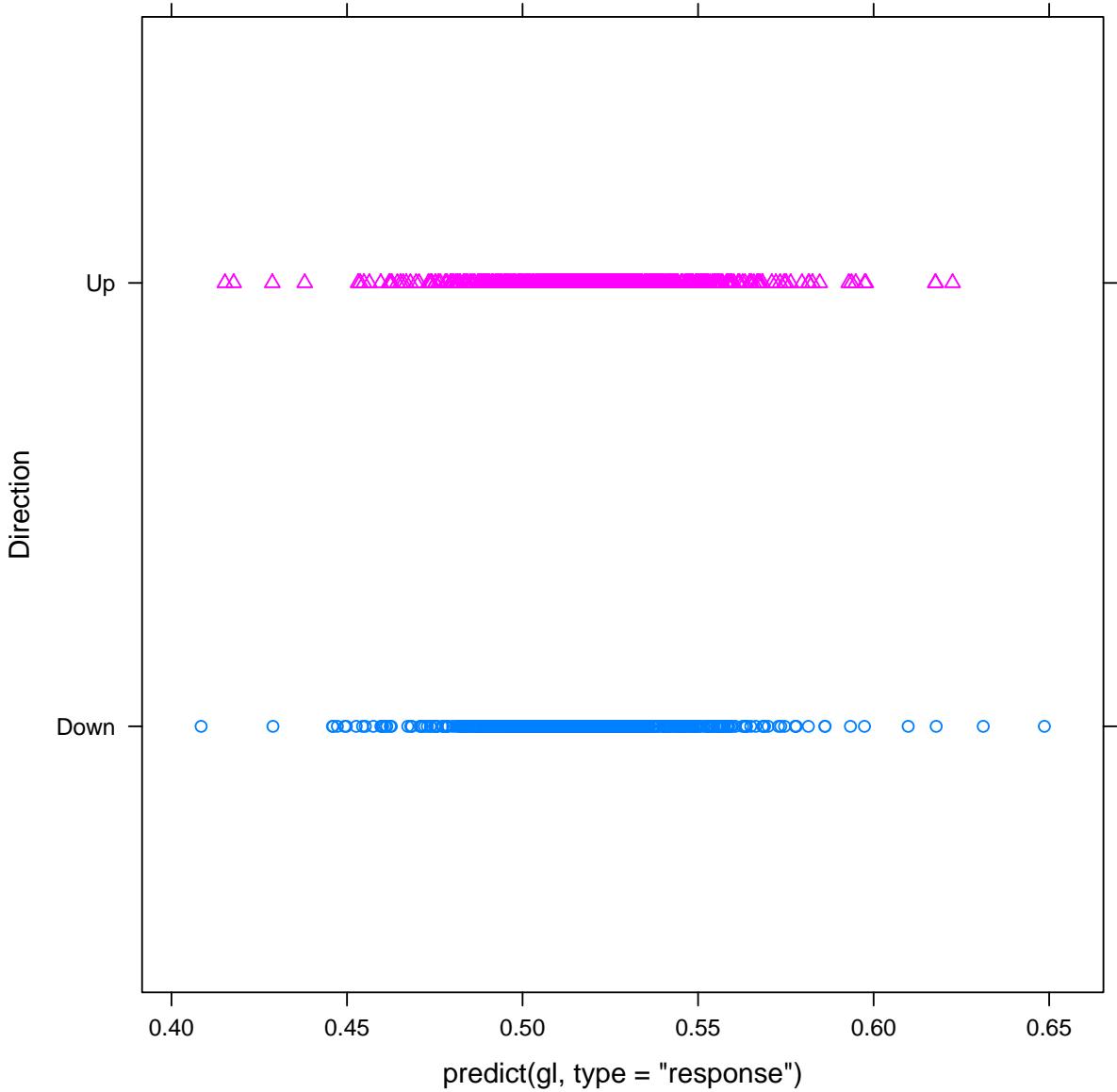
gl <- glm(Direction ~ Volume + Lag1 + Lag2 +
  Lag3 + Lag4 + Lag5, data = Smarket, family = binomial(link = "logit"))
summary(gl)

##
## Call:
## glm(formula = Direction ~ Volume + Lag1 + Lag2 + Lag3 + Lag4 +
##       Lag5, family = binomial(link = "logit"), data = Smarket)
##
## Deviance Residuals:
##    Min      1Q  Median      3Q     Max 
## -1.446  -1.203   1.065   1.145   1.326 
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)    
## (Intercept) -0.126000  0.240736 -0.523   0.601    
## Volume       0.135441  0.158360  0.855   0.392    
## Lag1        -0.073074  0.050167 -1.457   0.145    
## Lag2        -0.042301  0.050086 -0.845   0.398    
## Lag3         0.011085  0.049939  0.222   0.824    
## Lag4         0.009359  0.049974  0.187   0.851    
## Lag5         0.010313  0.049511  0.208   0.835    
## 
## (Dispersion parameter for binomial family taken to be 1)
##
## Null deviance: 1731.2 on 1249 degrees of freedom
## Residual deviance: 1727.6 on 1243 degrees of freedom
## AIC: 1741.6
##
## Number of Fisher Scoring iterations: 3

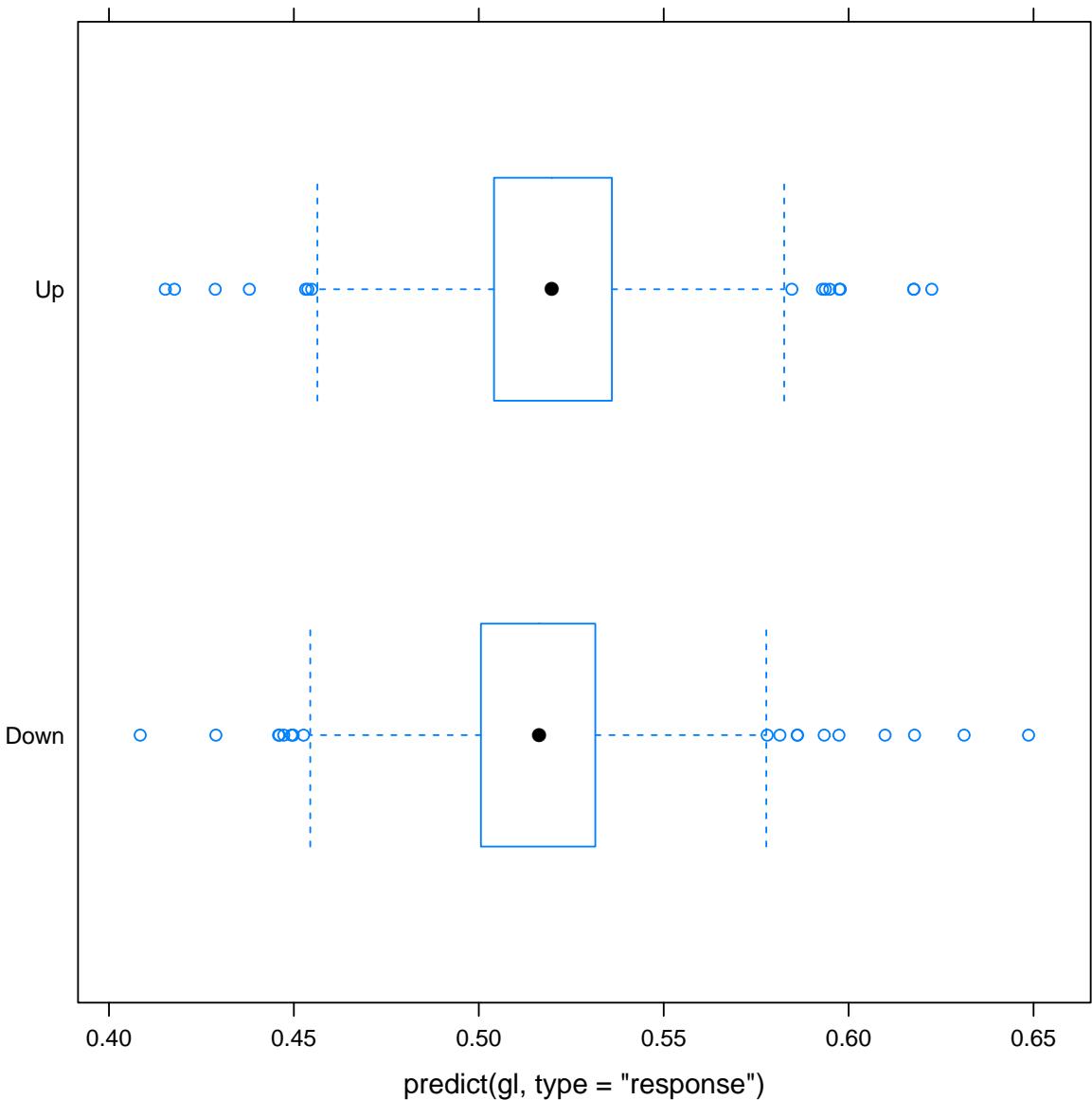
source("class.R")
roc <- ROC(predict(gl), Smarket$Direction)
plot(roc)
```



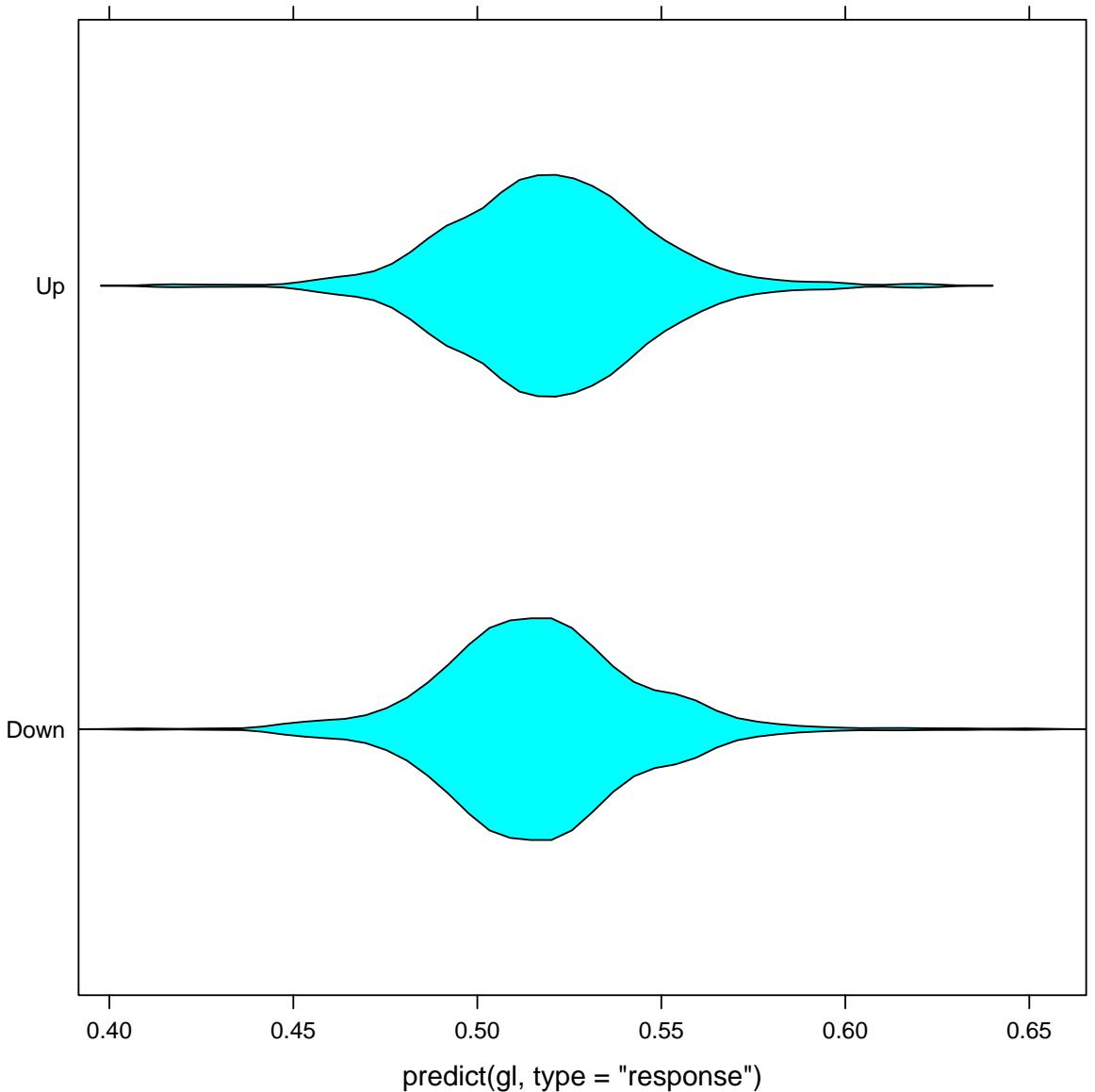
```
AUC(predict(gl), Smarket$Direction)  
## [1] 0.5387341  
  
xyplot(Direction ~ predict(gl, type = "response"),  
       data = Smarket, groups = Direction, par.settings = simpleTheme(pch = 1:2))
```



```
bwplot(Direction ~ predict(gl, type = "response"),
        data = Smarket)
```



```
bwplot(Direction ~ predict(gl, type = "response"),
        data = Smarket, panel = panel.violin)
```



```

gl <- glm(Direction ~ Volume + Lag1 + Lag2,
            data = Smarket, family = binomial(link = "logit"))
summary(gl)

##
## Call:
## glm(formula = Direction ~ Volume + Lag1 + Lag2, family = binomial(link = "logit"))
##      data = Smarket)
##
## Deviance Residuals:
##      Min        1Q    Median        3Q       Max
## -1.452   -1.203    1.068    1.146    1.331
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)

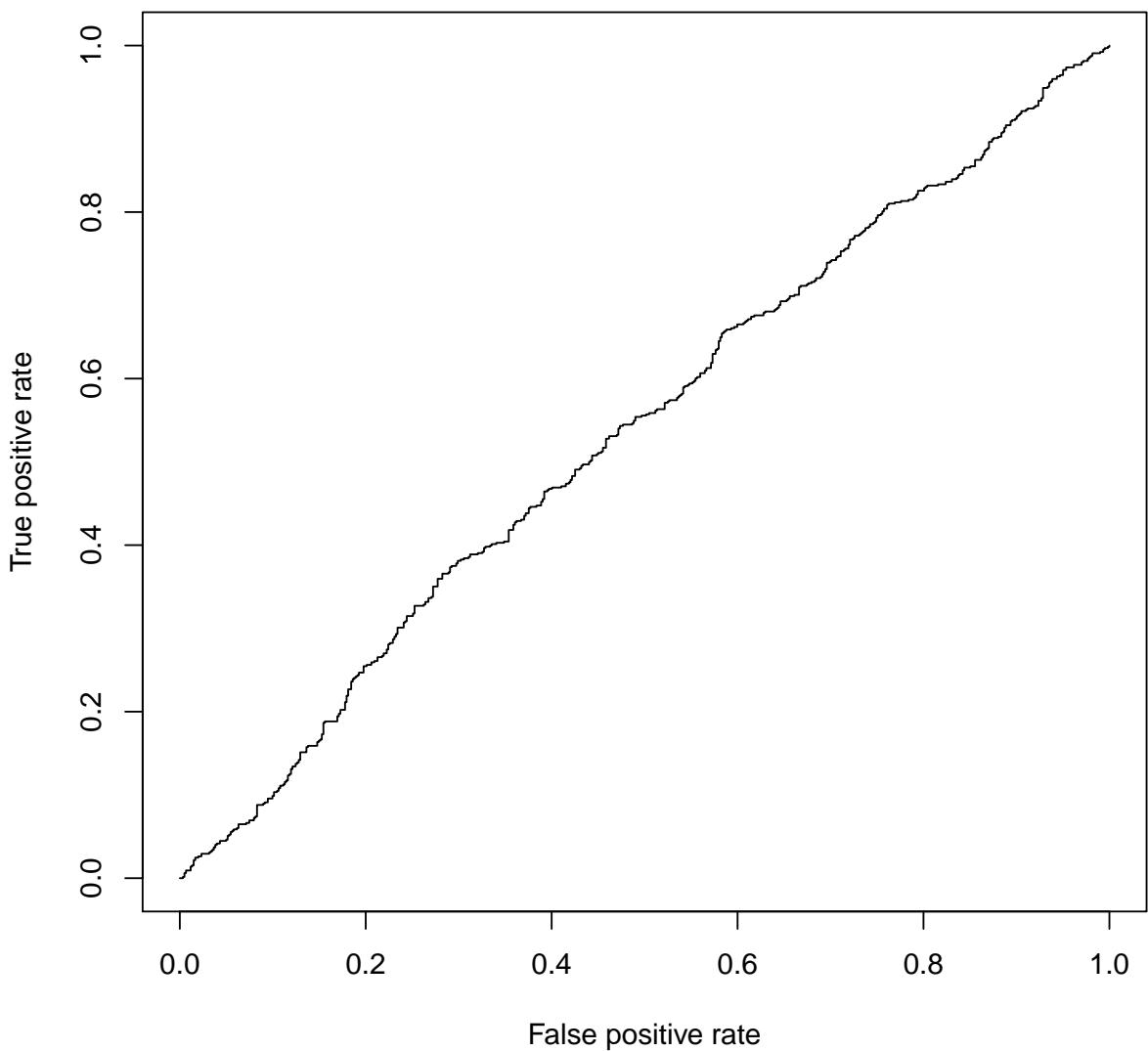
```

```

## (Intercept) -0.12058 0.24018 -0.502 0.616
## Volume      0.13184 0.15799 0.835 0.404
## Lag1        -0.07326 0.05017 -1.460 0.144
## Lag2        -0.04279 0.05006 -0.855 0.393
##
## (Dispersion parameter for binomial family taken to be 1)
##
## Null deviance: 1731.2 on 1249 degrees of freedom
## Residual deviance: 1727.7 on 1246 degrees of freedom
## AIC: 1735.7
##
## Number of Fisher Scoring iterations: 3

roc <- ROC(predict(gl), Smarket$Direction)
plot(roc)

```



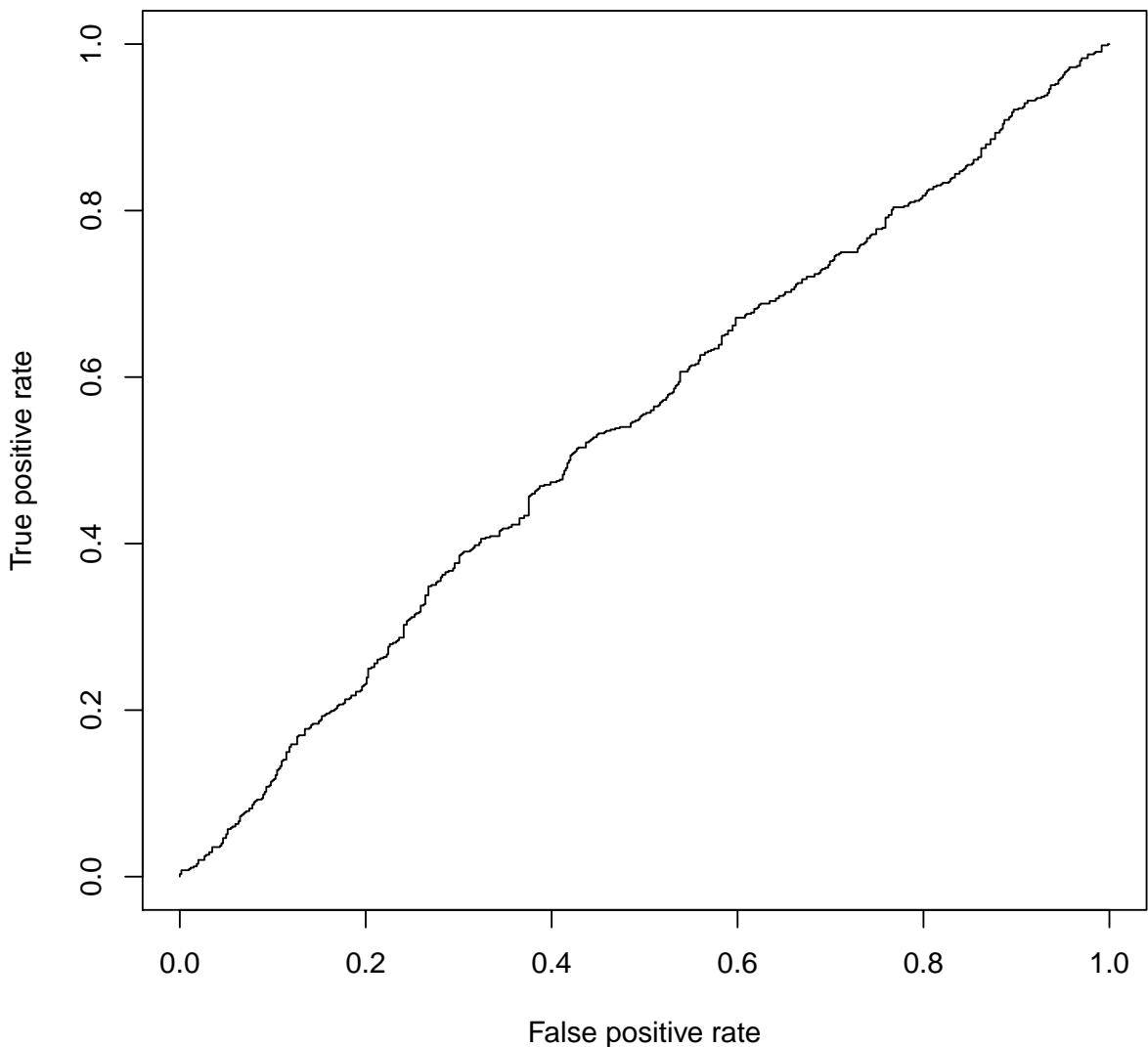
```

AUC(predict(gl), Smarket$Direction)

## [1] 0.537096

gl <- glm(Direction ~ poly(Volume, Lag1,
  Lag2, degree = 2), data = Smarket, family = binomial(link = "logit"))
roc <- ROC(predict(gl), Smarket$Direction)
plot(roc)

```



```

AUC(predict(gl), Smarket$Direction)

## [1] 0.5401183

library(e1071)
tn.glm <- tune(glm, Direction ~ poly(Volume,

```

```

Lag1, Lag2, degree = 2), data = Smarket,
family = binomial(link = "logit"), predict.func = simple.predict.glm,
tunecontrol = tune.control(cross = 100))
tn.mglm <- tune(glm, Direction ~ poly(Volume,
Lag1, Lag2, degree = 2), data = Smarket,
family = binomial(link = "logit"), predict.func = function(...) my.predict.glm(
measure = "err"), tunecontrol = tune.control(cross = 100))
tn.glm

##
## Error estimation of 'glm' using 100-fold cross validation: 0.5007692

tn.mglm

##
## Error estimation of 'glm' using 100-fold cross validation: 0.4994231

tn.mglm$performances$dispersion

## [1] 0.1389086

tn.qda <- tune(qda, Direction ~ Volume +
Lag1 + Lag2, data = Smarket, predict.func = simple.predict.da,
tunecontrol = tune.control(cross = 100))
tn.qda

##
## Error estimation of 'qda' using 100-fold cross validation: 0.5196154

Smarket.train <- subset(Smarket, Year <=
2004)
Smarket.test <- subset(Smarket, Year > 2004)
qd <- qda(Direction ~ Lag1 + Lag2, data = Smarket.train)
cm.train <- table(actual = Smarket.train$Direction,
predicted = predict(qd, Smarket.train)$class)
cm.test <- table(actual = Smarket.test$Direction,
predicted = predict(qd, Smarket.test)$class)
chisq.test(cm.test)

##
## Pearson's Chi-squared test with Yates' continuity
## correction
##
## data: cm.test
## X-squared = 5.6585, df = 1, p-value = 0.01737

```

5.4 banknote

```

source("class.R")
banknote <- read.csv("banknote/data_banknote_authentication.txt",
  header = FALSE, comment.char = "#")
colnames(banknote) <- c("variance", "skewness",
  "curtosis", "entropy", "class")
banknote$class <- factor(banknote$class,
  labels = c("genuine", "forged"))
summary(banknote)

##      variance           skewness          curtosis
##  Min.   :-7.0421   Min.   :-13.773   Min.   :-5.2861
##  1st Qu.:-1.7730   1st Qu.: -1.708   1st Qu.:-1.5750
##  Median : 0.4962   Median :  2.320   Median : 0.6166
##  Mean   : 0.4337   Mean   :  1.922   Mean   : 1.3976
##  3rd Qu.: 2.8215   3rd Qu.:  6.815   3rd Qu.: 3.1793
##  Max.   : 6.8248   Max.   : 12.952   Max.   :17.9274
##      entropy           class
##  Min.   :-8.5482   genuine:762
##  1st Qu.:-2.4135   forged :610
##  Median :-0.5867
##  Mean   :-1.1917
##  3rd Qu.: 0.3948
##  Max.   : 2.4495

nb <- naiveBayes(class ~ ., data = banknote)
ld <- lda(class ~ ., data = banknote)
qd <- qda(class ~ ., data = banknote)
gl <- glm(class ~ ., data = banknote, family = binomial(link = "logit"))

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

table(predicted = predict(nb, banknote),
  actual = banknote$class)

##      actual
## predicted genuine forged
##  genuine     671    127
##  forged       91    483

tn <- tune(naiveBayes, class ~ ., data = banknote)
tn

##
## Error estimation of 'naiveBayes' using 10-fold cross validation: 0.1581561

tn$performances$dispersion

## [1] 0.02276673

```



```

## 
## Error estimation of 'qda' using 10-fold cross validation: 0.01676716
tn.glm

## 
## Error estimation of 'glm' using 10-fold cross validation: 0.01092246
tn.nb

## 
## Error estimation of 'naiveBayes' using 10-fold cross validation: 0.1588649
tn.mlda

## 
## Error estimation of 'my.lda' using 10-fold cross validation: 0.01166296
tn.mqda

## 
## Error estimation of 'my.qda' using 100-fold cross validation: 0.0007142857
summary(tn.mglm$best.model)

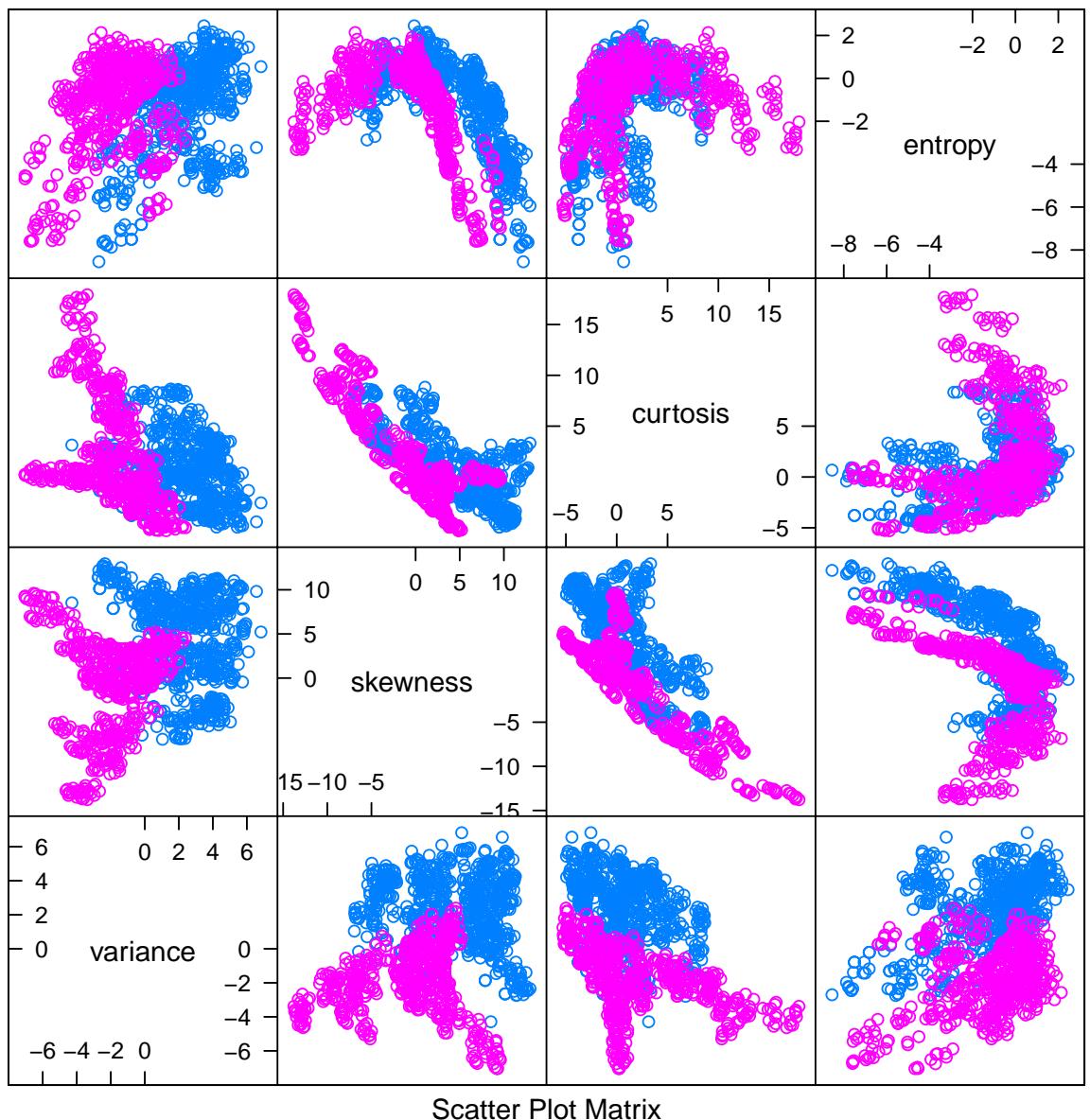
## 
## Call:
## best.tune(method = glm, train.x = class ~ ., data = banknote,
## predict.func = function(...) my.predict.glm(..., measure = "err"),
## family = binomial(link = "logit"))
## 
## Deviance Residuals:
##      Min       1Q   Median       3Q      Max
## -1.70001   0.00000   0.00000   0.00029   2.24614
## 
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept)  7.3218    1.5589   4.697 2.64e-06 ***
## variance     -7.8593    1.7383  -4.521 6.15e-06 ***
## skewness     -4.1910    0.9041  -4.635 3.56e-06 ***
## curtosis     -5.2874    1.1612  -4.553 5.28e-06 ***
## entropy      -0.6053    0.3307  -1.830  0.0672 .
## --- 
## Signif. codes:
## 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
## 
## (Dispersion parameter for binomial family taken to be 1)
## 
## Null deviance: 1885.122 on 1371 degrees of freedom
## Residual deviance: 49.891 on 1367 degrees of freedom
## AIC: 59.891
## 
## Number of Fisher Scoring iterations: 12

```

```
summary(tn.mqda$best.model)

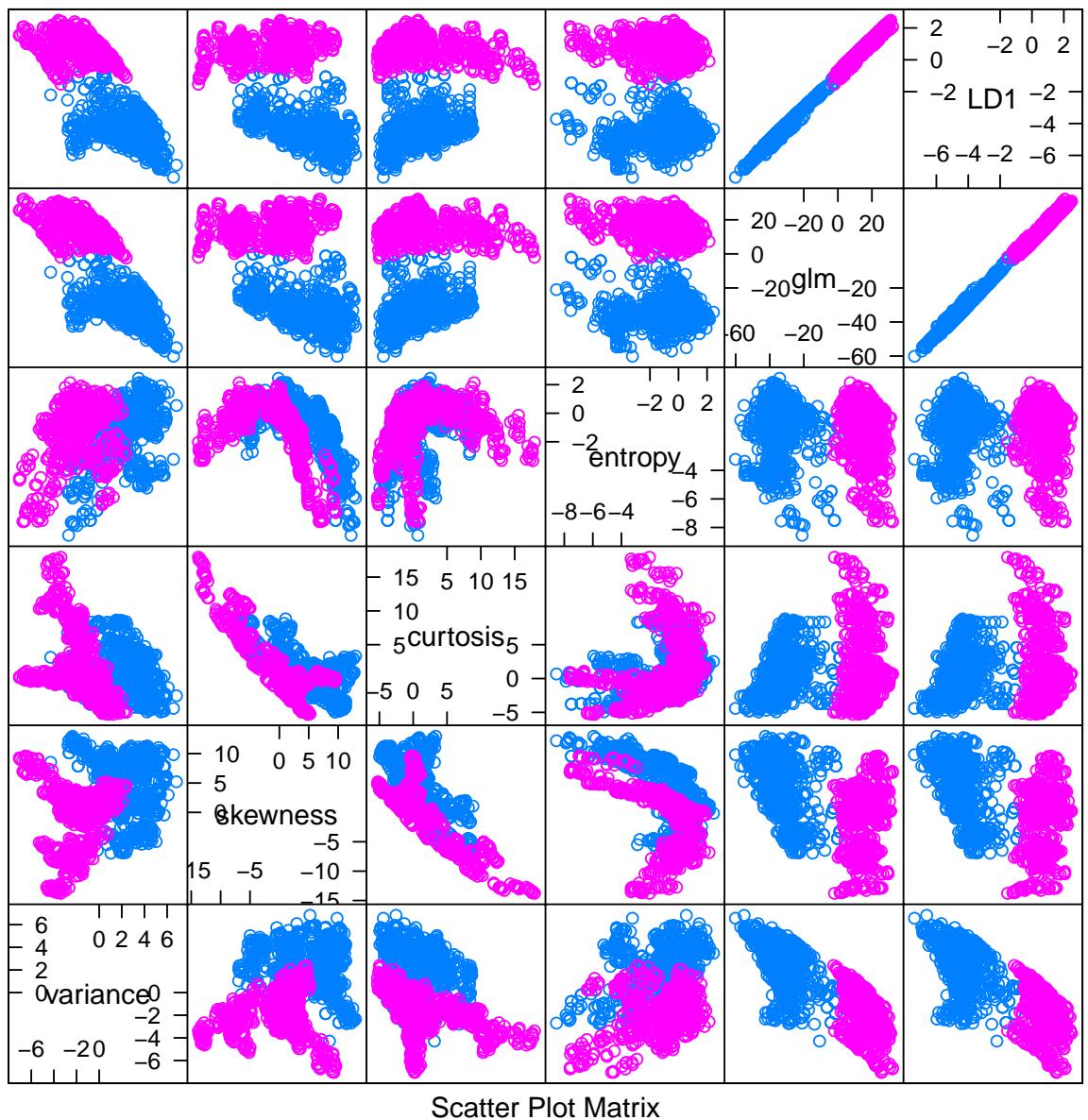
##          Length Class     Mode
## prior      2    -none-   numeric
## counts     2    -none-   numeric
## means      8    -none-   numeric
## scaling   32    -none-   numeric
## ldet       2    -none-   numeric
## lev        2    -none-   character
## N          1    -none-   numeric
## call       6    -none-   call
## terms      3    terms   call
## xlevels    0    -none-   list
## data       5    data.frame list

splom(subset(banknote, select = -class),
      groups = banknote$class)
```

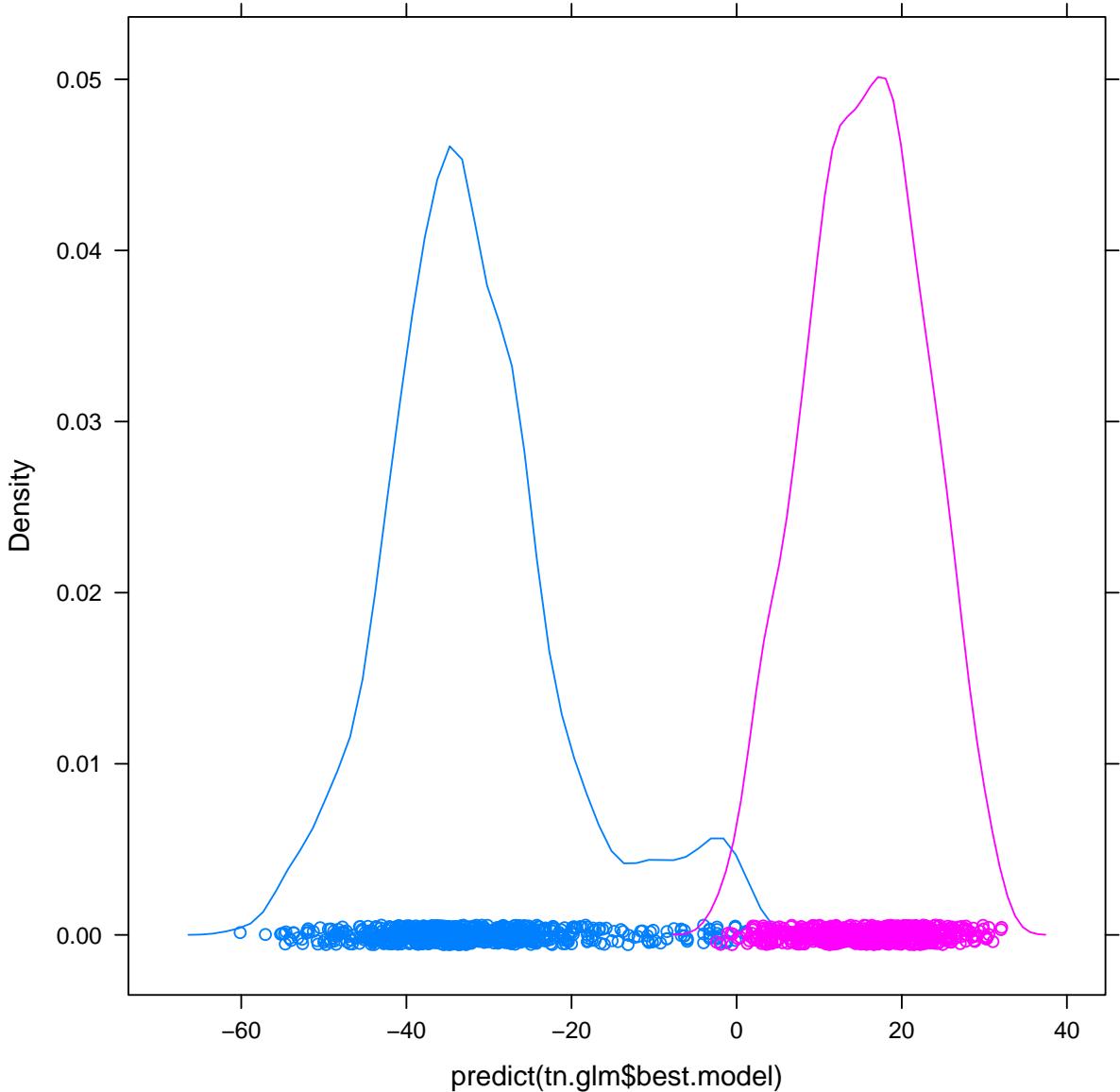


Scatter Plot Matrix

```
splom(cbind(subset(banknote, select = -class),
            glm = predict(tn.mglm$best.model), lda = as.matrix(subset(banknote,
            select = -class)) %*% tn.mlida$best.model$scaling),
            groups = banknote$class)
```



```
densityplot(~predict(tn.glm$best.model),
            groups = banknote$class)
```



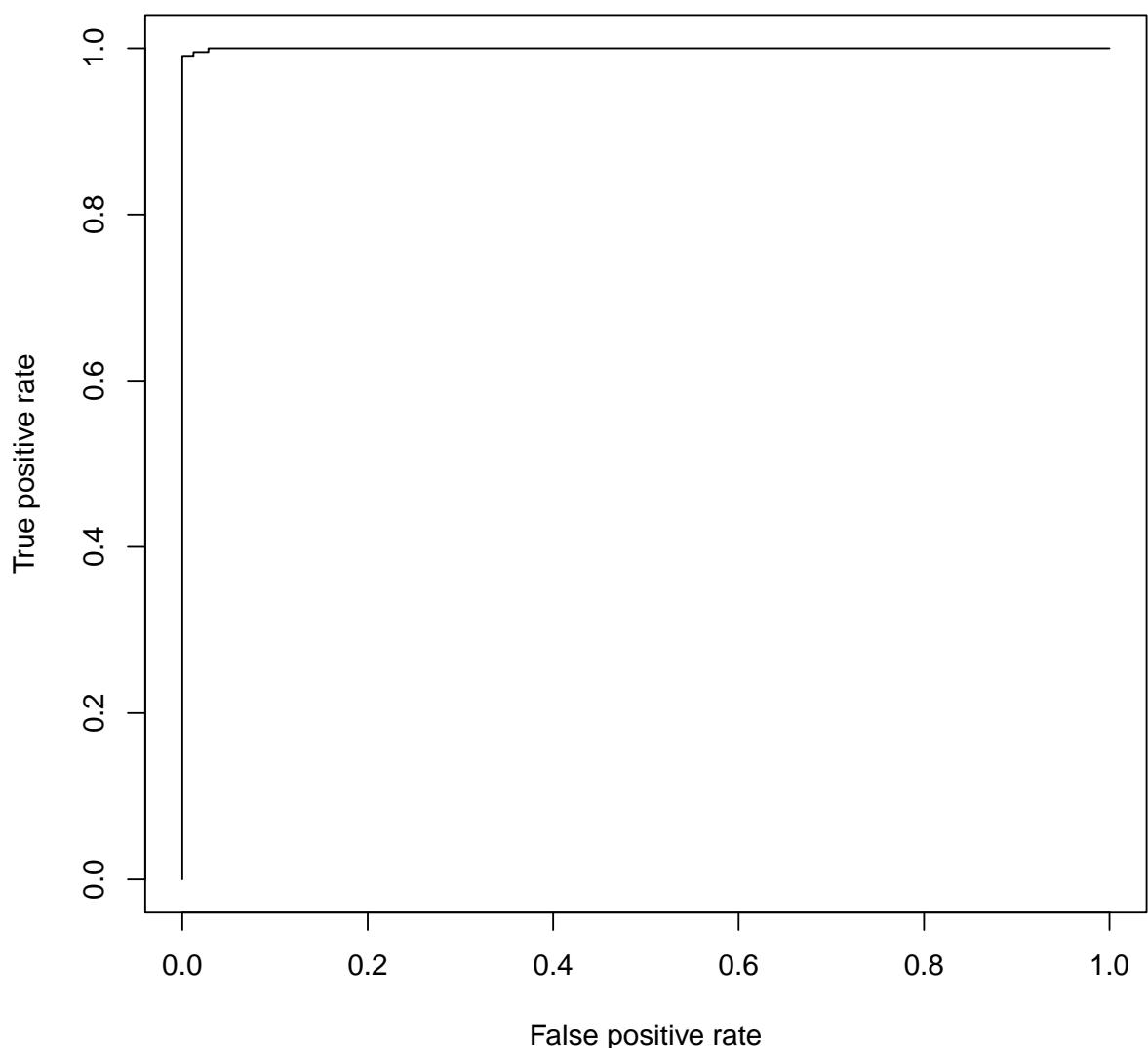
```

train <- sample(nrow(banknote), size = 0.66 *
  nrow(banknote))
gl <- glm(class ~ ., data = banknote, subset = train,
  family = binomial)

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

roc <- ROC(predict(gl, banknote[-train, ],
  type = "response"), as.numeric(banknote[-train,
  ]$class))
plot(roc)

```



6 Материалы с занятия 17 октября

6.1 Bootstrap and CV

```
library(MASS)
library(lattice)
library(latticeExtra)
library(e1071)
library(boot)
library(mvtnorm)
source("class.R")
N <- 100
x <- rnorm(N)
```

```

b.np <- boot(x, function(data, subset) mean(data[subset]),  

             R = 999)  

boot.ci(b.np)

## Warning in boot.ci(b.np): bootstrap variances needed for studentized intervals

## BOOTSTRAP CONFIDENCE INTERVAL CALCULATIONS
## Based on 999 bootstrap replicates
##
## CALL :
## boot.ci(boot.out = b.np)
##
## Intervals :
## Level      Normal          Basic
## 95%   (-0.2851,  0.1013 )  (-0.2801,  0.1081 )
##
## Level      Percentile       BCa
## 95%   (-0.2924,  0.0958 )  (-0.2927,  0.0956 )
## Calculations and Intervals on Original Scale

boot.ci(b.np)$normal

## Warning in boot.ci(b.np): bootstrap variances needed for studentized intervals

##      conf
## [1,] 0.95 -0.2851178 0.1013173

t.test(x)

##
## One Sample t-test
##
## data: x
## t = -0.9086, df = 99, p-value = 0.3658
## alternative hypothesis: true mean is not equal to 0
## 95 percent confidence interval:
## -0.2933620  0.1090756
## sample estimates:
## mean of x
## -0.09214323

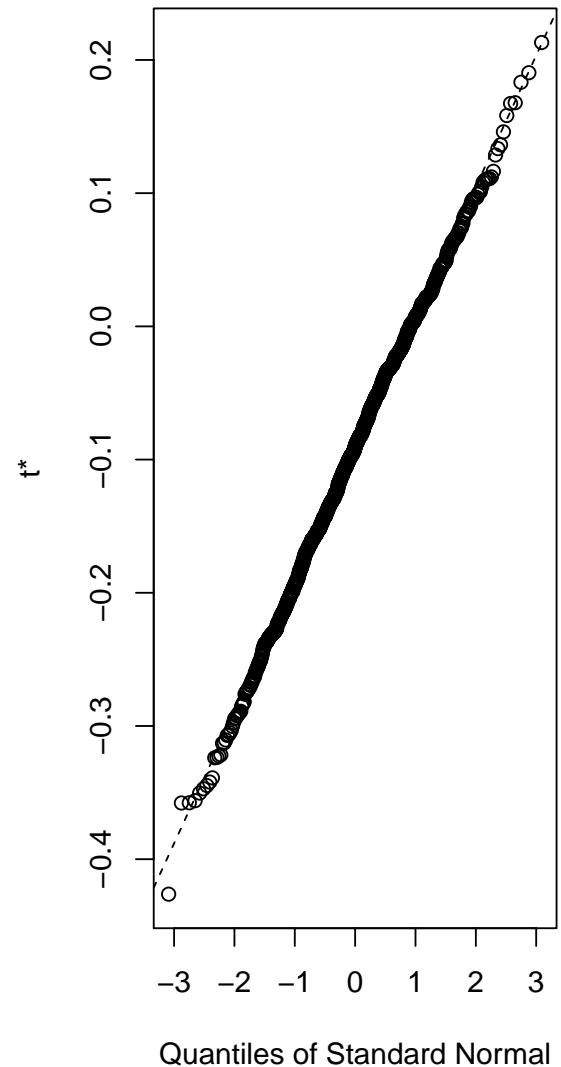
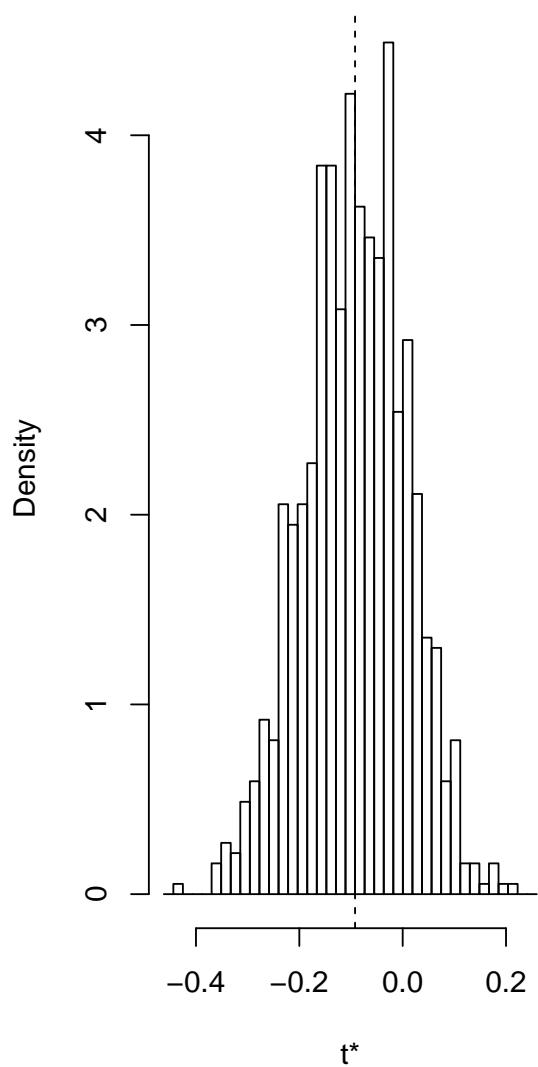
b.p <- boot(x, mean, R = 999, sim = "parametric",
            ran.gen = function(data, pars) rnorm(length(data),
                mean = pars$mean, sd = pars$sd),
            mle = list(mean = mean(x), sd = sd(x)))
boot.ci(b.p, type = "norm")

## BOOTSTRAP CONFIDENCE INTERVAL CALCULATIONS
## Based on 999 bootstrap replicates
##

```

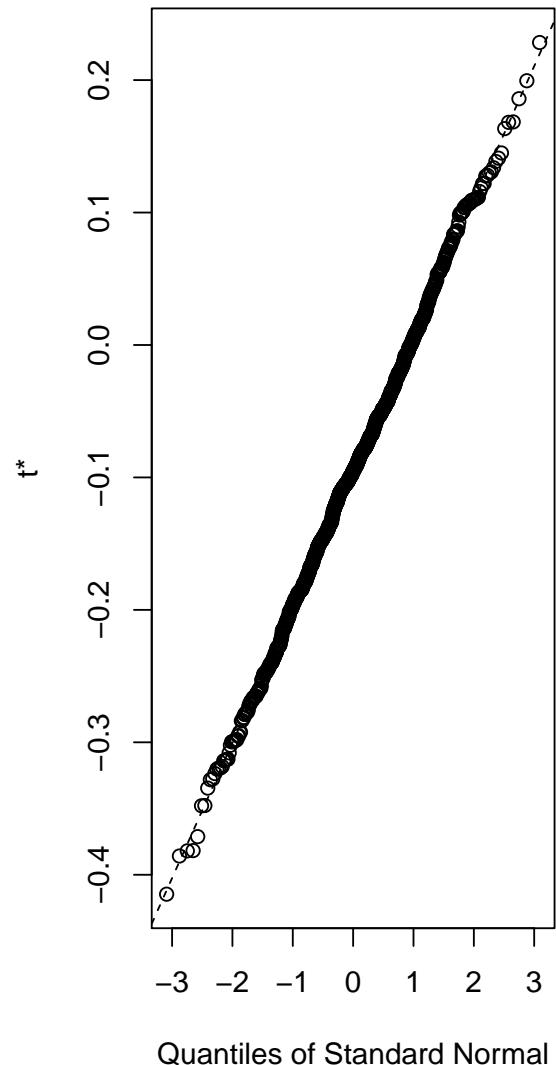
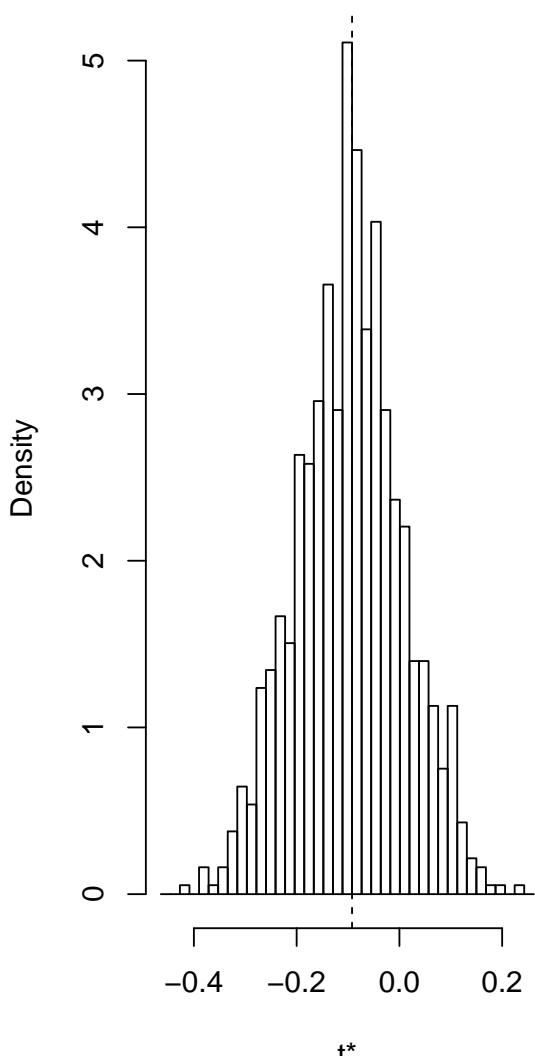
```
## CALL :  
## boot.ci(boot.out = b.p, type = "norm")  
##  
## Intervals :  
## Level      Normal  
## 95%   (-0.2885,  0.1120 )  
## Calculations and Intervals on Original Scale  
  
boot.ci(b.p, type = "perc")  
  
## BOOTSTRAP CONFIDENCE INTERVAL CALCULATIONS  
## Based on 999 bootstrap replicates  
##  
## CALL :  
## boot.ci(boot.out = b.p, type = "perc")  
##  
## Intervals :  
## Level      Percentile  
## 95%   (-0.2987,  0.1091 )  
## Calculations and Intervals on Original Scale  
  
plot(b.np)
```

Histogram of t



```
plot(b.p)
```

Histogram of t^*



```

x <- runif(N)
b.np <- boot(x, function(x, subset) max(x[subset]),
  R = 999)
b.p <- boot(x, max, R = 999, sim = "parametric",
  ran.gen = function(data, pars) runif(length(data),
    min = pars$min, max = pars$max),
  mle = list(min = min(x), max = max(x)))
boot.ci(b.np)

## Warning in boot.ci(b.np): bootstrap variances needed for studentized intervals

## BOOTSTRAP CONFIDENCE INTERVAL CALCULATIONS
## Based on 999 bootstrap replicates
##
## CALL :

```

```

## boot.ci(boot.out = b.np)
##
## Intervals :
## Level      Normal          Basic
## 95%   ( 0.9651,  1.0023 )   ( 0.9801,  1.0131 )
##
## Level      Percentile       BCa
## 95%   ( 0.9470,  0.9801 )   ( 0.9451,  0.9801 )
## Calculations and Intervals on Original Scale

boot.ci(b.p, type = "norm")

## BOOTSTRAP CONFIDENCE INTERVAL CALCULATIONS
## Based on 999 bootstrap replicates
##
## CALL :
## boot.ci(boot.out = b.p, type = "norm")
##
## Intervals :
## Level      Normal
## 95%   ( 0.9722,  1.0059 )
## Calculations and Intervals on Original Scale

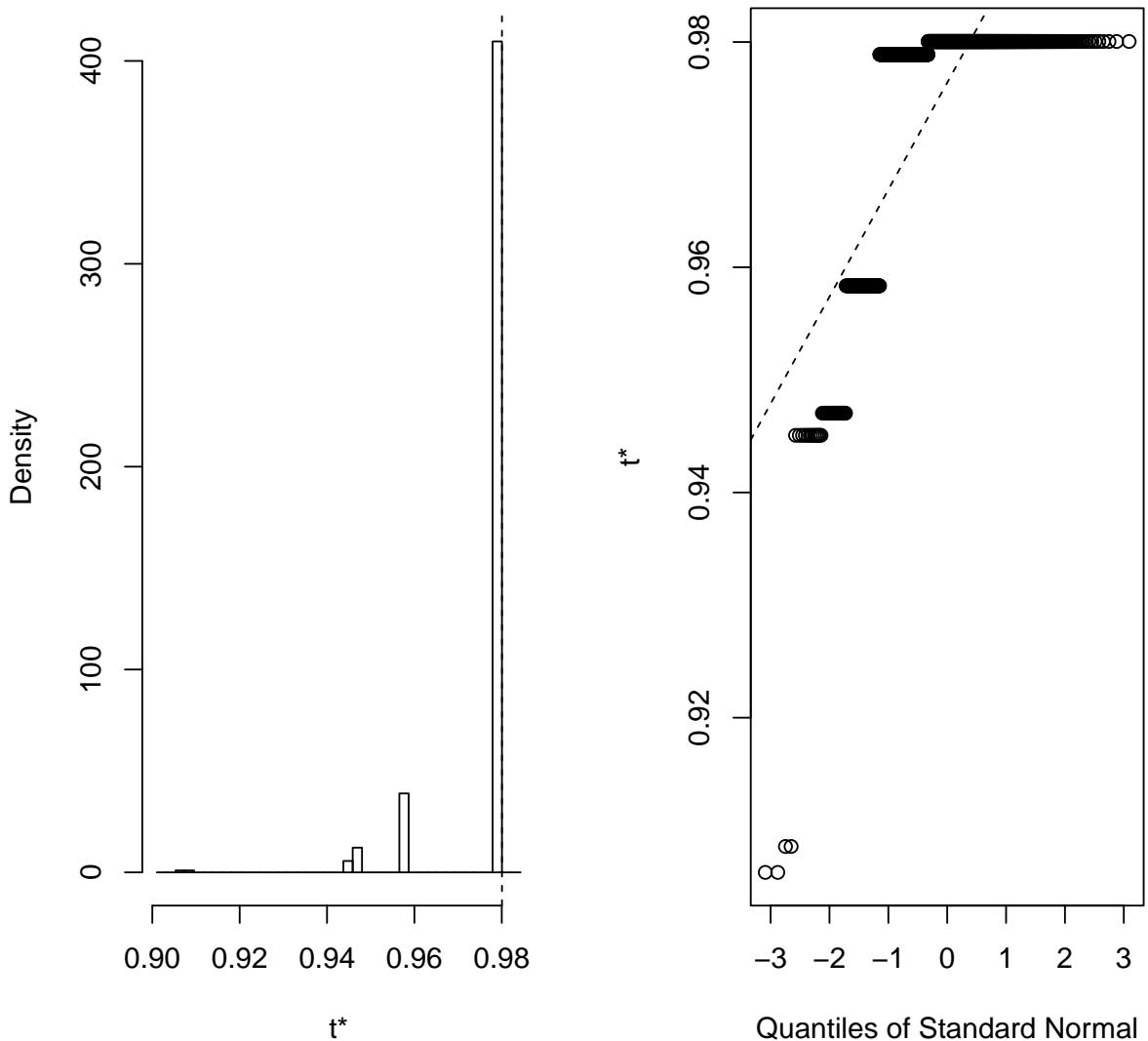
boot.ci(b.p, type = "perc")

## BOOTSTRAP CONFIDENCE INTERVAL CALCULATIONS
## Based on 999 bootstrap replicates
##
## CALL :
## boot.ci(boot.out = b.p, type = "perc")
##
## Intervals :
## Level      Percentile
## 95%   ( 0.9476,  0.9798 )
## Calculations and Intervals on Original Scale

plot(b.np)

```

Histogram of t



```

lda.model <- function(data, groups) {
  data <- as.matrix(data)
  means <- aggregate(data, list(groups = groups),
    mean)
  data <- data - as.matrix(means[match(groups,
    means$groups), -1])
  list(cov = cov(data), means = means[,
    -1, drop = FALSE])
}
model <- lda.model(subset(iris, select = -Species),
  iris$Species)
make.data <- function(data, groups, lda.model,
  size = nrow(data), groups.name = "Species") {
  ind <- sample(seq_along(levels(groups)),
    size = size, replace = TRUE)

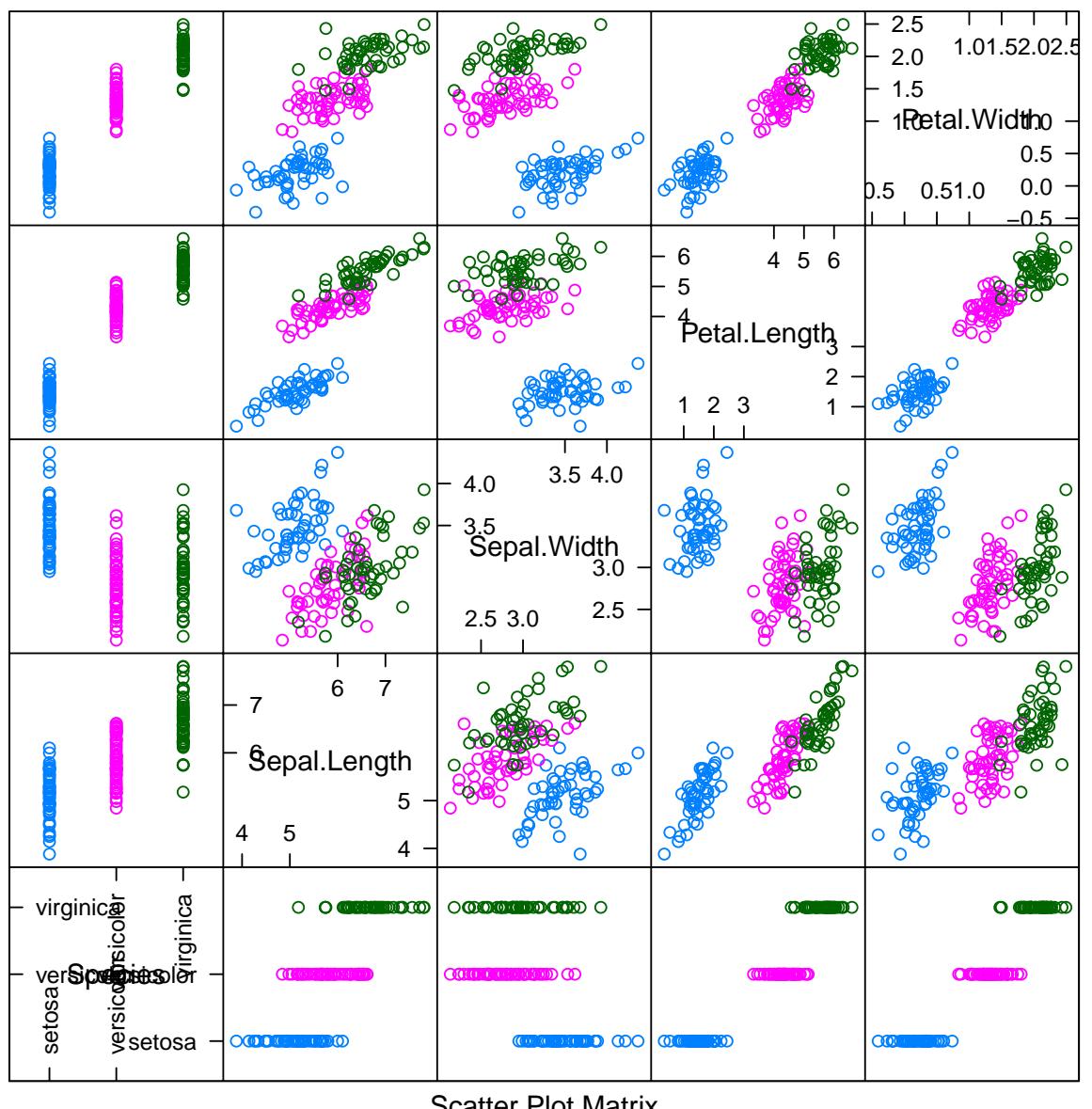
```

```

res <- data.frame(name = factor(levels(groups)[ind] ,
  levels = levels(groups)))
names(res) <- groups.name
mx <- lda.model$means[ind, ]
mx <- mx + rmvnorm(nrow(mx), sigma = lda.model$cov)
colnames(mx) <- colnames(lda.model$cov)
res <- cbind(res, as.data.frame(mx))
rownames(res) <- NULL
res
}

res <- make.data(iris, iris$Species, model)
splom(res, groups = res$Species)

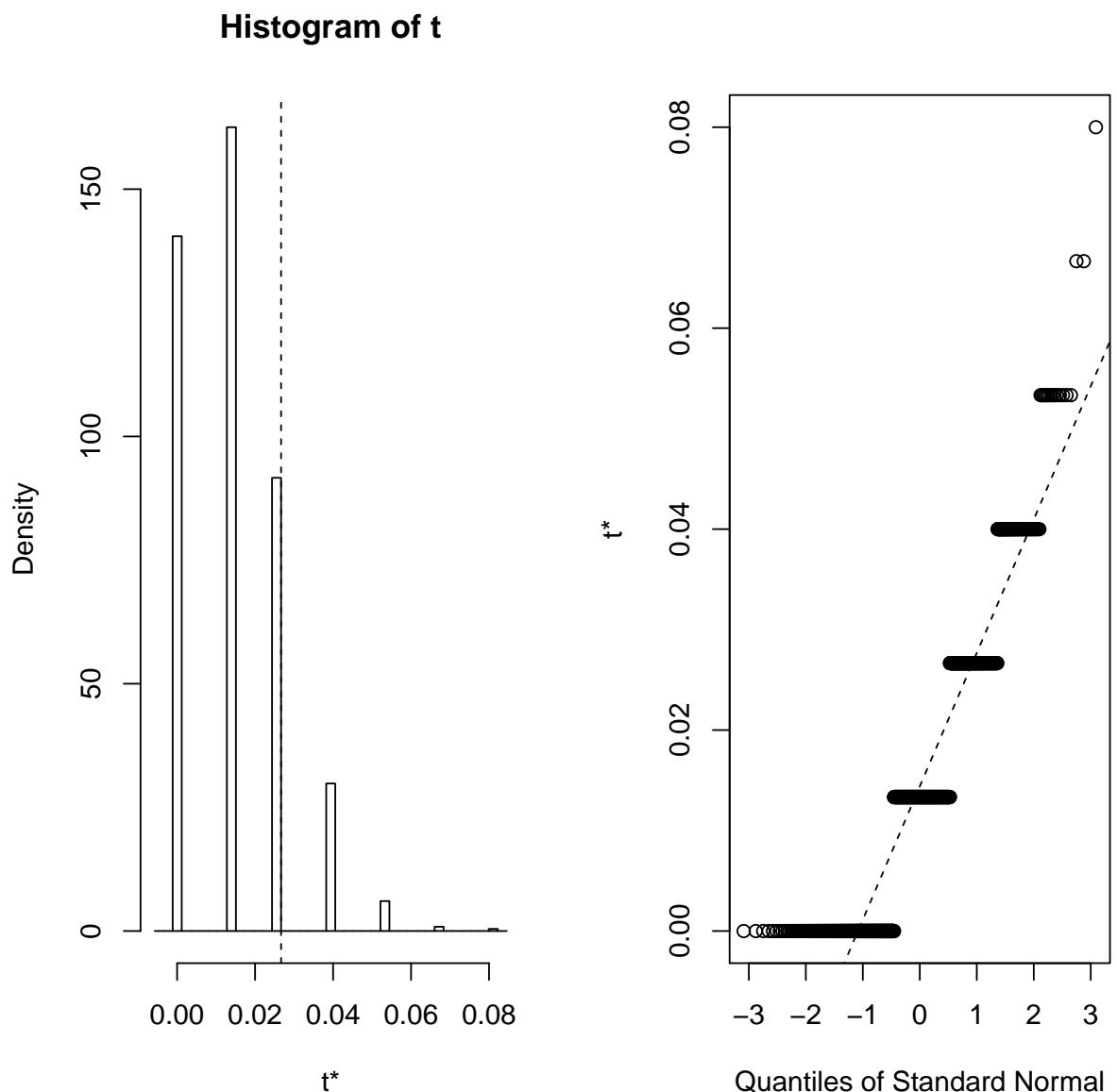
```



```

b <- boot(iris, function(data) {
  tn <- tune(my.lda, Species ~ ., data = data,
    predict.func = simple.predict.da,
    tunecontrol = tune.control(sampling = "fix",
      fix = 1/2))
  tn$best.performance
}, R = 999, sim = "parametric", ran.gen = function(data,
  mle, ..., size = 300) make.data(data,
  mle$groups, mle$lda.model, ...), mle = list(groups = iris$Species,
  lda.model = lda.model(subset(iris, select = -Species),
    groups = iris$Species)))
plot(b)

```



```

boot.ci(b, type = "perc")

## BOOTSTRAP CONFIDENCE INTERVAL CALCULATIONS
## Based on 999 bootstrap replicates
##
## CALL :
## boot.ci(boot.out = b, type = "perc")
##
## Intervals :
## Level      Percentile
## 95%    ( 0.00,  0.04 )
## Calculations and Intervals on Original Scale

Advertising <- read.csv("Advertising.csv")
Advertising$X <- NULL
degrees <- 1:10
tns.cv <- list()
tns.boot <- list()
mse <- rep(NA_real_, length(degrees))
for (i in seq_along(degrees)) {
  model <- Sales ~ Radio + poly(TV, degree = degrees[i])
  tns.cv[[i]] <- tune(lm, model, data = Advertising,
    tunecontrol = tune.control(sampling = "cross",
      cross = 15))
  tns.boot[[i]] <- tune(lm, model, data = Advertising,
    tunecontrol = tune.control(sampling = "boot",
      nboot = 100))
  mse[i] <- mean(residuals(lm(model, data = Advertising))^2)
}
cv.errors <- sapply(tns.cv, function(el) el$performance$error)
boot.errors <- sapply(tns.boot, function(el) el$performance$error)
degrees[which.min(cv.errors)]

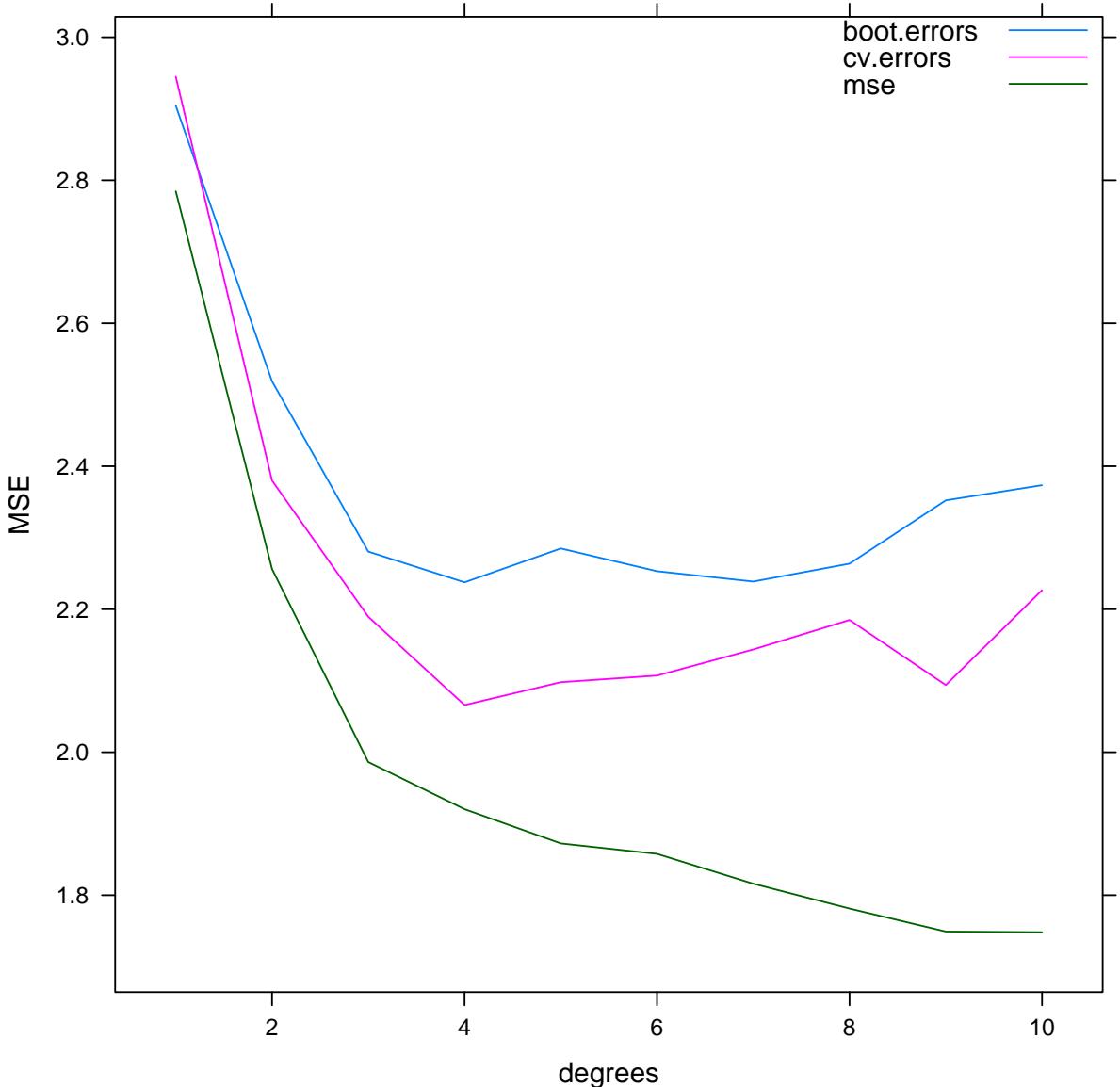
## [1] 4

degrees[which.min(boot.errors)]

## [1] 4

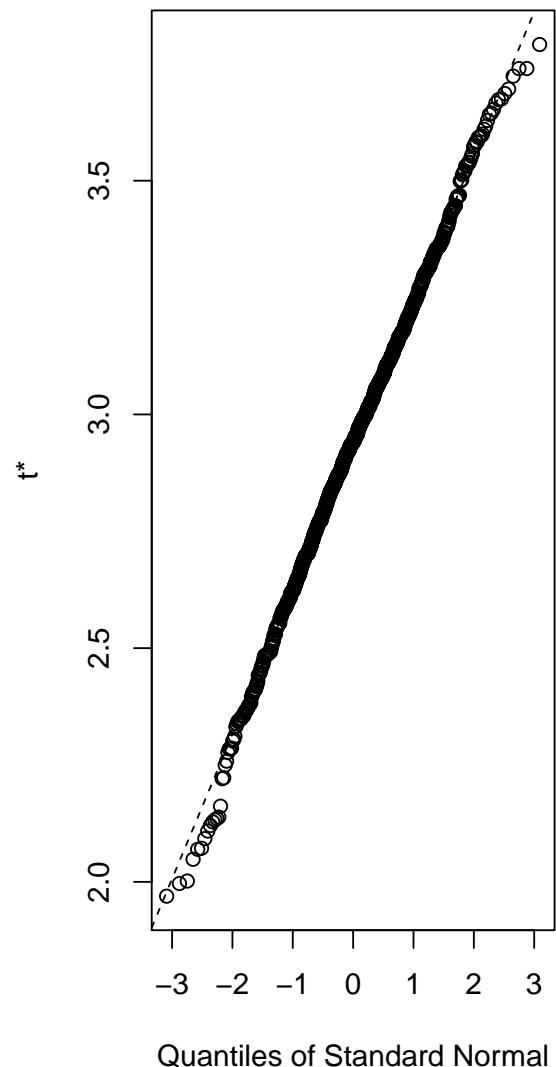
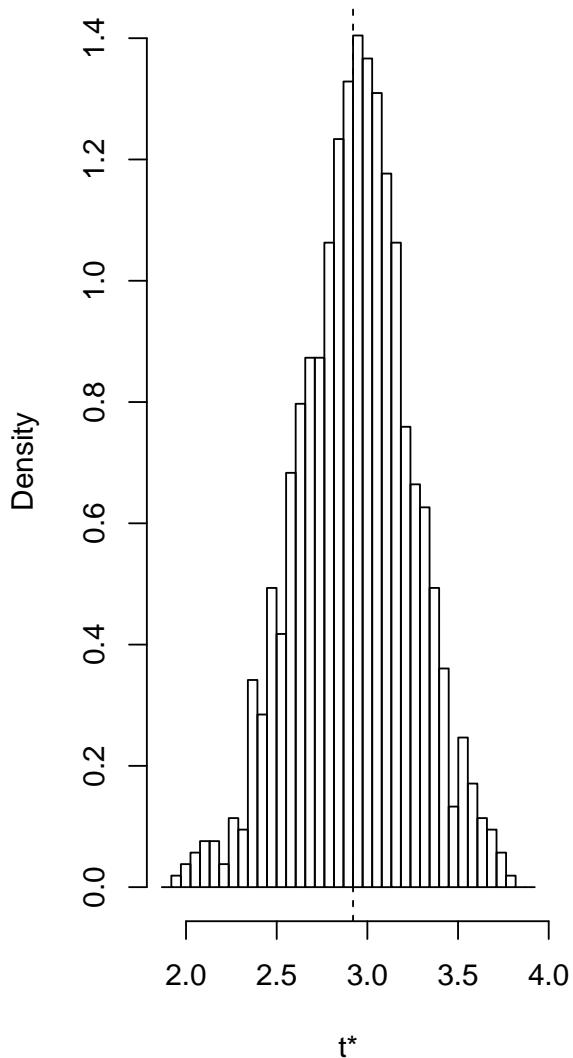
xyplot(boot.errors + cv.errors + mse ~ degrees,
  type = "l", auto.key = list(corner = c(1,
    1), lines = TRUE, points = FALSE),
  ylab = "MSE")

```



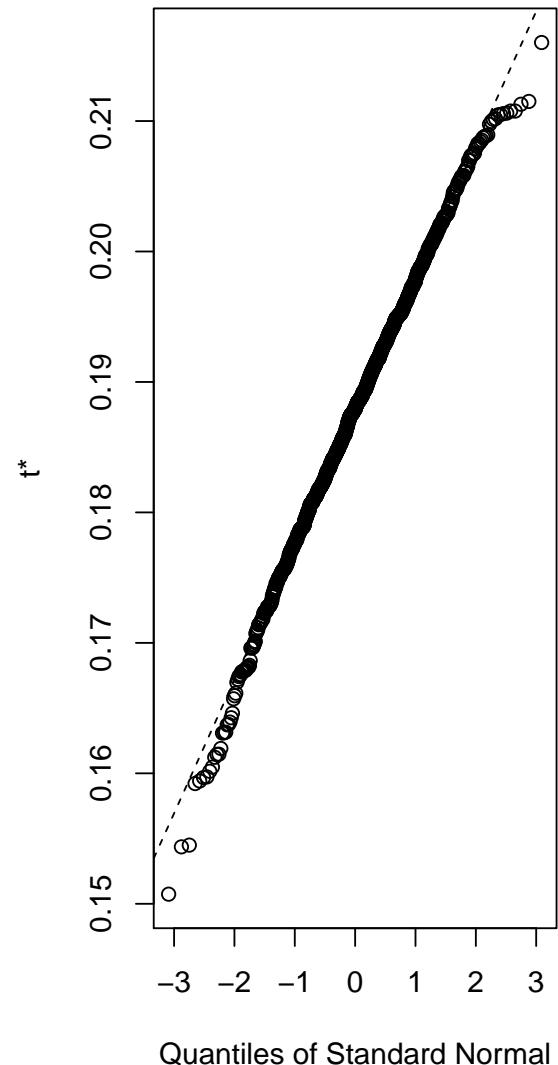
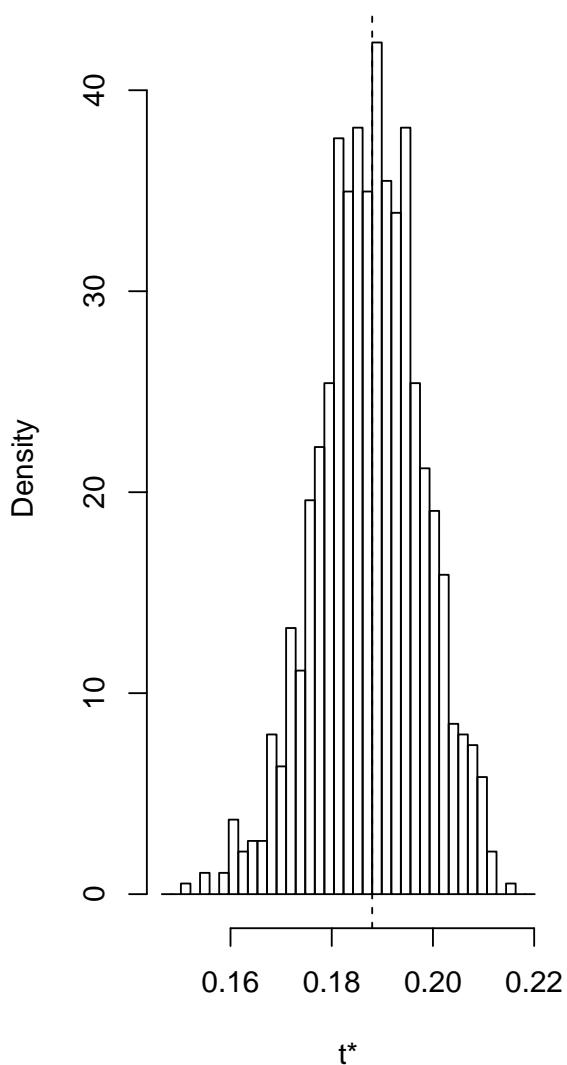
```
b <- boot(Advertising, function(...) coef(lm(Sales ~  
    Radio + TV, ...)), R = 999)  
plot(b, index = 1)
```

Histogram of t



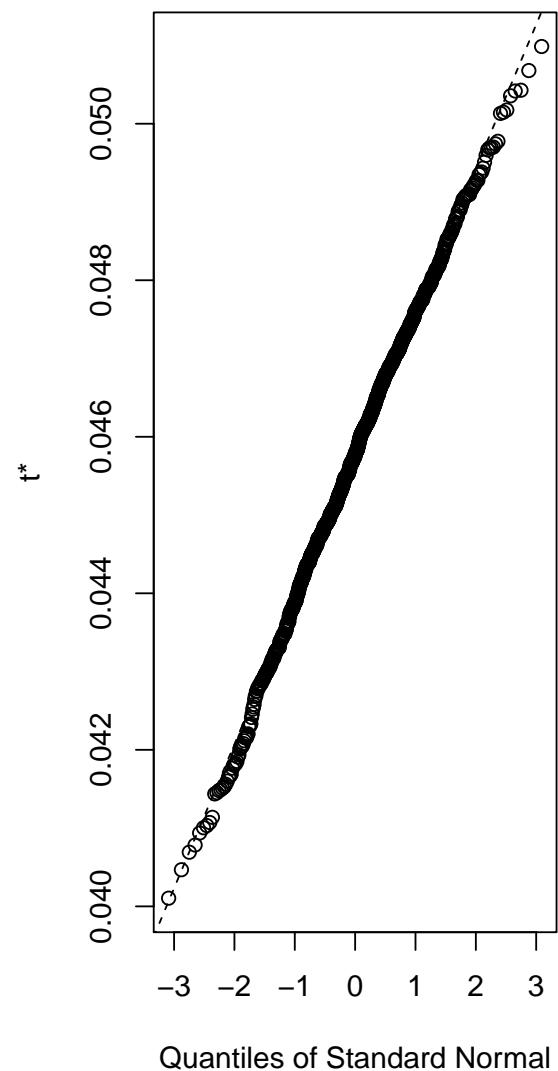
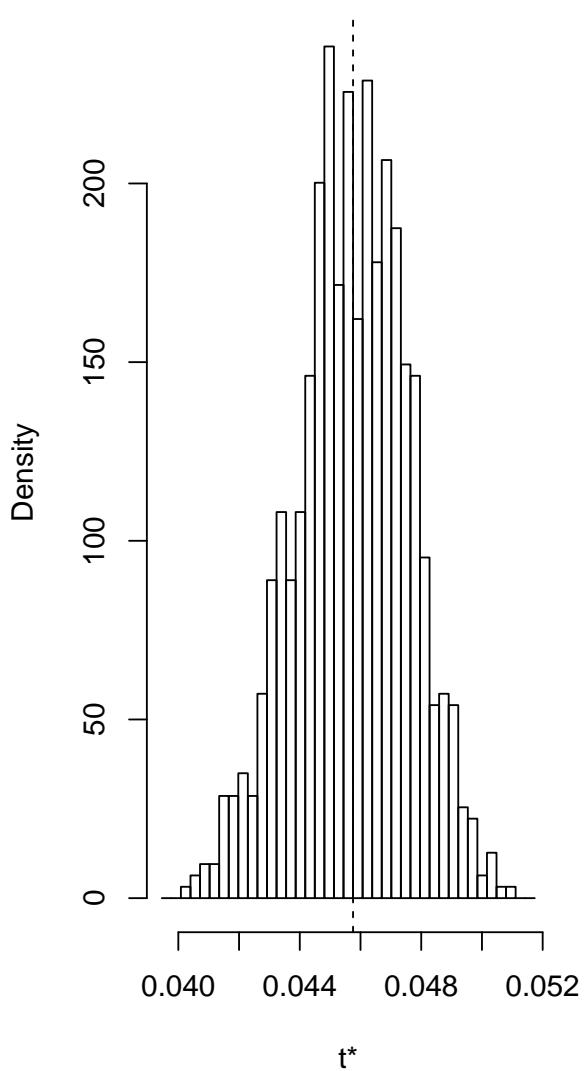
```
plot(b, index = 2)
```

Histogram of t



```
plot(b, index = 3)
```

Histogram of t^*



```

boot.ci(b, index = 1)

## Warning in boot.ci(b, index = 1): bootstrap variances needed for studentized
## intervals

## BOOTSTRAP CONFIDENCE INTERVAL CALCULATIONS
## Based on 999 bootstrap replicates
##
## CALL :
## boot.ci(boot.out = b, index = 1)
##
## Intervals :
## Level      Normal          Basic
## 95%   ( 2.303,  3.514 )  ( 2.290,  3.530 )
##
```

```

## Level      Percentile          BCa
## 95%   ( 2.312,  3.553 )   ( 2.255,  3.513 )
## Calculations and Intervals on Original Scale

boot.ci(b, index = 1)$percent

## Warning in boot.ci(b, index = 1): bootstrap variances needed for studentized
## intervals

##      conf
## [1,] 0.95 25 975 2.31171 3.55264

boot.ci(b, index = 2)$percent

## Warning in boot.ci(b, index = 2): bootstrap variances needed for studentized
## intervals

##      conf
## [1,] 0.95 25 975 0.1669722 0.2074499

boot.ci(b, index = 3)$percent

## Warning in boot.ci(b, index = 3): bootstrap variances needed for studentized
## intervals

##      conf
## [1,] 0.95 25 975 0.04185257 0.04920403

l <- lm(Sales ~ Radio + TV, data = Advertising)
confint(l)

##              2.5 %      97.5 %
## (Intercept) 2.34034299 3.50185683
## Radio        0.17213877 0.20384969
## TV           0.04301292 0.04849671

```

```

library(MASS)
library(e1071)
source("class.R")
library(nnet)
fertility <- read.csv("fertility/fertility_Diagnosis.txt",
  comment.char = "#", header = FALSE)
names(fertility) <- c("Season", "Age", "ChildishDesiases",
  "Trauma", "Surgical", "Fevers", "Alcohol",
  "Smoking", "Sitting", "Diagnosis")
fertility$Season <- factor(fertility$Season,
  labels = c("winter", "spring", "summer",
  "fall"))
contrasts(fertility$Season) <- contr.sum

```

```

fertility$Age <- 18 + 18 * fertility$Age
fertility$ChildishDesiases <- factor(fertility$ChildishDesiases,
  labels = c("yes", "no"))
contrasts(fertility$ChildishDesiases) <- c(1,
  0)
fertility$Trauma <- factor(fertility$Trauma,
  labels = c("yes", "no"))
contrasts(fertility$Trauma) <- c(1, 0)
fertility$Surgical <- factor(fertility$Surgical,
  labels = c("yes", "no"))
contrasts(fertility$Surgical) <- c(1, 0)
fertility$Fevers <- factor(fertility$Fevers,
  labels = c("less3", "more3", "no"), ordered = TRUE)
fertility$Fevers <- factor(fertility$Fevers,
  levels = rev(levels(fertility$Fevers)),
  ordered = TRUE)
contrasts(fertility$Fevers) <- contr.helmert
fertility$Alcohol <- factor(fertility$Alcohol,
  labels = c("sevday", "evday", "sevweek",
    "evweek", "never"), ordered = TRUE)
fertility$Alcohol <- factor(fertility$Alcohol,
  levels = rev(levels(fertility$Alcohol)),
  ordered = TRUE)
contrasts(fertility$Alcohol) <- contr.helmert
fertility$Smoking <- factor(fertility$Smoking,
  labels = c("never", "occasional", "daily"),
  ordered = TRUE)
contrasts(fertility$Smoking) <- contr.helmert
fertility$Sitting <- fertility$Sitting *
  16
fertility$Diagnosis <- factor(fertility$Diagnosis,
  labels = c("Normal", "Altered"))
gl <- glm(Diagnosis ~ ., data = fertility,
  family = binomial)
summary(gl)

##
## Call:
## glm(formula = Diagnosis ~ ., family = binomial, data = fertility)
##
## Deviance Residuals:
##      Min       1Q   Median       3Q      Max
## -1.1621  -0.4872  -0.2585  -0.1271   2.8269
##
## Coefficients:
##                               Estimate Std. Error z value Pr(>|z|)
## (Intercept)           -20.9269    678.7346  -0.031   0.9754
## Season1                -1.4493     1.0583  -1.370   0.1708

```

```

## Season2          -0.3051    0.6931   -0.440    0.6598
## Season3          0.9589    1.2124    0.791    0.4290
## Age              0.3677    0.2162   1.701    0.0890 .
## ChildishDesiases1 -0.5696    1.0441   -0.546    0.5854
## Trauma1          1.6410    0.8572   1.914    0.0556 .
## Surgical1        -0.3057    0.7979   -0.383    0.7017
## Fevers1          0.2473    0.5347   0.462    0.6438
## Fevers2          0.5369    0.3723   1.442    0.1493
## Alcohol1          0.9215    0.5298   1.740    0.0819 .
## Alcohol2          0.4349    0.3150   1.381    0.1674
## Alcohol3          -3.3031    599.8863  -0.006    0.9956
## Alcohol4          -2.4536    494.6789  -0.005    0.9960
## Smoking1          -0.1733    0.5031   -0.345    0.7304
## Smoking2          0.1632    0.3036   0.537    0.5910
## Sitting           0.2050    0.1511   1.357    0.1747
## ---
## Signif. codes:
## 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
## Null deviance: 73.385 on 99 degrees of freedom
## Residual deviance: 55.134 on 83 degrees of freedom
## AIC: 89.134
##
## Number of Fisher Scoring iterations: 15

gl <- glm(Diagnosis ~ Trauma + Age + Alcohol,
            data = fertility, family = binomial)
summary(gl)

##
## Call:
## glm(formula = Diagnosis ~ Trauma + Age + Alcohol, family = binomial,
##      data = fertility)
##
## Deviance Residuals:
##      Min       1Q     Median       3Q      Max
## -1.2511  -0.5020  -0.3513  -0.2706  2.4948
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept) -14.7395   678.7121  -0.022  0.9827
## Trauma1      1.3935    0.7439   1.873  0.0610 .
## Age          0.2066    0.1590   1.300  0.1937
## Alcohol1     0.8010    0.4381   1.828  0.0675 .
## Alcohol2     0.2621    0.2646   0.991  0.3218
## Alcohol3    -3.2959    599.8862  -0.005  0.9956

```

```

## Alcohol4      -2.2602   494.6788  -0.005   0.9964
## ---
## Signif. codes:
## 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
## Null deviance: 73.385 on 99 degrees of freedom
## Residual deviance: 63.914 on 93 degrees of freedom
## AIC: 77.914
##
## Number of Fisher Scoring iterations: 15

gl.aic <- stepAIC(gl)

## Start:  AIC=77.91
## Diagnosis ~ Trauma + Age + Alcohol
##
##          Df Deviance    AIC
## - Alcohol  4   69.022 75.022
## - Age      1   65.616 77.616
## <none>     63.914 77.914
## - Trauma   1   67.938 79.938
##
## Step:  AIC=75.02
## Diagnosis ~ Trauma + Age
##
##          Df Deviance    AIC
## <none>     69.022 75.022
## - Age      1   71.280 75.280
## - Trauma   1   72.118 76.118

summary(gl.aic)

##
## Call:
## glm(formula = Diagnosis ~ Trauma + Age, family = binomial, data = fertility)
##
## Deviance Residuals:
##      Min       1Q     Median      3Q      Max
## -1.0360  -0.5396  -0.4415  -0.3155   2.6531
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept) -9.3009    4.5276  -2.054   0.0399 *
## Trauma1      1.2102    0.7336   1.650   0.0990 .
## Age          0.2152    0.1422   1.513   0.1302
## ---
## Signif. codes:

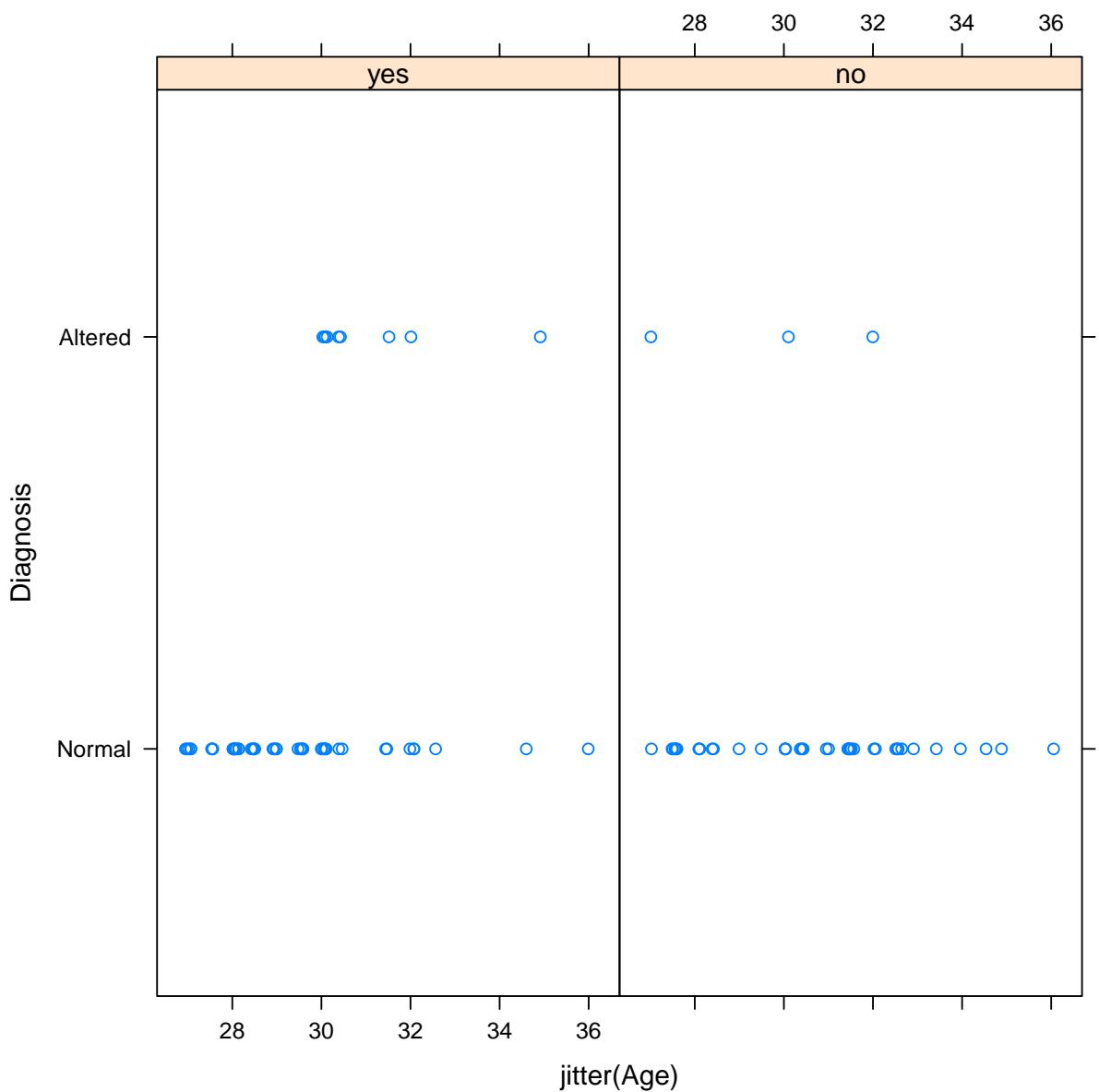
```

```

## 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
## Null deviance: 73.385 on 99 degrees of freedom
## Residual deviance: 69.022 on 97 degrees of freedom
## AIC: 75.022
##
## Number of Fisher Scoring iterations: 5

gl <- glm(Diagnosis ~ Age * Trauma, data = fertility,
            family = binomial)
xyplot(Diagnosis ~ jitter(Age) | Trauma,
       data = fertility, auto.key = list(corner = c(0,
                                                 1)))

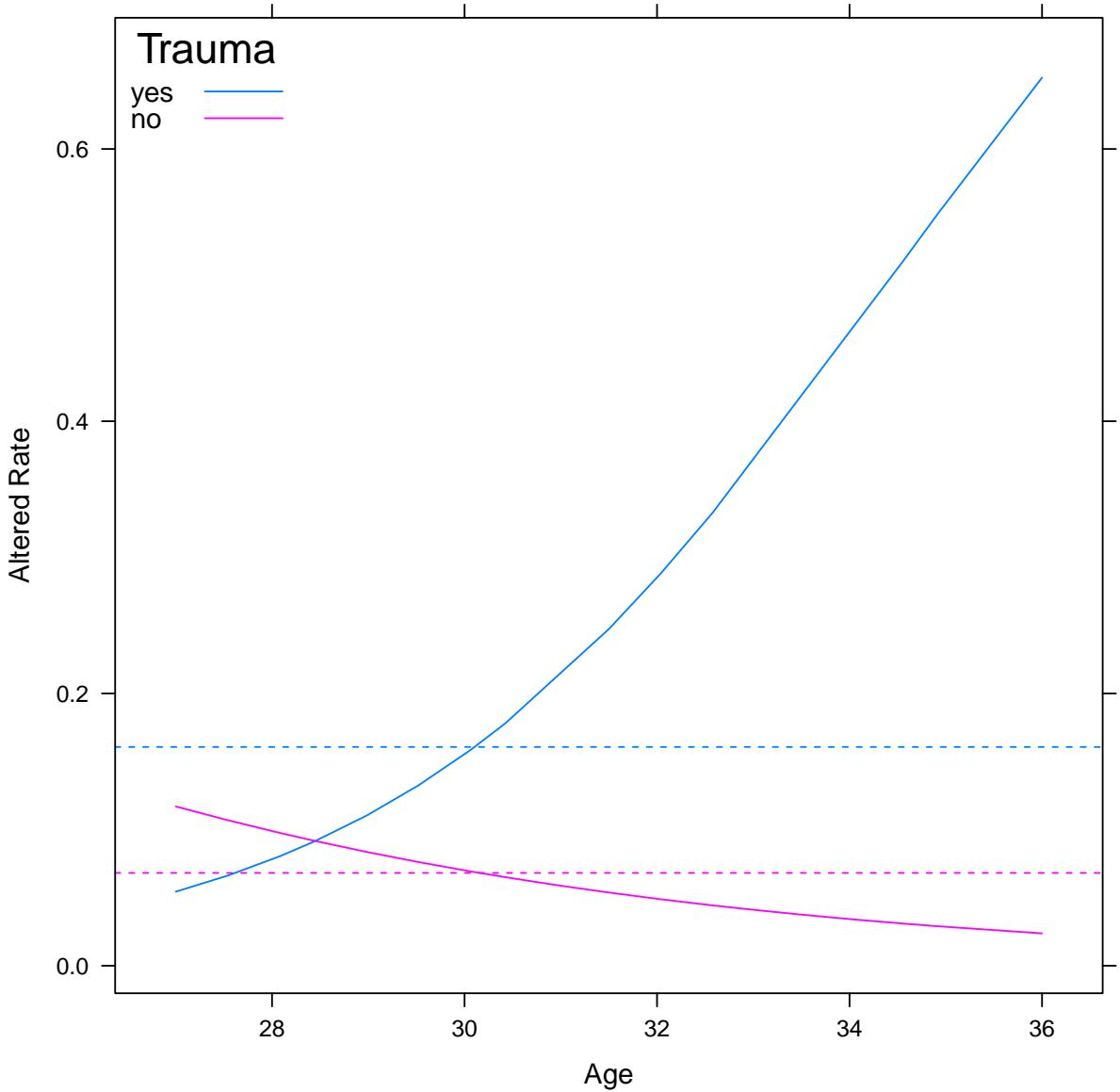
```



```

gl <- glm(Diagnosis ~ Age * Trauma, data = fertility,
           family = binomial(link = "logit"))
fertility.sorted <- fertility[order(fertility$Age),
]
xyplot(predict(gl, fertility.sorted, type = "response") ~
    Age, groups = Trauma, data = fertility.sorted,
    type = "l", auto.key = list(corner = c(0,
        1), title = "Trauma", lines = TRUE,
        points = FALSE), xlab = "Age", ylab = "Altered Rate") +
layer_(panel.superpose(x = fertility$Diagnosis ==
    "Altered", panel.groups = function(x,
        ...) panel.abline(h = mean(x),
        ...), lty = "dashed", groups = fertility$Trauma,
        ...))

```



```

gl <- glm(Diagnosis ~ Sitting, data = fertility,
           family = binomial)
summary(gl)

##
## Call:
## glm(formula = Diagnosis ~ Sitting, family = binomial, data = fertility)
##
## Deviance Residuals:
##      Min       1Q   Median       3Q      Max
## -0.5602  -0.5135  -0.4967  -0.4902   2.1151
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept) -2.14628    0.74487  -2.881  0.00396 **
## Sitting      0.02335    0.10174   0.230  0.81844
## ---
## Signif. codes:
## 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
## Null deviance: 73.385 on 99 degrees of freedom
## Residual deviance: 73.333 on 98 degrees of freedom
## AIC: 77.333
##
## Number of Fisher Scoring iterations: 4

ld <- lda(Diagnosis ~ Age * Trauma, data = fertility)
table(predicted = predict(ld, fertility)$class,
      actual = fertility$Diagnosis)

## Warning in predict.lda(ld, fertility): variable names in 'newdata' do not match
## those in 'object'

##          actual
## predicted Normal Altered
##   Normal     82      12
##   Altered     6       0

qd <- qda(Diagnosis ~ Age * Trauma, data = fertility)
table(predicted = predict(qd, fertility)$class,
      actual = fertility$Diagnosis)

## Warning in predict.qda(qd, fertility): variable names in 'newdata' do not match
## those in 'object'

##          actual
## predicted Normal Altered
##   Normal     76      11
##   Altered     12      1

```

```

ld <- lda(Diagnosis ~ Age + Trauma, data = fertility)
table(predicted = predict(ld, fertility)$class,
      actual = fertility$Diagnosis)

## Warning in predict.lda(ld, fertility): variable names in 'newdata' do not match
## those in 'object'

##           actual
## predicted Normal Altered
##   Normal     88     12
##   Altered      0      0

tune(glm, data = fertility, Diagnosis ~ Age *
      Trauma, family = binomial, tunecontrol = tune.control(sampling = "boot"),
      predict.func = my.predict.glm)

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

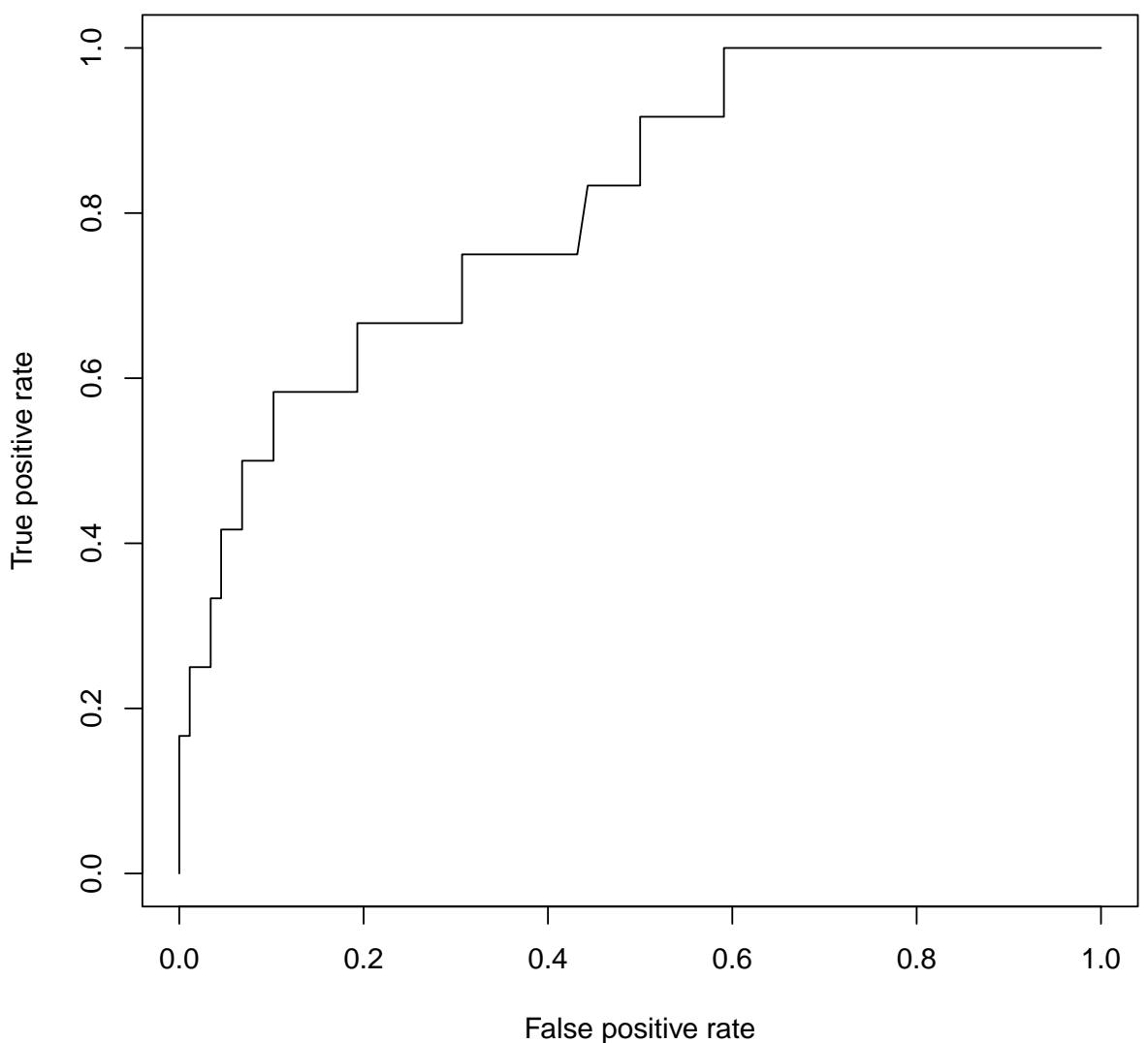
##
## Error estimation of 'glm' using bootstrapping: 0.2045734

nb <- naiveBayes(Diagnosis ~ ., data = fertility)
table(predicted = predict(nb, fertility),
      actual = fertility$Diagnosis)

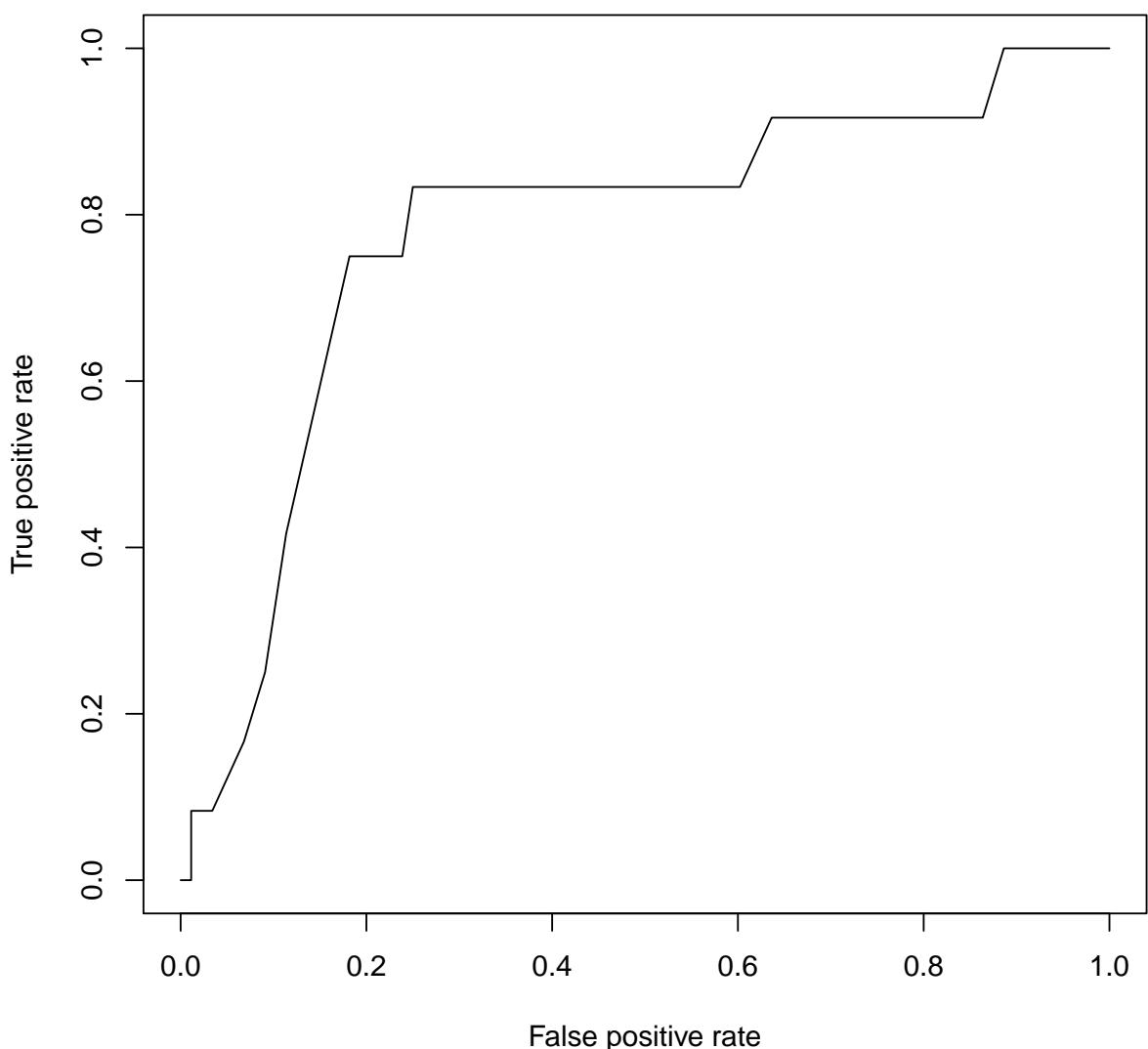
##           actual
## predicted Normal Altered
##   Normal     88     11
##   Altered      0      1

pred.nb <- predict(nb, fertility, type = "raw")[, 2]
pred.glm <- predict(glm(Diagnosis ~ Age *
      Trauma, data = fertility, family = binomial))
plot(ROC(pred.nb, fertility$Diagnosis))

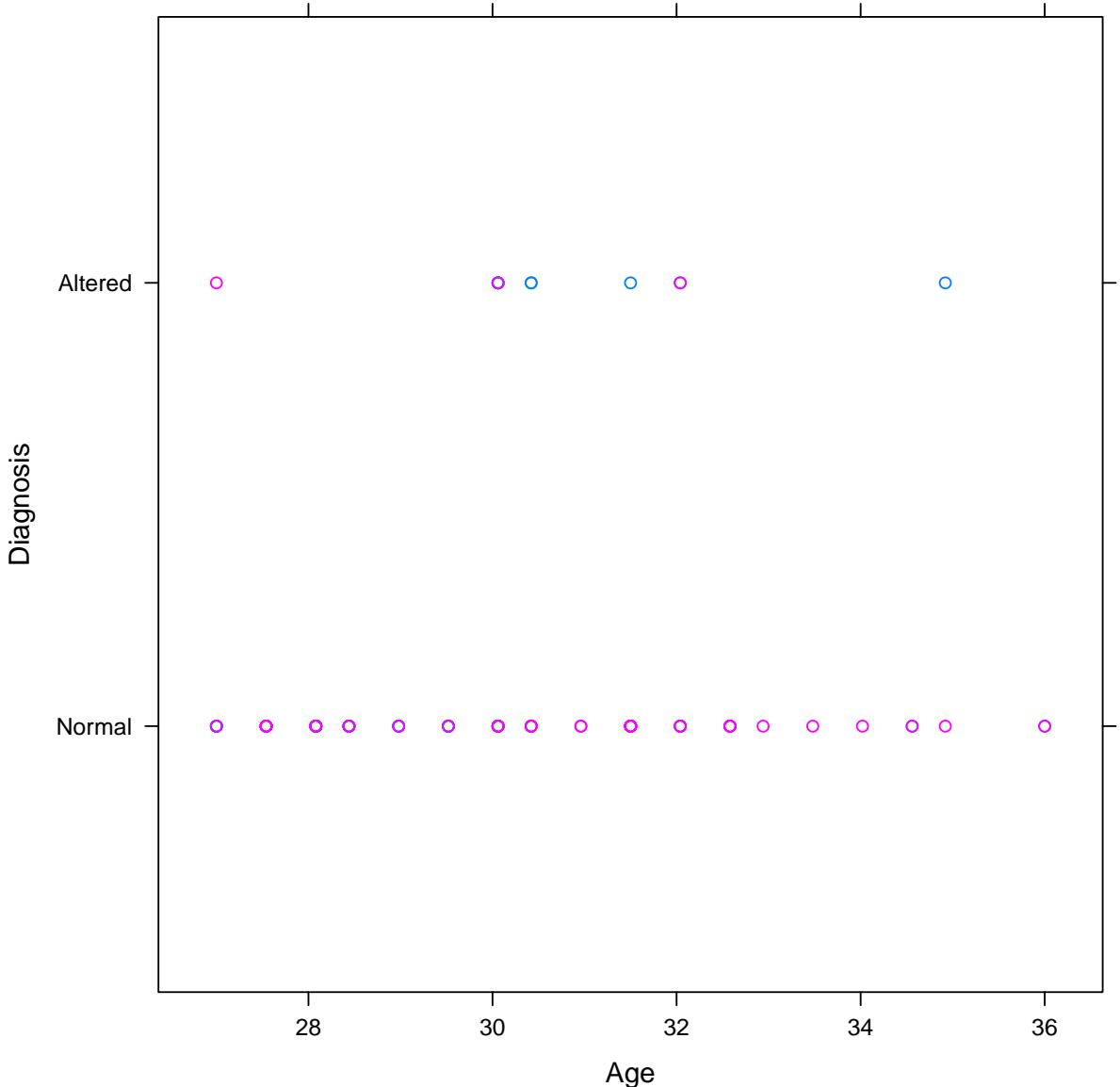
```



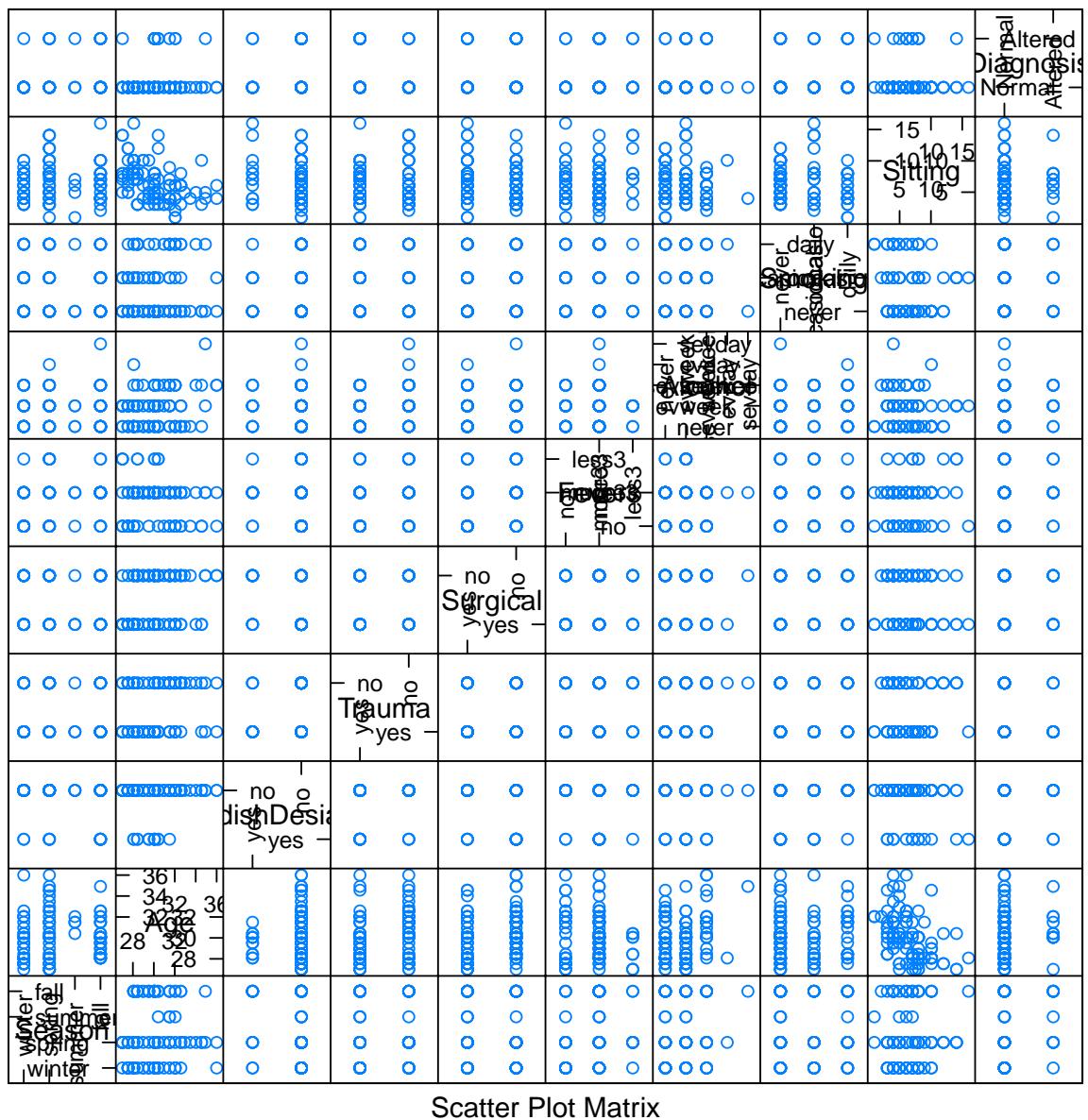
```
plot(ROC(pred.glm, fertility$Diagnosis))
```



```
xyplot(Diagnosis ~ Age, groups = Trauma,  
       data = fertility)
```



```
gl <- glm(Diagnosis ~ Age + Trauma + Alcohol +
  Sitting + Season, data = fertility, family = binomial)
library(lattice)
splom(fertility)
```



Scatter Plot Matrix

7 Рисование

Вообще в R существует по крайней мере три “школы” рисования. Во-первых, это классический пакет `graphics`. Пользоваться им не советую, во-первых, потому что в результате обычно получаются картинки посредственного качества, а во-вторых из-за его крайне примитивной идеологии: график рассматривается как холст, на котором можно что-то рисовать и подрисовывать, но нельзя ничего исправить.

Современный объектный подход к рисованию реализуют пакеты `lattice` и `ggplot2`. В них функции рисования возвращают некий объект, который можно модифицировать, хранить, передавать и, конечно же, отрисовывать на конкретном устройстве.

Ниже я буду все графики делать в `lattice` из личных предпочтений. Некоторые находят `ggplot2` более эффективным и современным, но я привык к `lattice`.

Собственно, нам понадобится пакет `lattice` (он уже установлен, надо только подключить), а также будет полезным пакет `latticeExtra` (как видно из названия, он

расширяет возможности `lattice`; его нужно поставить с CRAN).

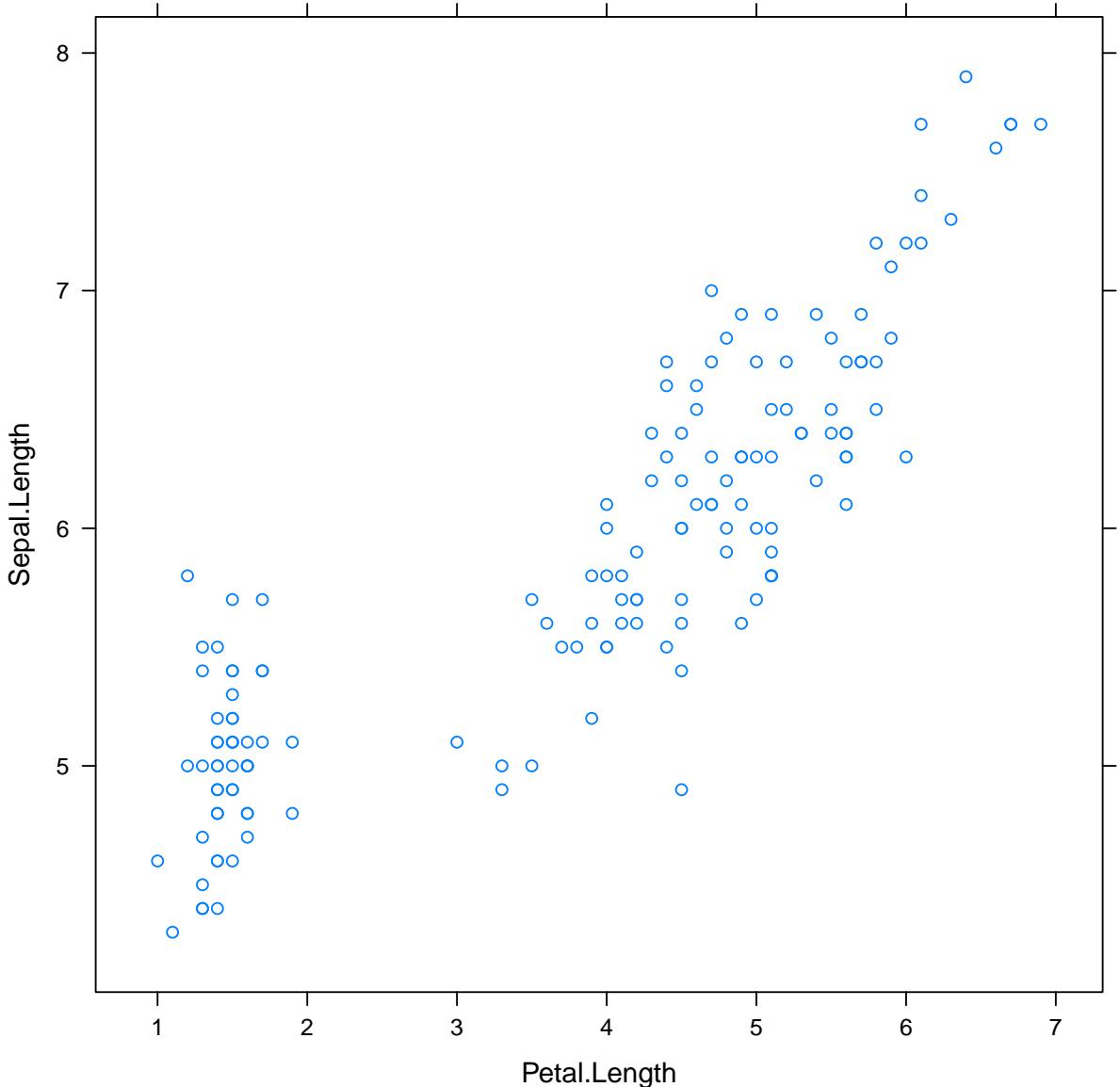
Все функции рисования в `lattice` принимают следующие параметры:

1. `x` — формула; собственно, зависимость, которую мы хотим изобразить
2. `data` — данные, относительно которых будет вычисляться формула (просто датафрейм или именованный список, содержащий использованные в формуле переменные)
3. `groups` — параметр, позволяющий нарисовать несколько наложенных линий на одном графике
4. `panel` — панельная функция, как именно рисовать каждый отдельный график (панель)
5. `...` — параметры, передаваемые в панельную функцию, а также всякие дополнительные параметры, вроде расположения и общего количества панелей.

На примере данных `iris`, посмотрим, какие бывают графики:

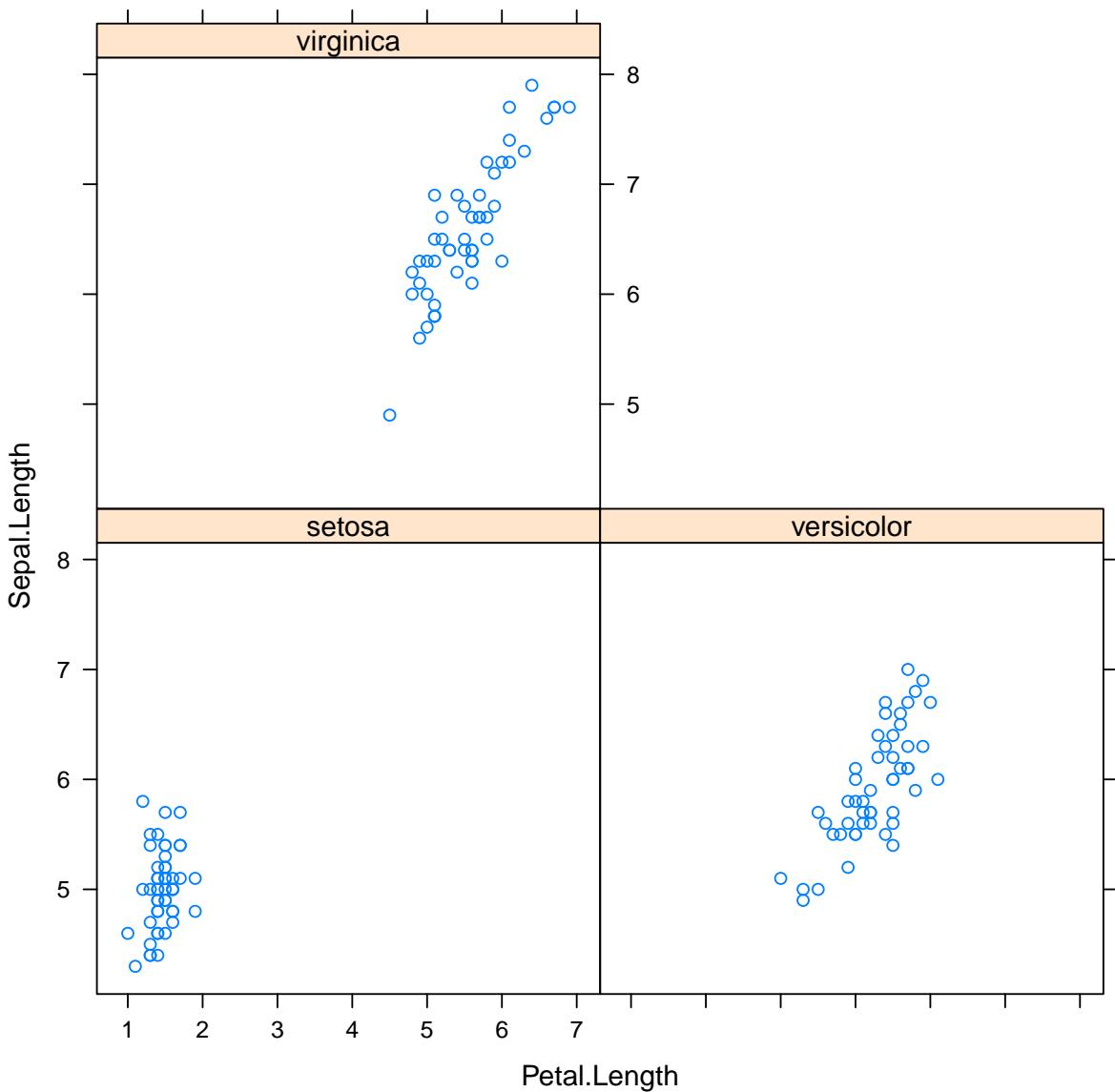
`xyplot()` — скаттерплот. Каждая строка датафрейма изображается отдельной точкой в координатах двух выбранных столбцов.

```
library(lattice)
xyplot(Sepal.Length ~ Petal.Length, data = iris)
```



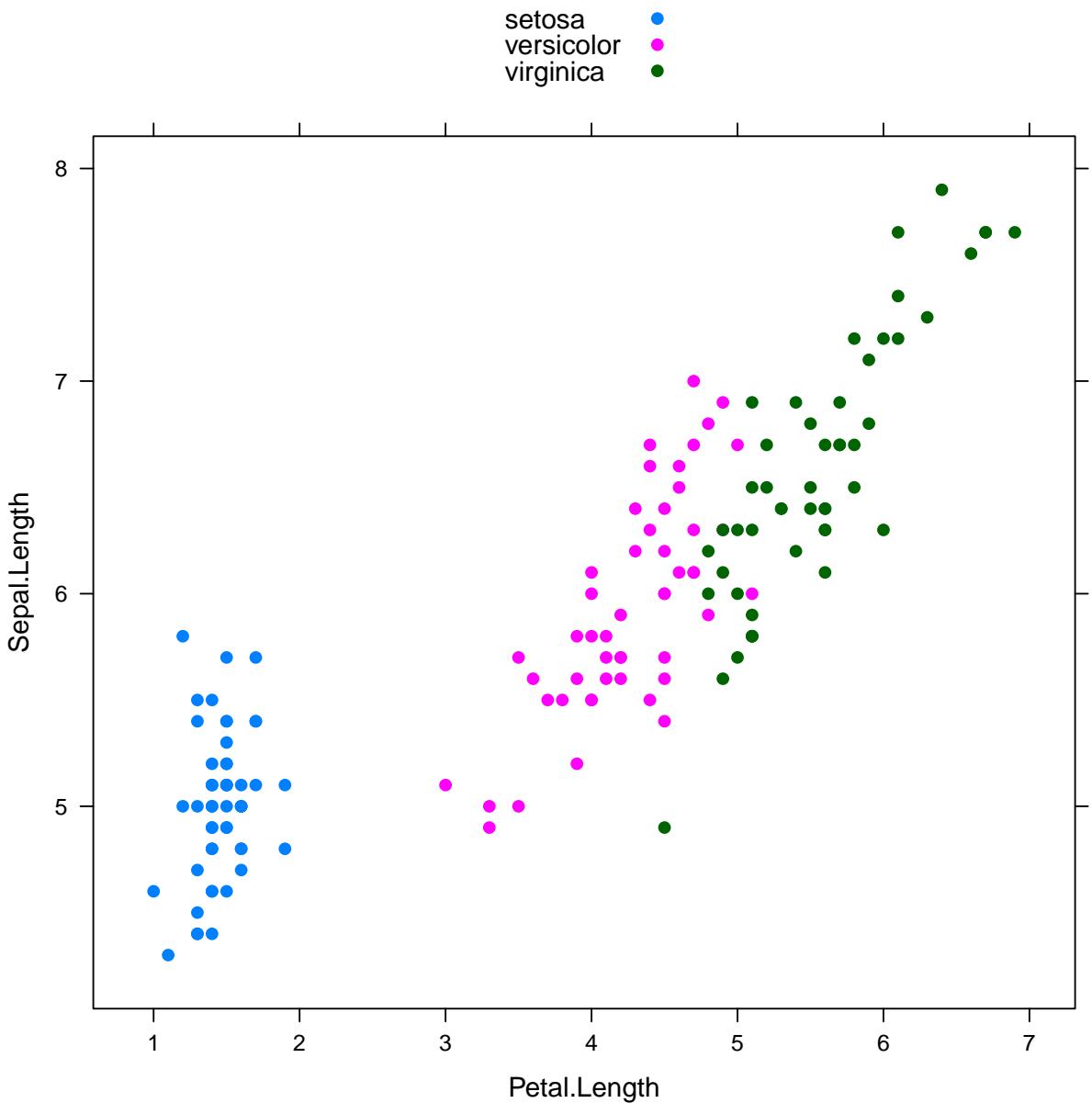
Не очень красиво и малоинформативно. Давайте нарисуем сорта на отдельных графиках:

```
xyplot(Sepal.Length ~ Petal.Length | Species,  
       data = iris)
```



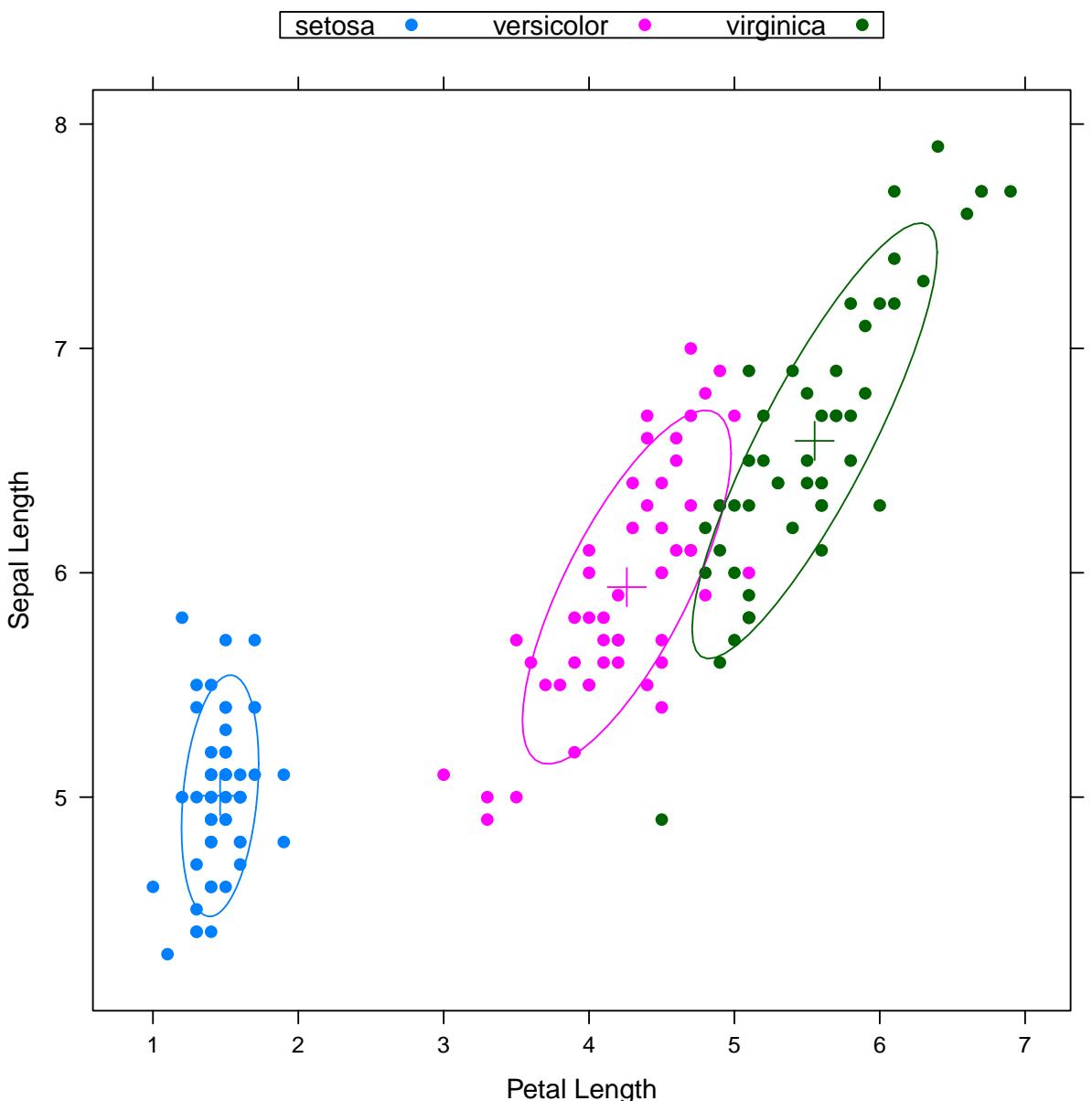
А может лучше все-таки на одном, но разными цветами?.. А еще я хочу сделать точки сплошным и добавить легенду:

```
xyplot(Sepal.Length ~ Petal.Length, groups = Species,
       data = iris, par.settings = simpleTheme(pch = 19),
       auto.key = TRUE)
```



```
library(latticeExtra)
```

```
xyplot(Sepal.Length ~ Petal.Length, groups = Species,
       data = iris, par.settings = simpleTheme(pch = 19),
       auto.key = list(columns = 3, border = TRUE),
       panel = function(...) {
         panel.xyplot(...)
         panel.ellipse(...)
       }, xlab = "Petal Length", ylab = "Sepal Length")
```



Напоследок приведу пример написания своей панельной функции для `xypplot()`:

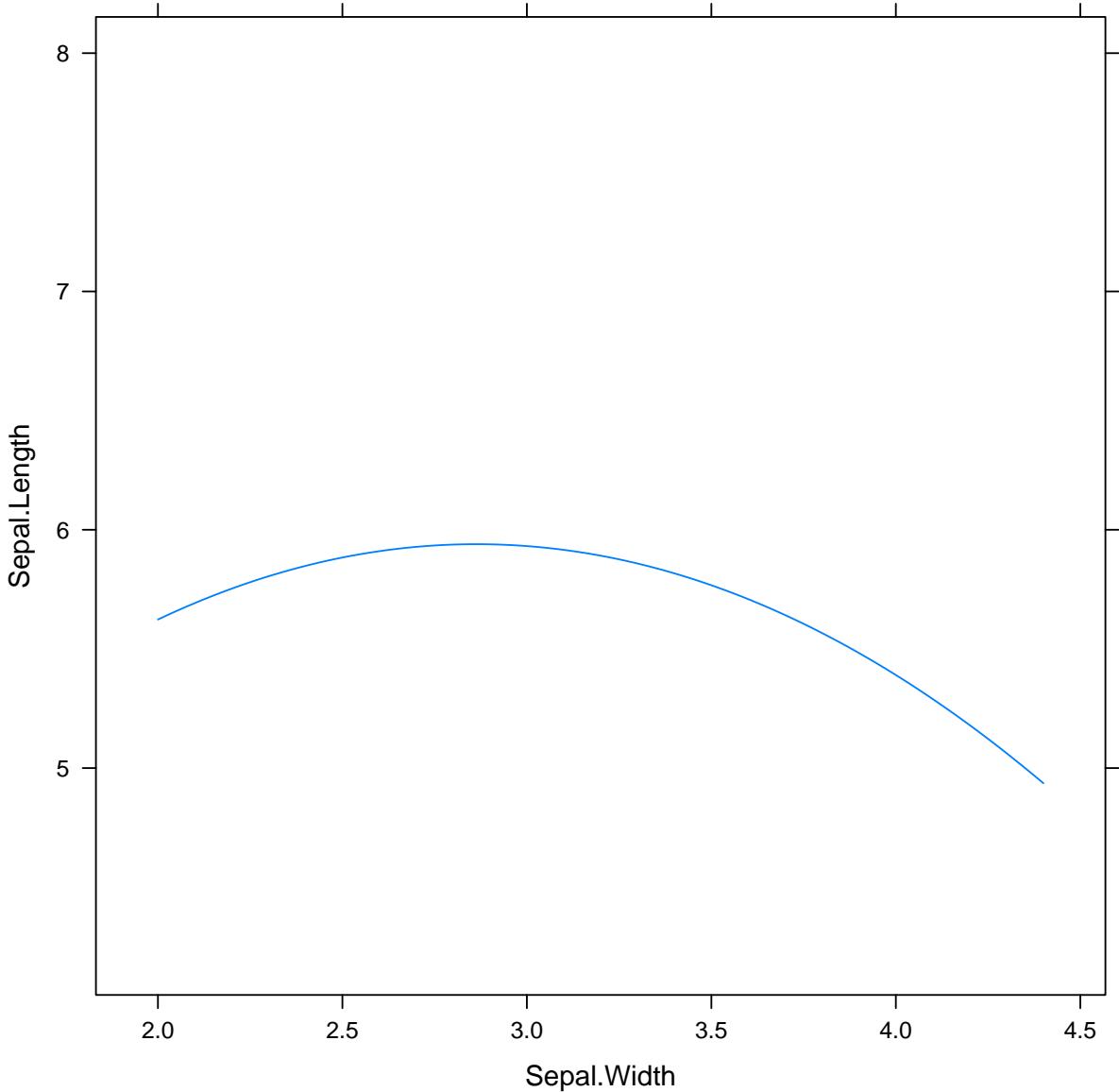
```
read_chunk("panel.lmpolyline.R")
```

```
panel.lmpolyline <- function(x, y, groups = NULL,
  degree = 1, col.line = par.line$col,
  lty = par.line$lty, lwd = par.line$lwd,
  alpha = par.line$alpha, ..., identifier = "lmpolyline") {
  x <- as.numeric(x)
  y <- as.numeric(y)
  if (!is.null(groups)) {
    par.line <- trellis.par.get("superpose.line")
    panel.superpose(x = x, y = y, groups = groups,
      degree = degree, col.line = col.line,
      lty = lty, lwd = lwd, alpha = alpha,
```

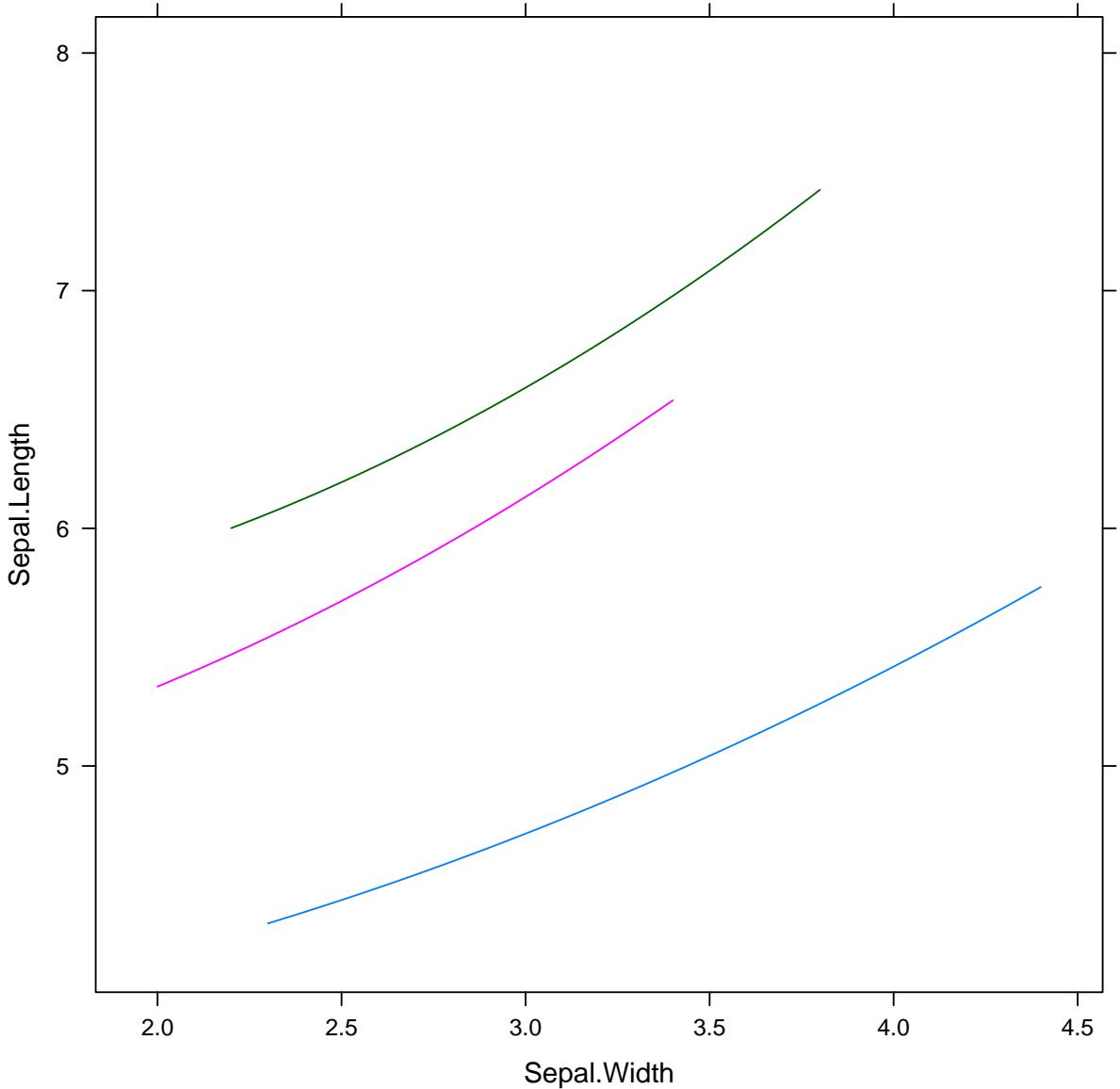
```
    panel.groups = sys.function(),
    ...)

} else {
  if (length(x) > degree) {
    l <- lm(y ~ poly(x, degree = degree))
    par.line <- trellis.par.get("plot.line")
    panel.curve(predict(l, list(x = x)),
                from = min(x), to = max(x),
                col.line = col.line, lty = lty,
                lwd = lwd, alpha = alpha,
                ..., identifier = identifier)
  }
}

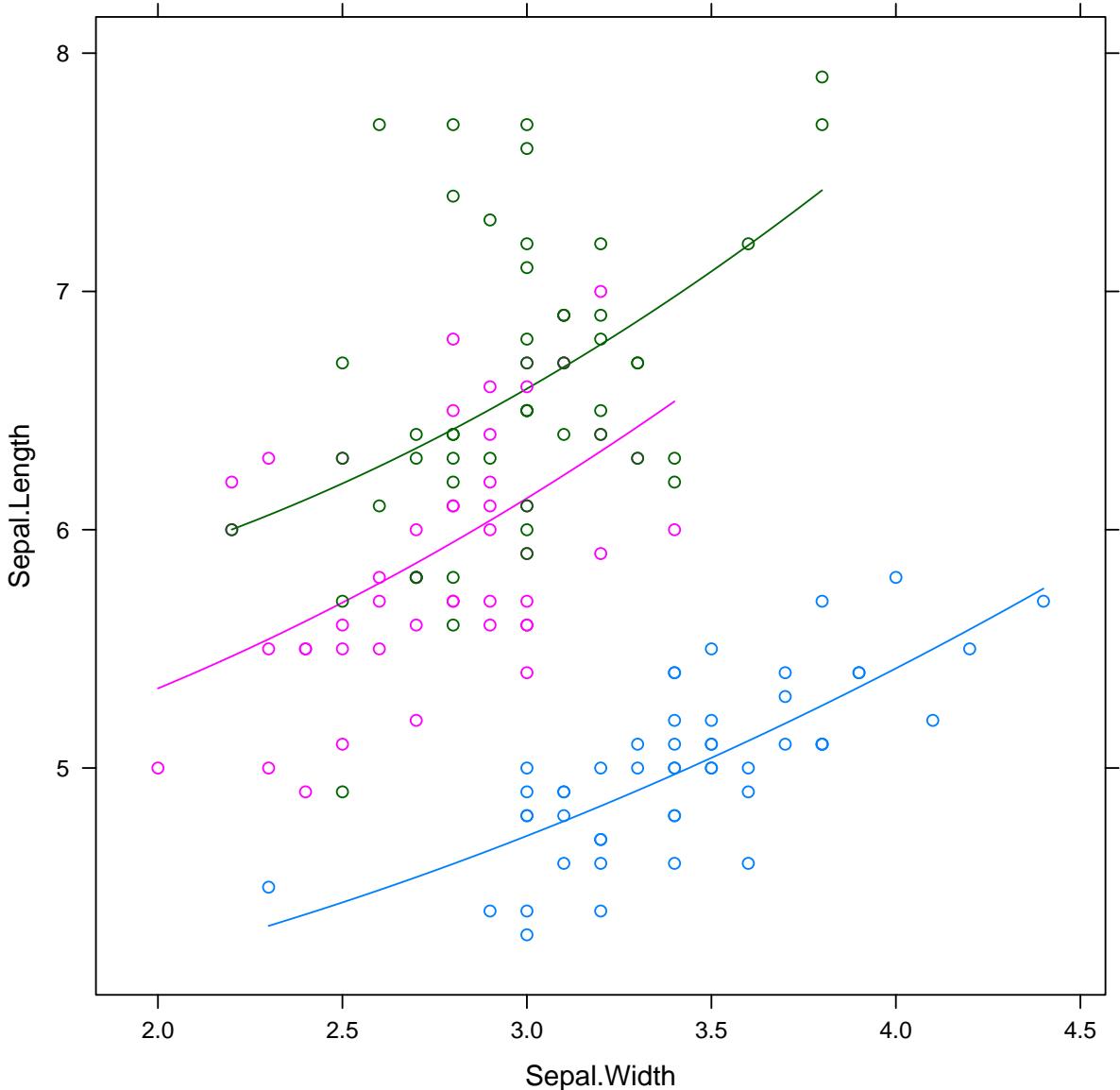
xyplot(Sepal.Length ~ Sepal.Width, data = iris,
       panel = panel.lmpolyline, degree = 2)
```



```
xyplot(Sepal.Length ~ Sepal.Width, groups = Species,  
       data = iris, panel = panel.lmpolyline,  
       degree = 2)
```



```
xyplot(Sepal.Length ~ Sepal.Width, groups = Species,  
       data = iris) + layer_(panel.lmpolyline(...,  
degree = 2))
```



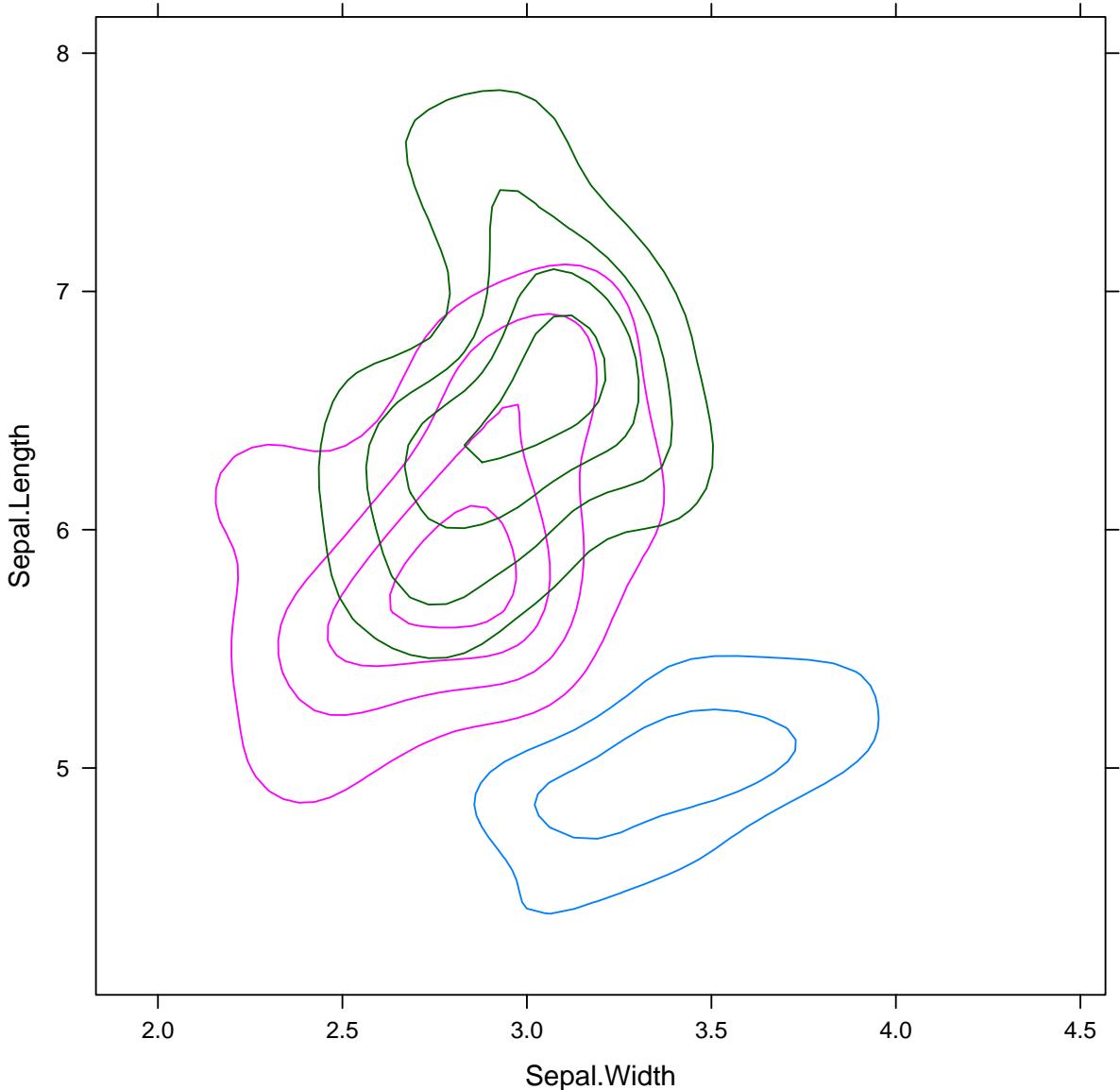
7.1 Оценка двумерной плотности (kde2)

```
panel.kde2d <- function(x, y, groups = NULL,
  subscripts, n = 100, cuts = 5, col.line = par.line$col,
  lty = par.line$lty, lwd = par.line$lwd,
  alpha = par.line$alpha, ..., identifier = "kde2d",
  col) {
  require("MASS")
  x <- as.numeric(x)
  y <- as.numeric(y)
  if (!is.null(groups)) {
    par.line <- trellis.par.get("superpose.line")
    panel.superpose(x = x, y = y, groups = groups,
      subscripts = subscripts, n = n,
```

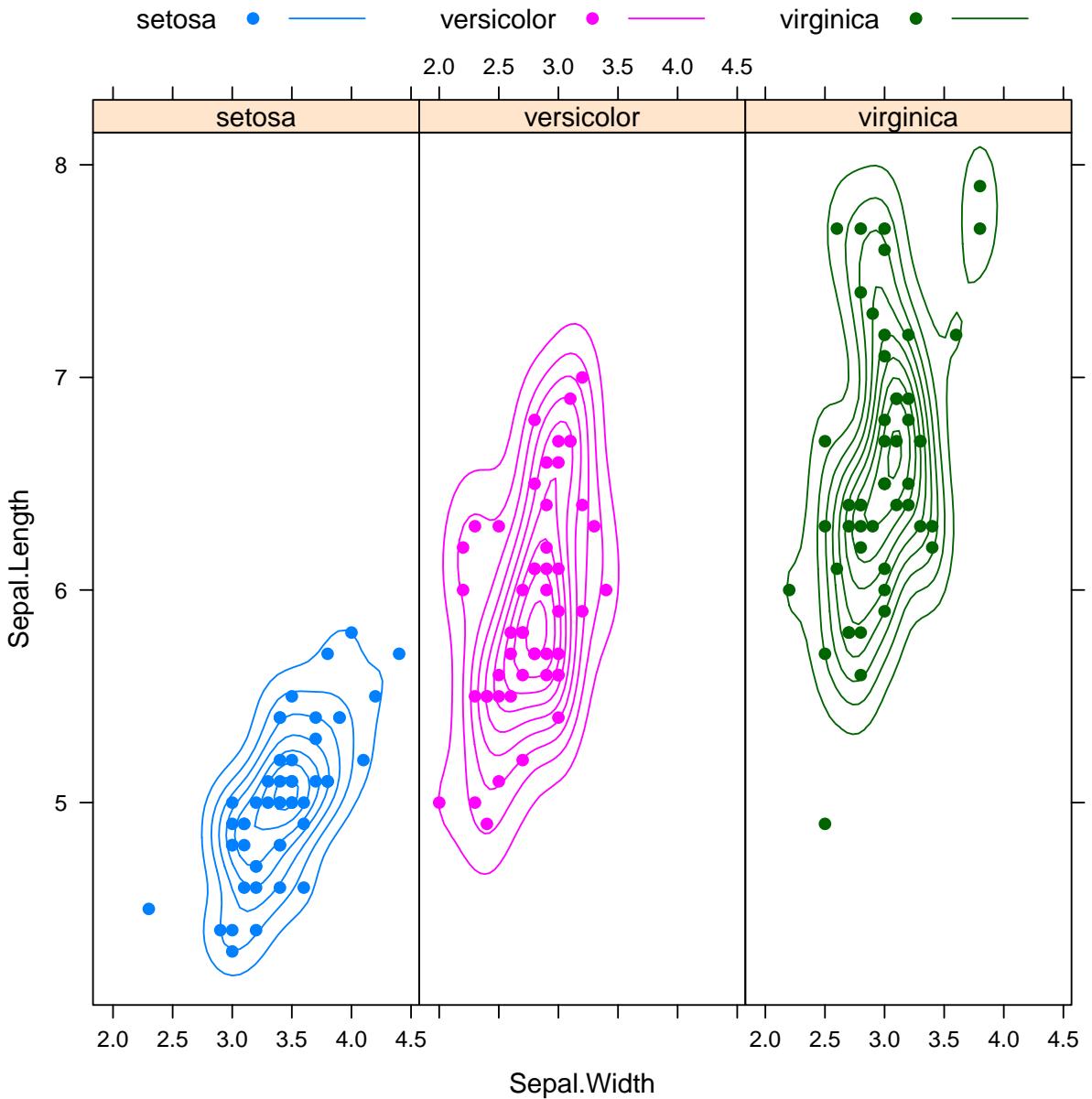
```

        cuts = cuts, panel.groups = sys.function(),
        col.line = col.line, lty = lty,
        lwd = lwd, alpha = alpha, ...)
} else {
  drange <- function(x) {
    r <- range(x)
    d <- diff(r)
    r + c(-d, d)
  }
  kde <- kde2d(x, y, n = n, lims = c(drange(x),
                                         drange(y)))
  data <- expand.grid(x = kde$x, y = kde$y)
  data$z <- as.vector(kde$z)
  plot.line <- trellis.par.get("plot.line")
  panel.contourplot(data$x, data$y,
                     data$z, at = pretty(data$z, n = cuts),
                     subscripts = seq_along(data$x),
                     contour = TRUE, region = FALSE,
                     col = col.line, lty = lty, lwd = lwd,
                     alpha = alpha, ..., identifier = identifier)
}
}
xyplot(Sepal.Length ~ Sepal.Width, data = iris,
       groups = Species, panel = panel.kde2d)

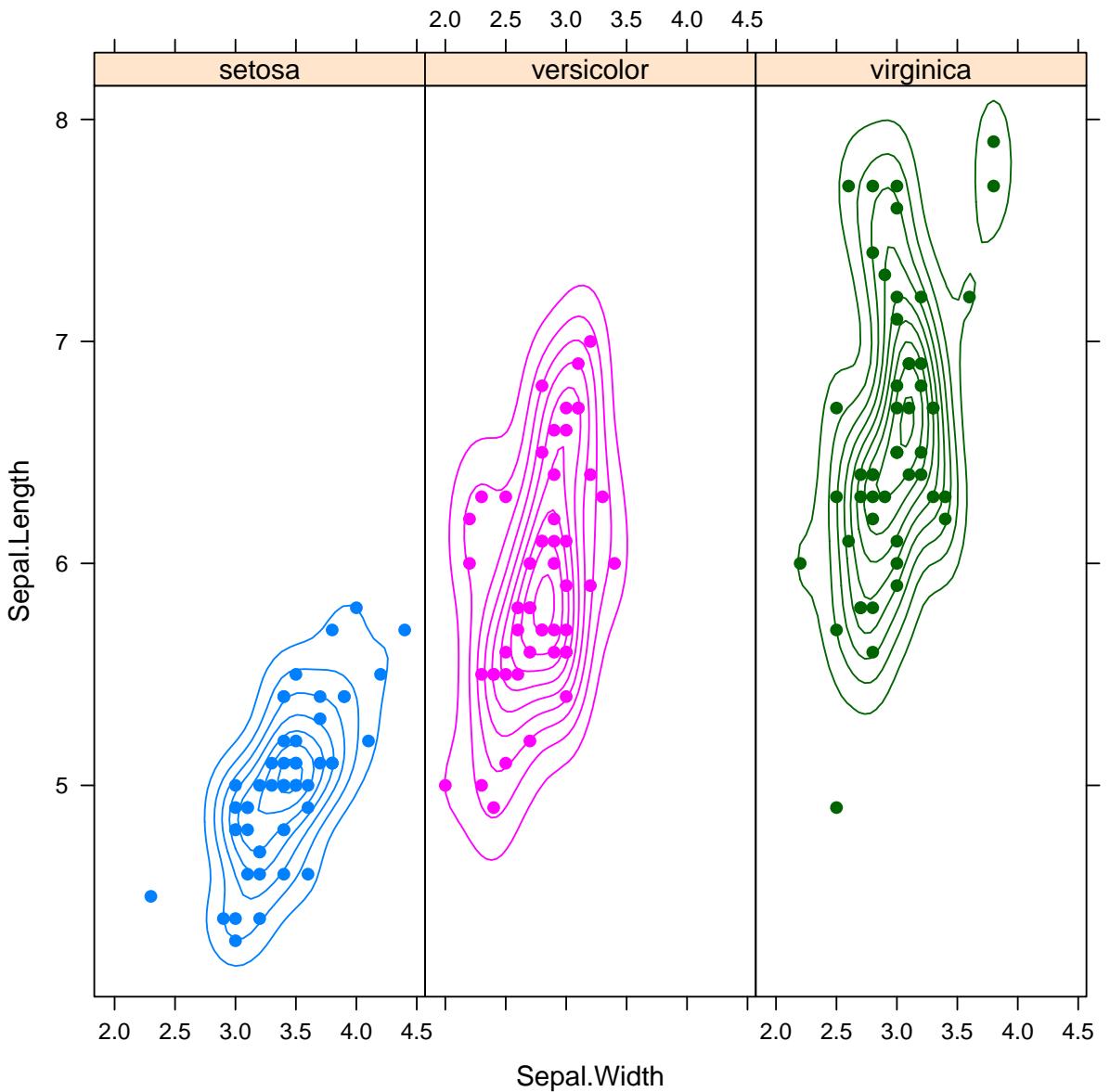
```



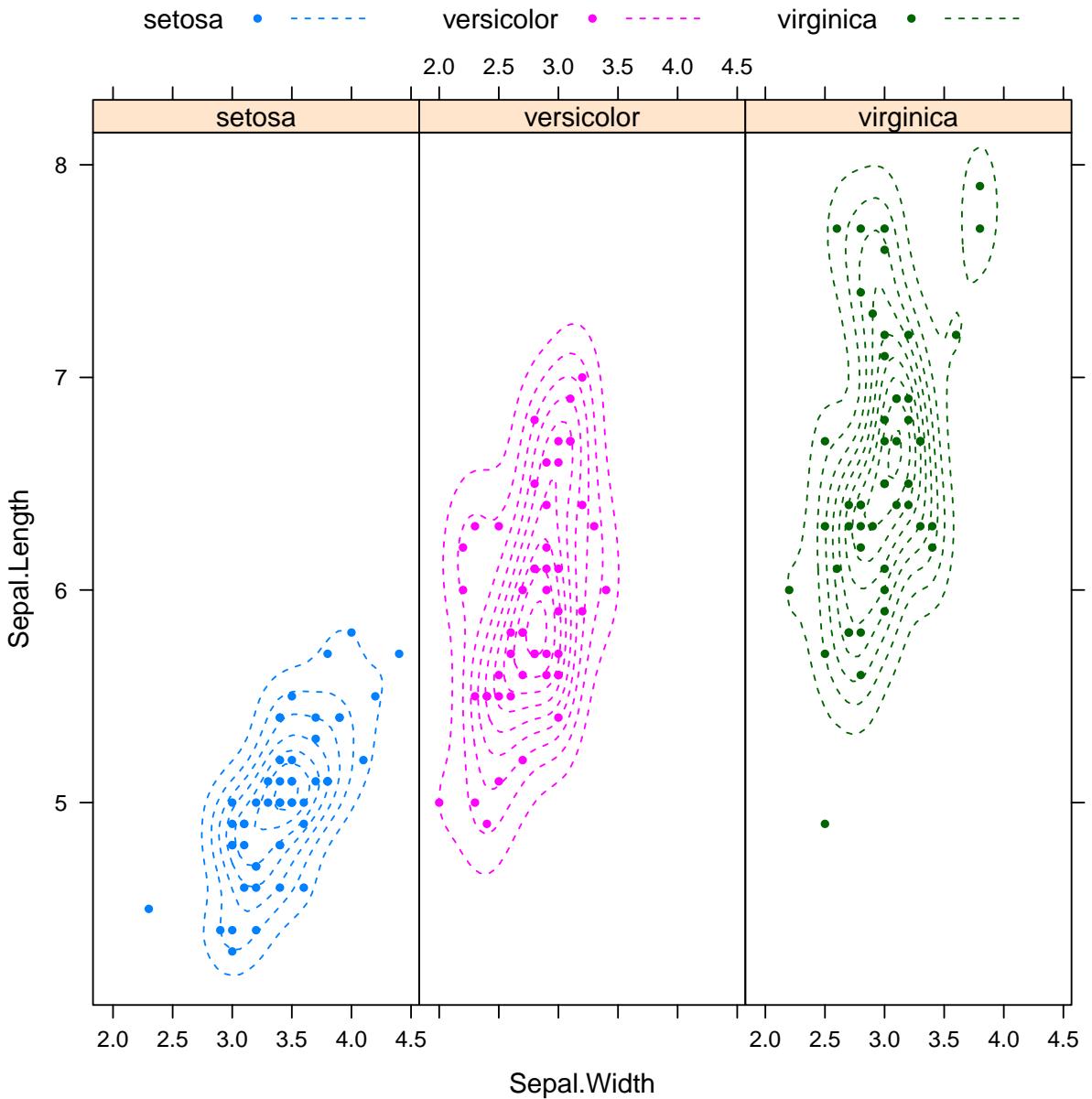
```
xyplot(Sepal.Length ~ Sepal.Width | Species,
       data = iris, groups = Species, par.settings = simpleTheme(pch = 19),
       auto.key = list(columns = 3, lines = TRUE),
       layout = c(3, 1)) + layer_(panel.kde2d(...,
       cuts = 10))
```



```
xyplot(Sepal.Length ~ Sepal.Width | Species,
       data = iris, groups = Species, par.settings = simpleTheme(pch = 19),
       layout = c(3, 1)) + layer(panel.kde2d(...,
       cuts = 10))
```

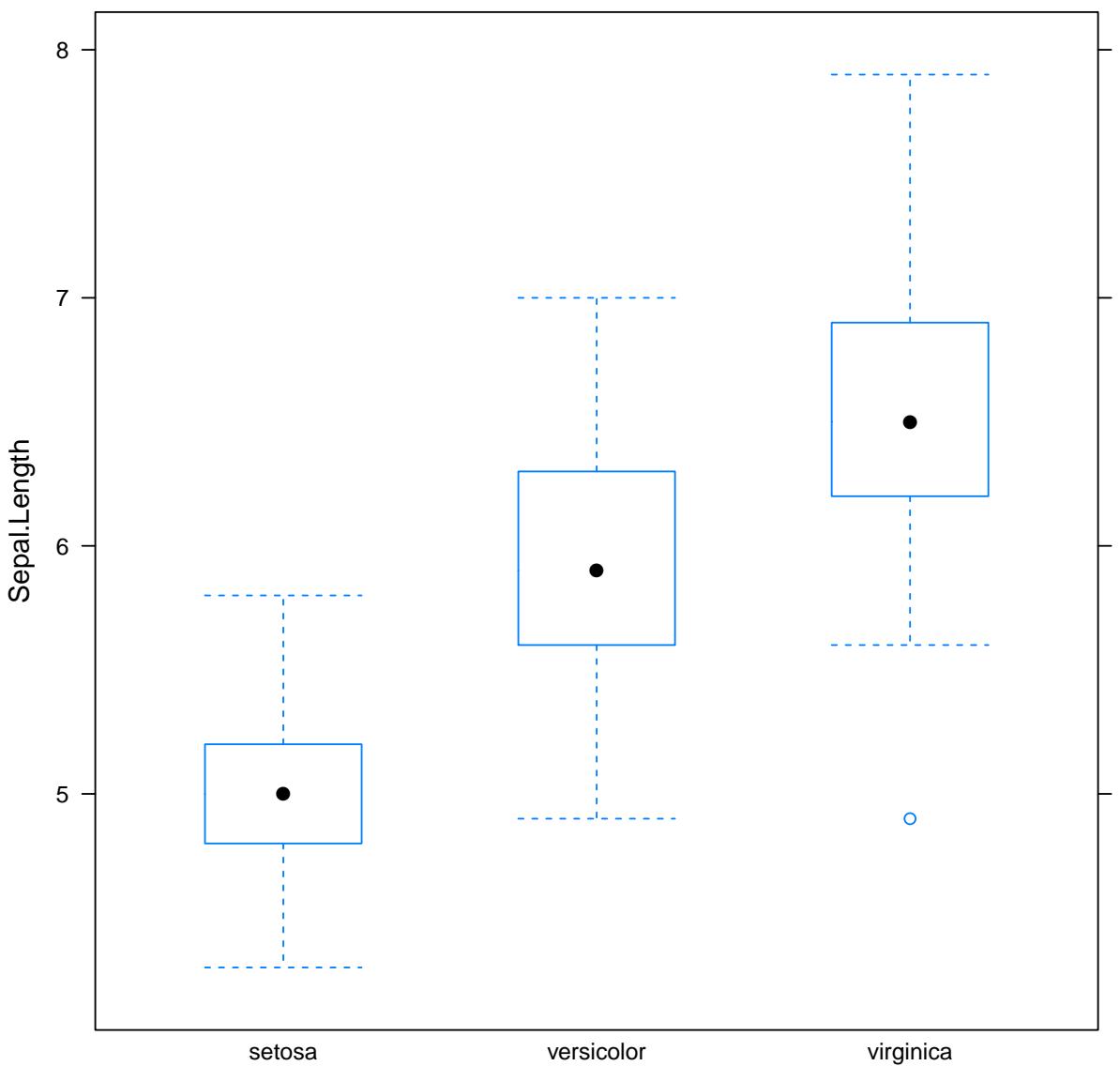


```
xyplot(Sepal.Length ~ Sepal.Width | Species,
       data = iris, groups = Species, par.settings = simpleTheme(pch = 19,
       cex = 0.5, lwd = 1, lty = "dashed"),
       auto.key = list(columns = 3, lines = TRUE),
       layout = c(3, 1)) + layer_(panel.kde2d(...,
       cuts = 10))
```

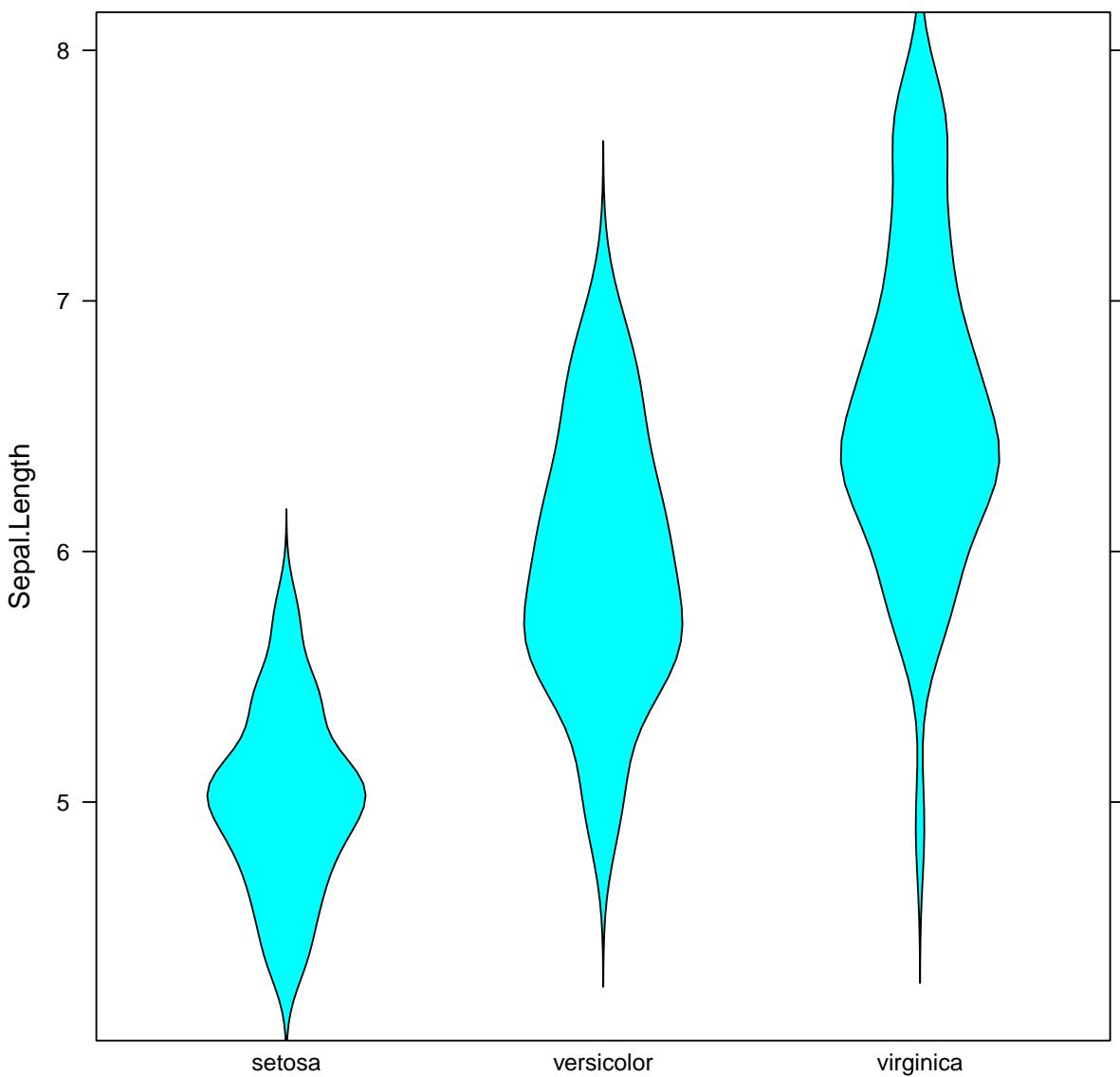


7.2 Boxplots, stripplots and dotplots (draft)

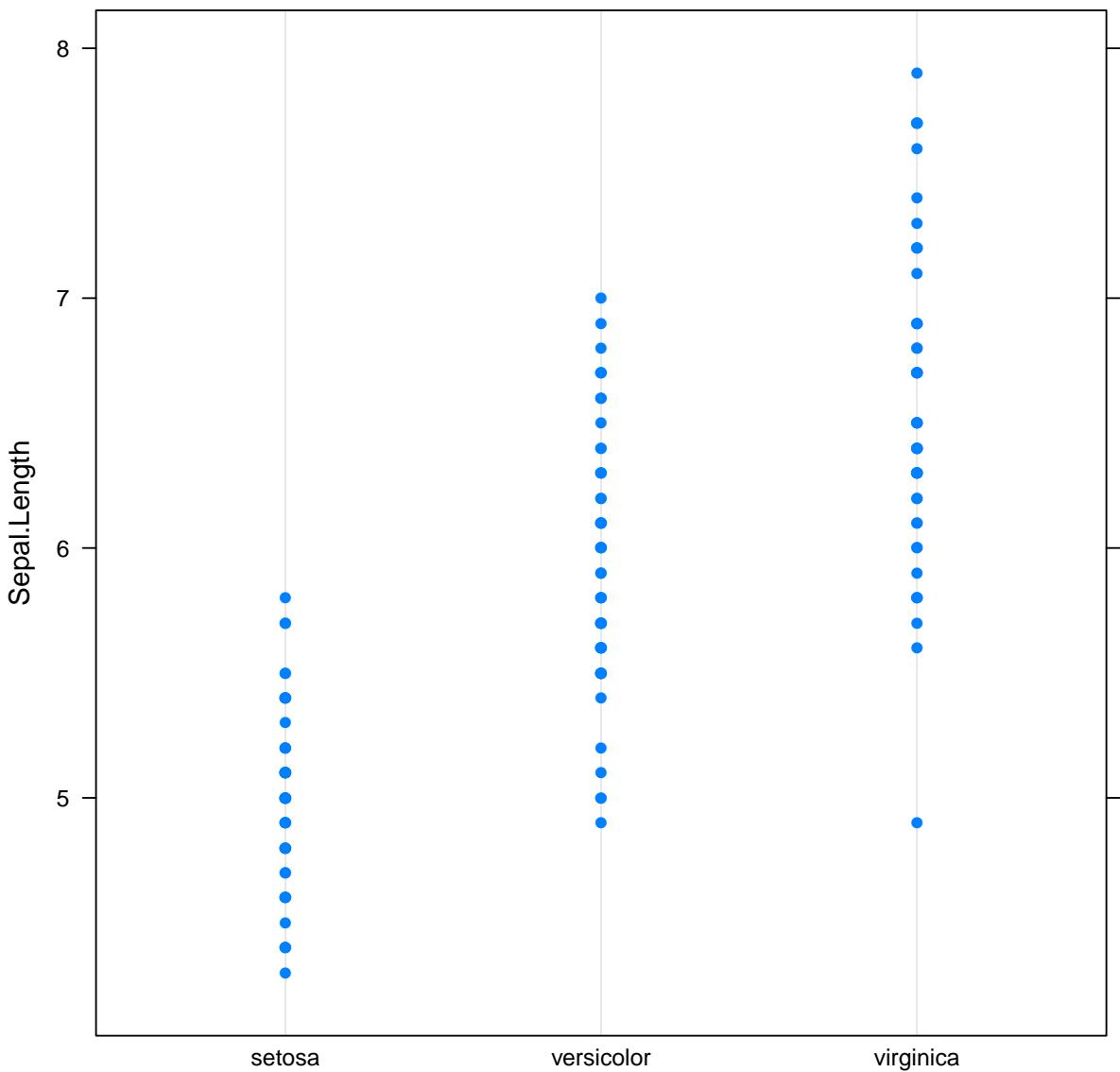
```
bwplot(Sepal.Length ~ Species, data = iris)
```



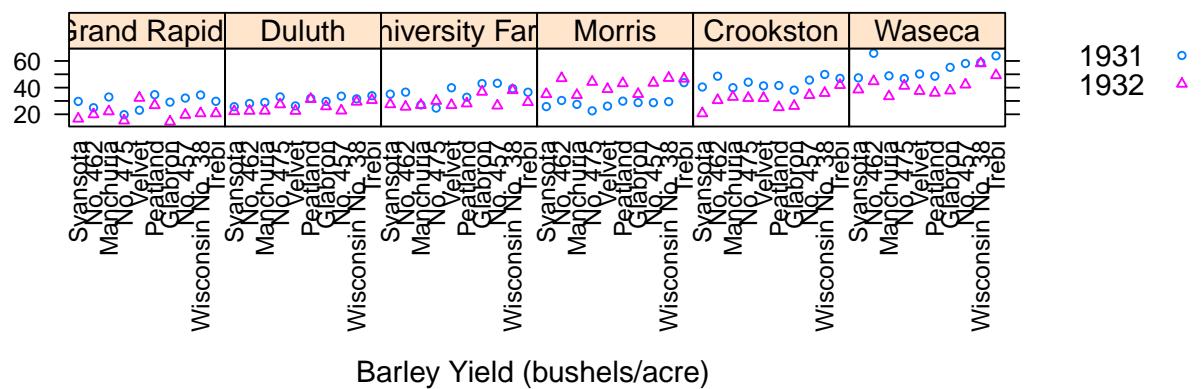
```
bwplot(Sepal.Length ~ Species, data = iris,  
       panel = panel.violin)
```



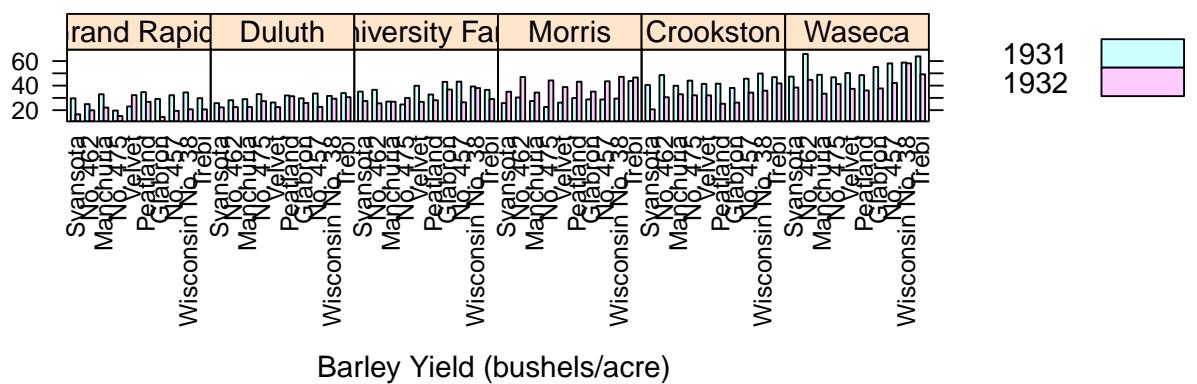
```
bwplot(Sepal.Length ~ Species, data = iris,  
       panel = panel.dotplot)
```



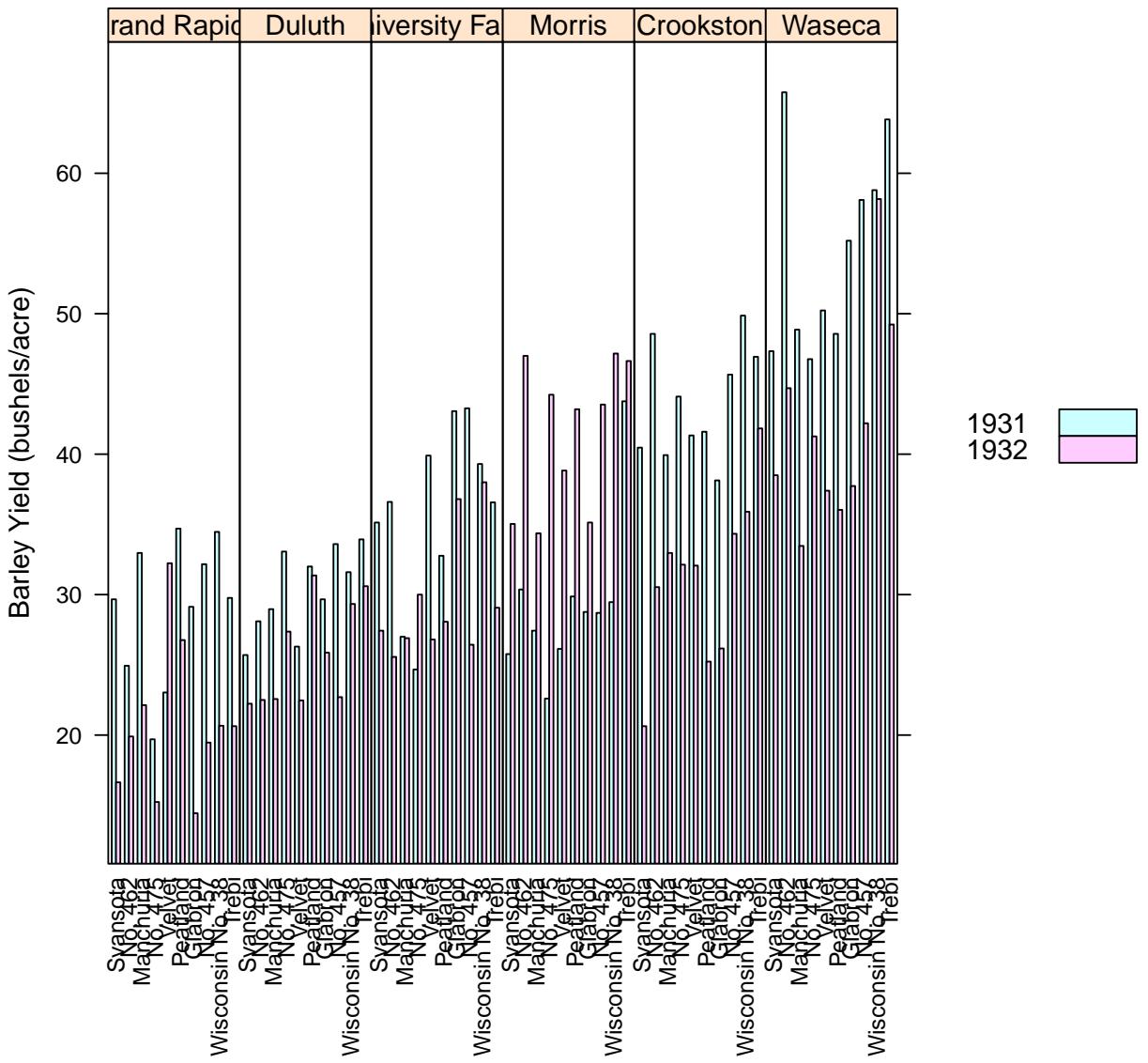
```
barley$year <- factor(barley$year, levels = c(1931,
  1932))
stripplot(yield ~ variety | site, data = barley,
  groups = year, par.settings = simpleTheme(pch = c(1,
  2), cex = 0.5), auto.key = list(space = "right"),
  aspect = 0.5, layout = c(6, 1), xlab = "Barley Yield (bushels/acre) ",
  ylab = NULL, scales = list(x = list(rot = 90)))
```



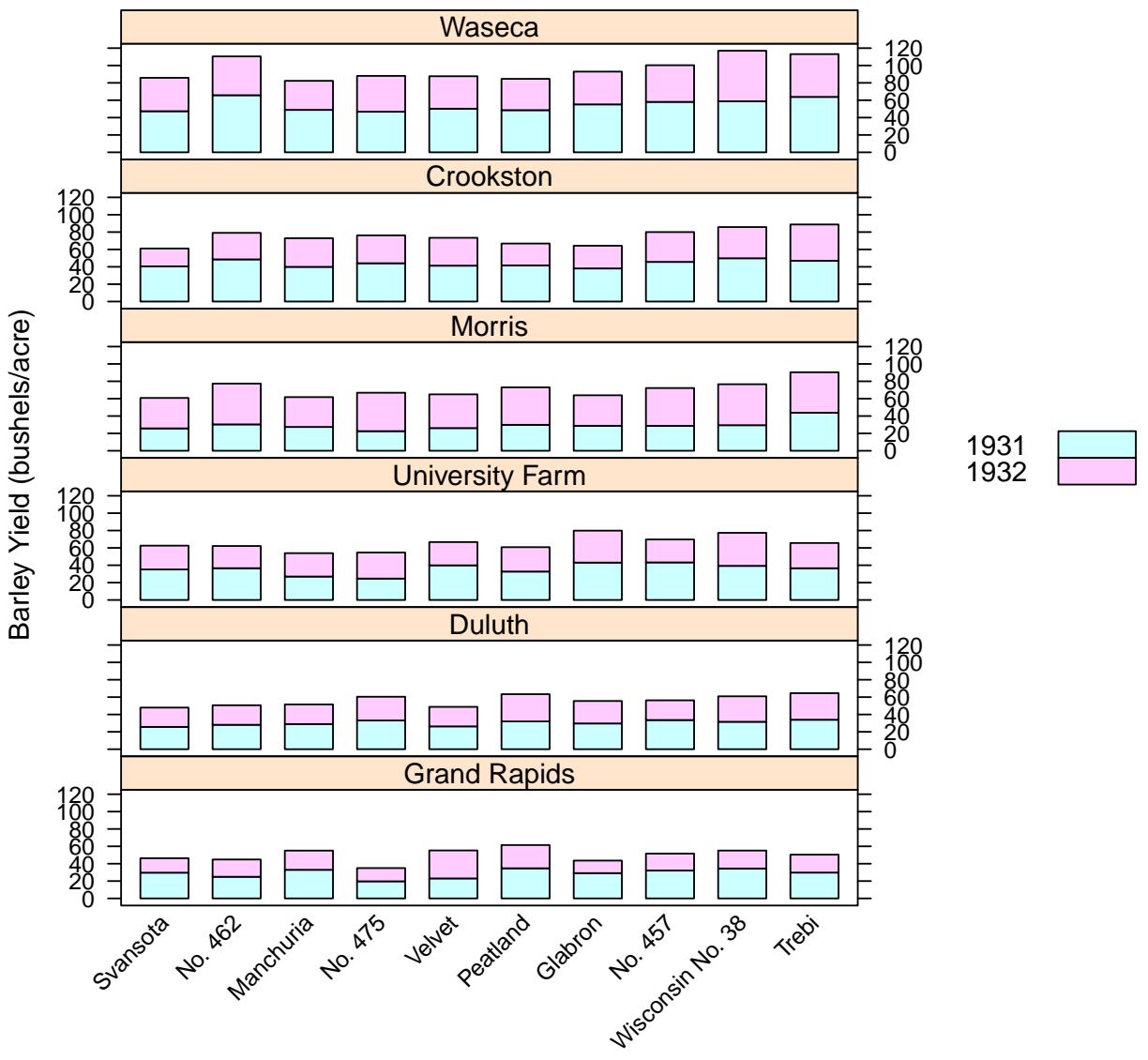
```
barchart(yield ~ variety | site, data = barley,
  groups = year, par.settings = simpleTheme(pch = c(1,
    2), cex = 0.5), auto.key = list(space = "right"),
  aspect = 0.5, layout = c(6, 1), xlab = "Barley Yield (bushels/acre)",
  ylab = NULL, scales = list(x = list(rot = 90)))
```



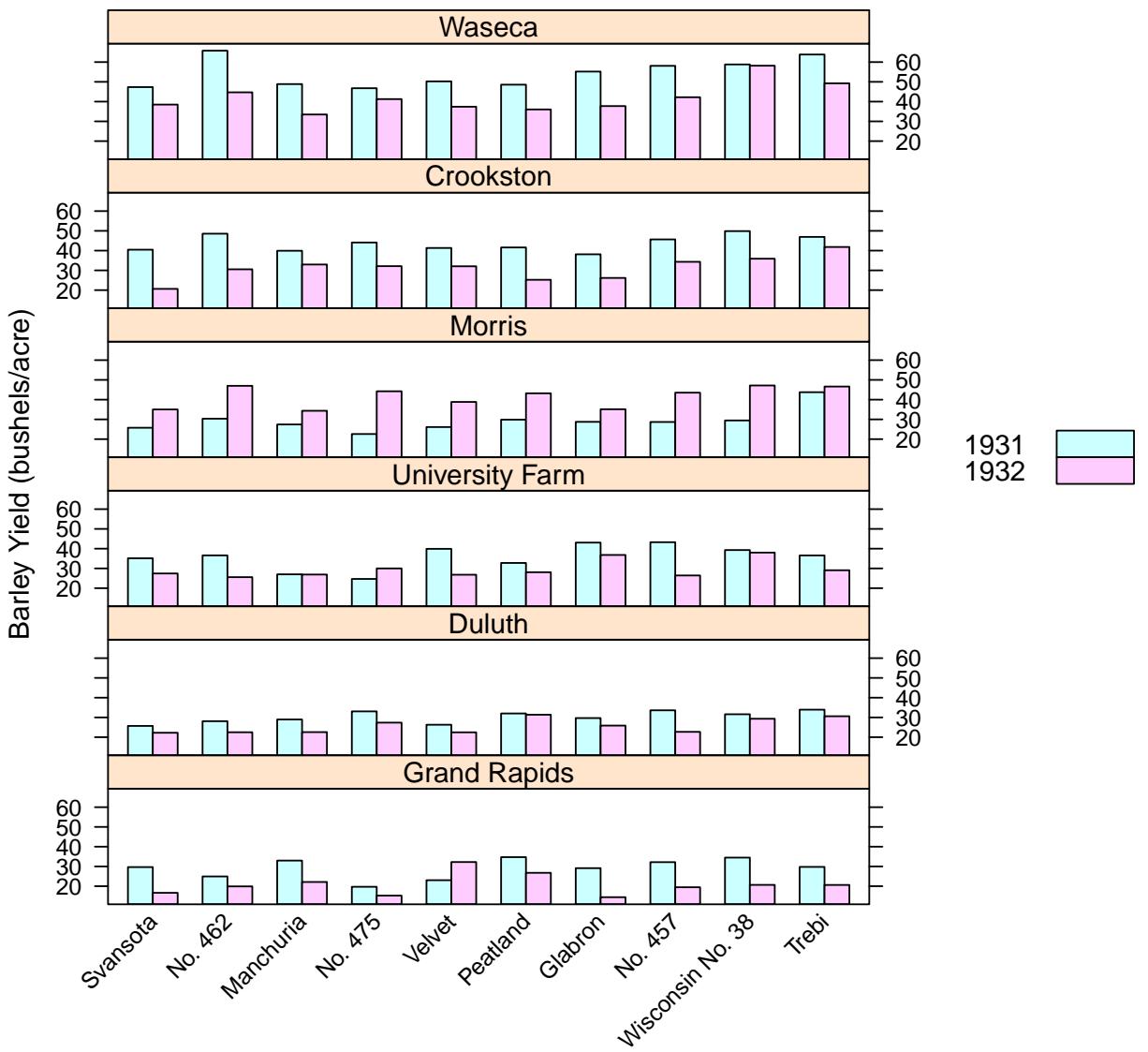
```
barchart(yield ~ variety | site, data = barley,
  groups = year, layout = c(6, 1), stack = FALSE,
  auto.key = list(space = "right"), ylab = "Barley Yield (bushels/acre)",
  scales = list(x = list(rot = 90)))
```



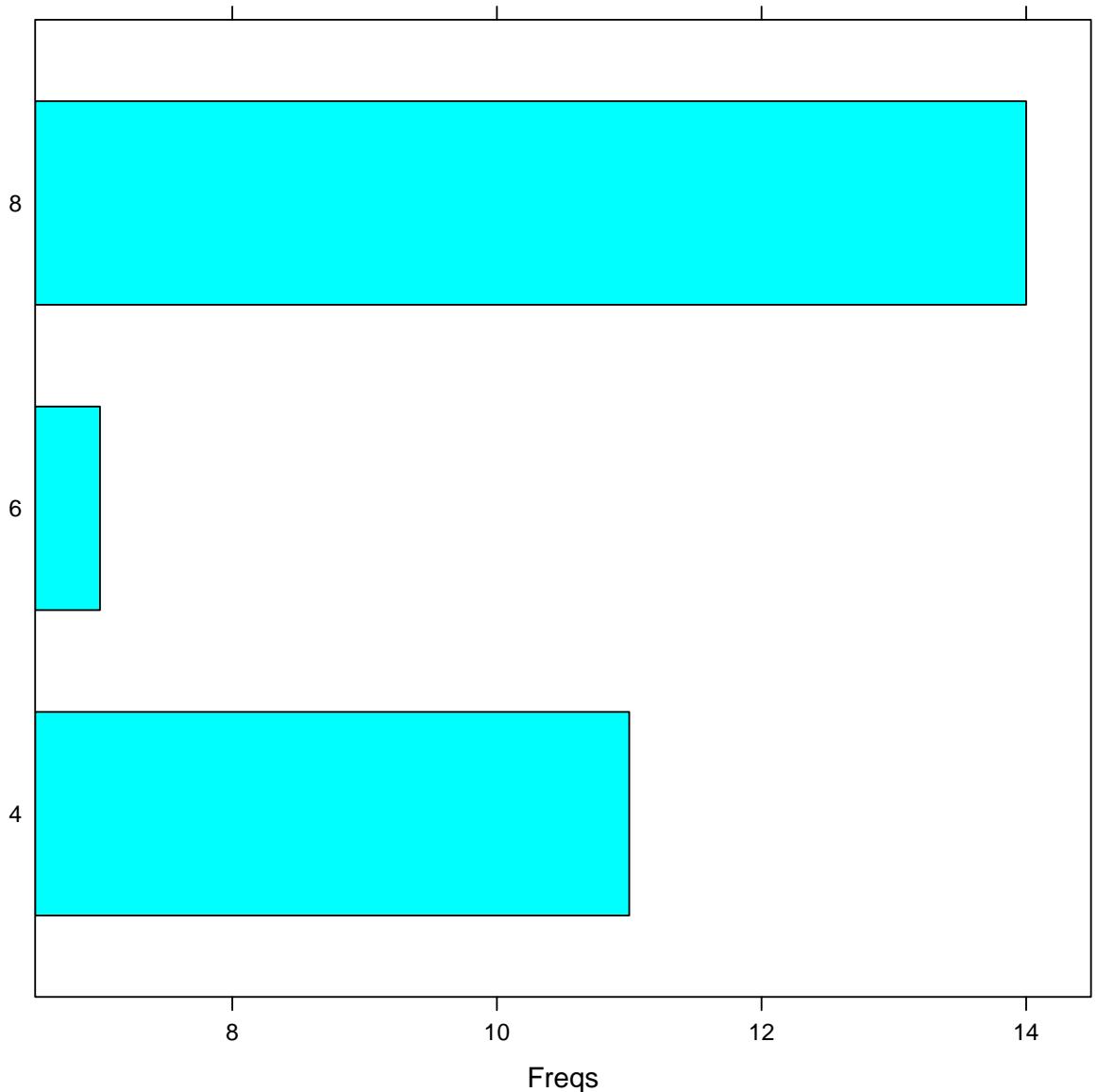
```
barchart(yield ~ variety | site, data = barley,
  groups = year, layout = c(1, 6), stack = TRUE,
  auto.key = list(space = "right"), ylab = "Barley Yield (bushels/acre)",
  scales = list(x = list(rot = 45)))
```



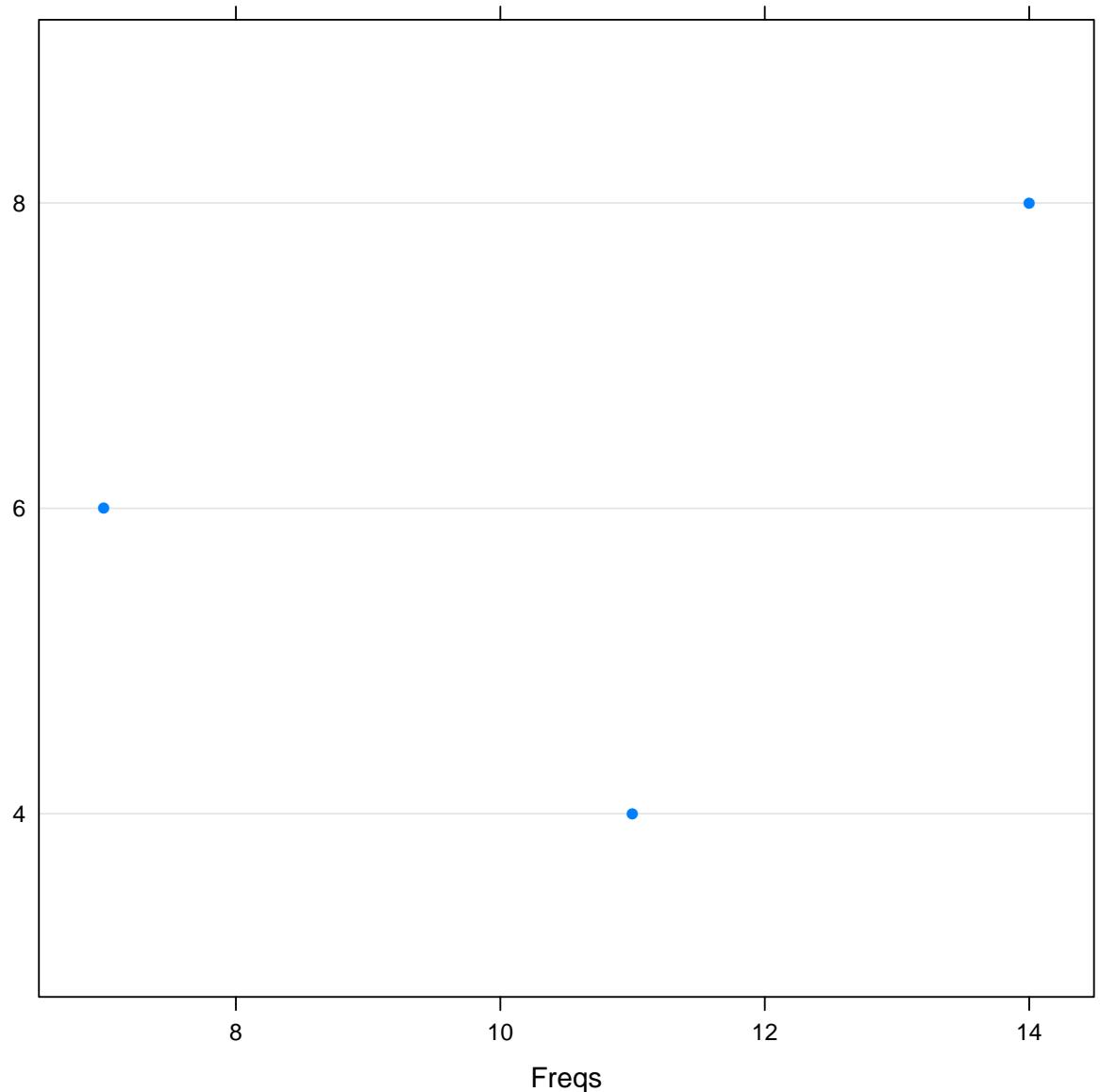
```
barchart(yield ~ variety | site, data = barley,
  groups = year, layout = c(1, 6), stack = FALSE,
  auto.key = list(space = "right"), ylab = "Barley Yield (bushels/acre)",
  scales = list(x = list(rot = 45)))
```



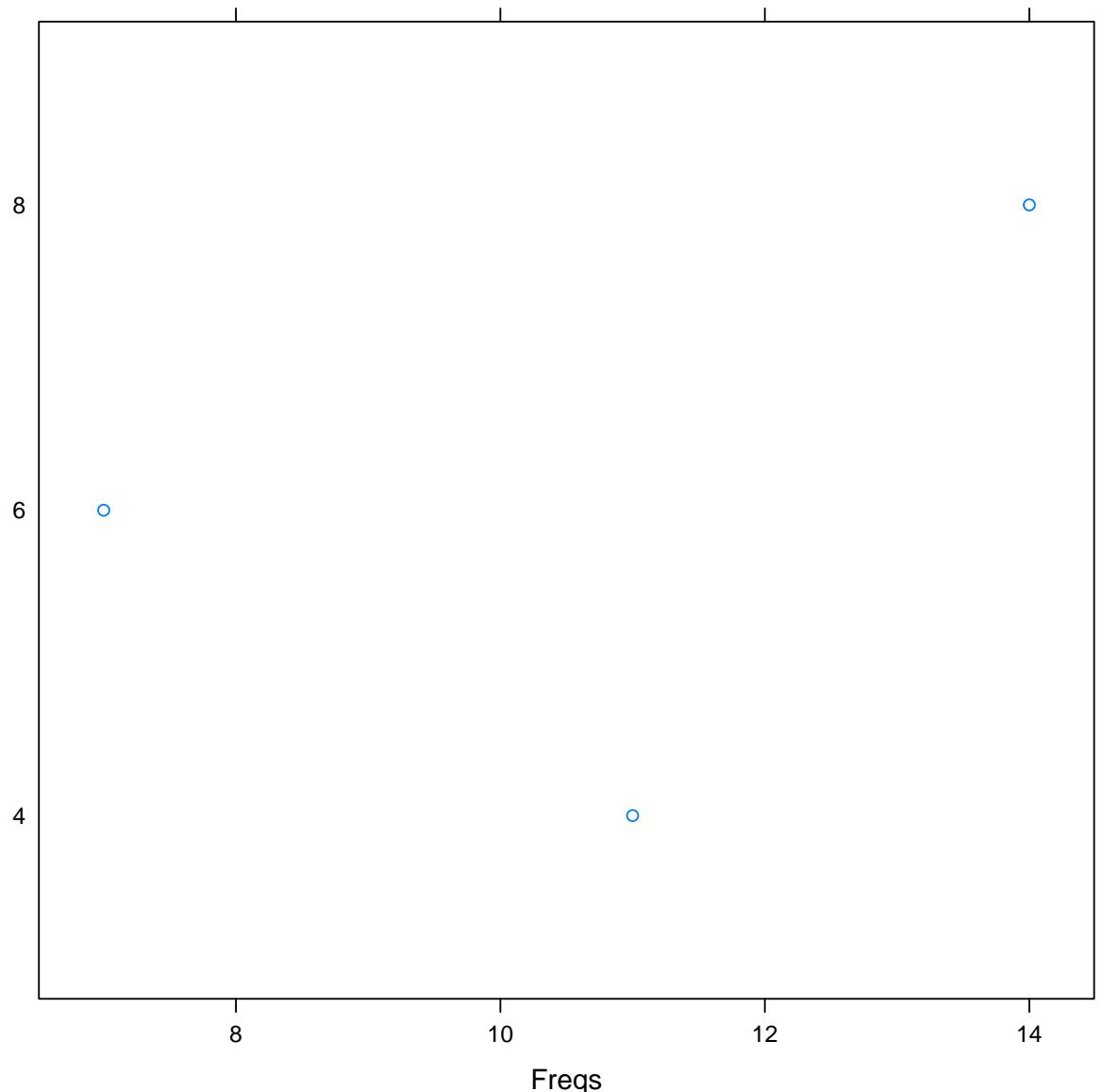
```
barchart(~table(cyl), data = mtcars, xlab = "Freqs")
```



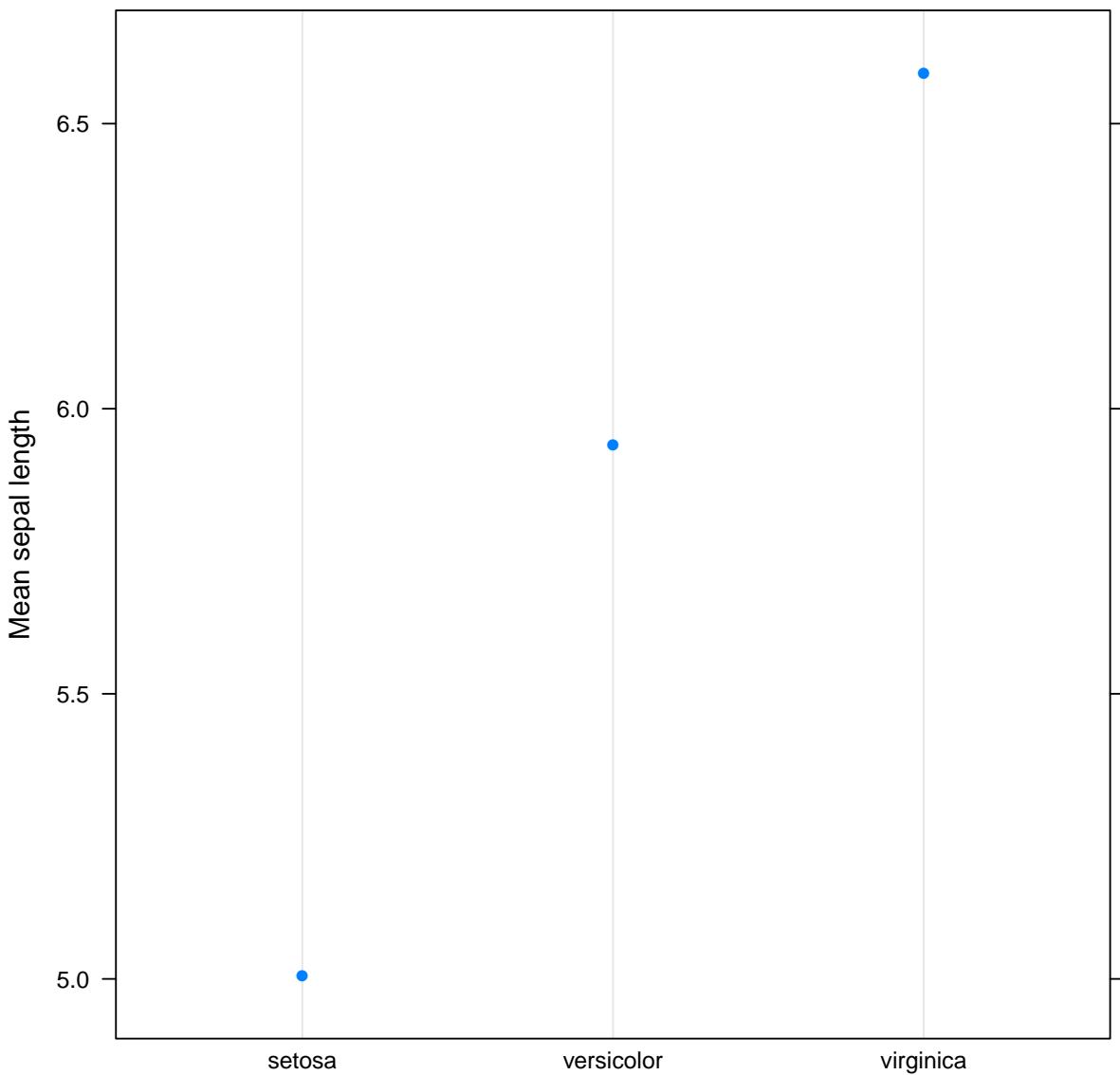
```
dotplot(~table(cyl), data = mtcars, xlab = "Freqs")
```



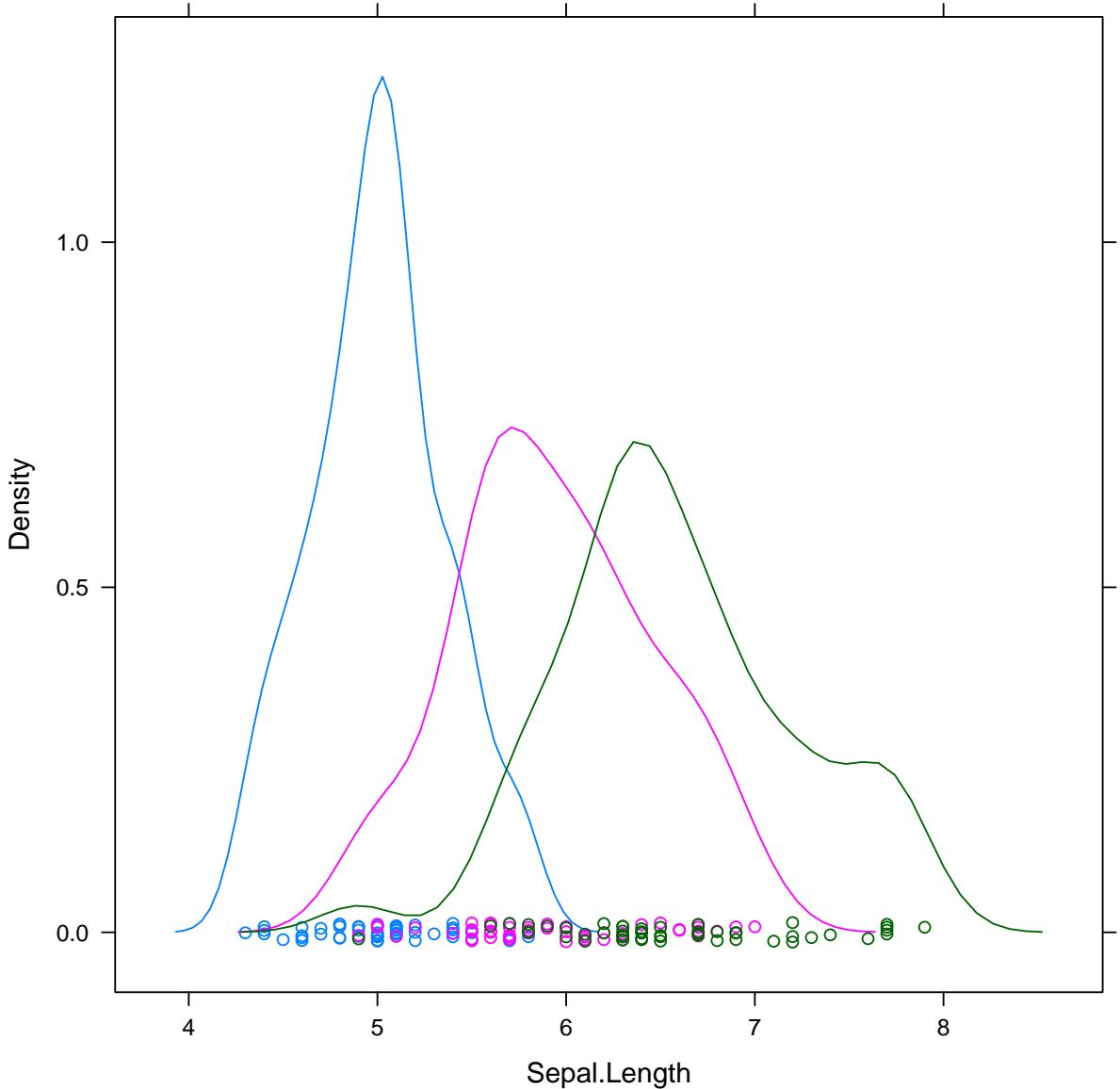
```
stripplot(~table(cyl), data = mtcars, xlab = "Freqs")
```



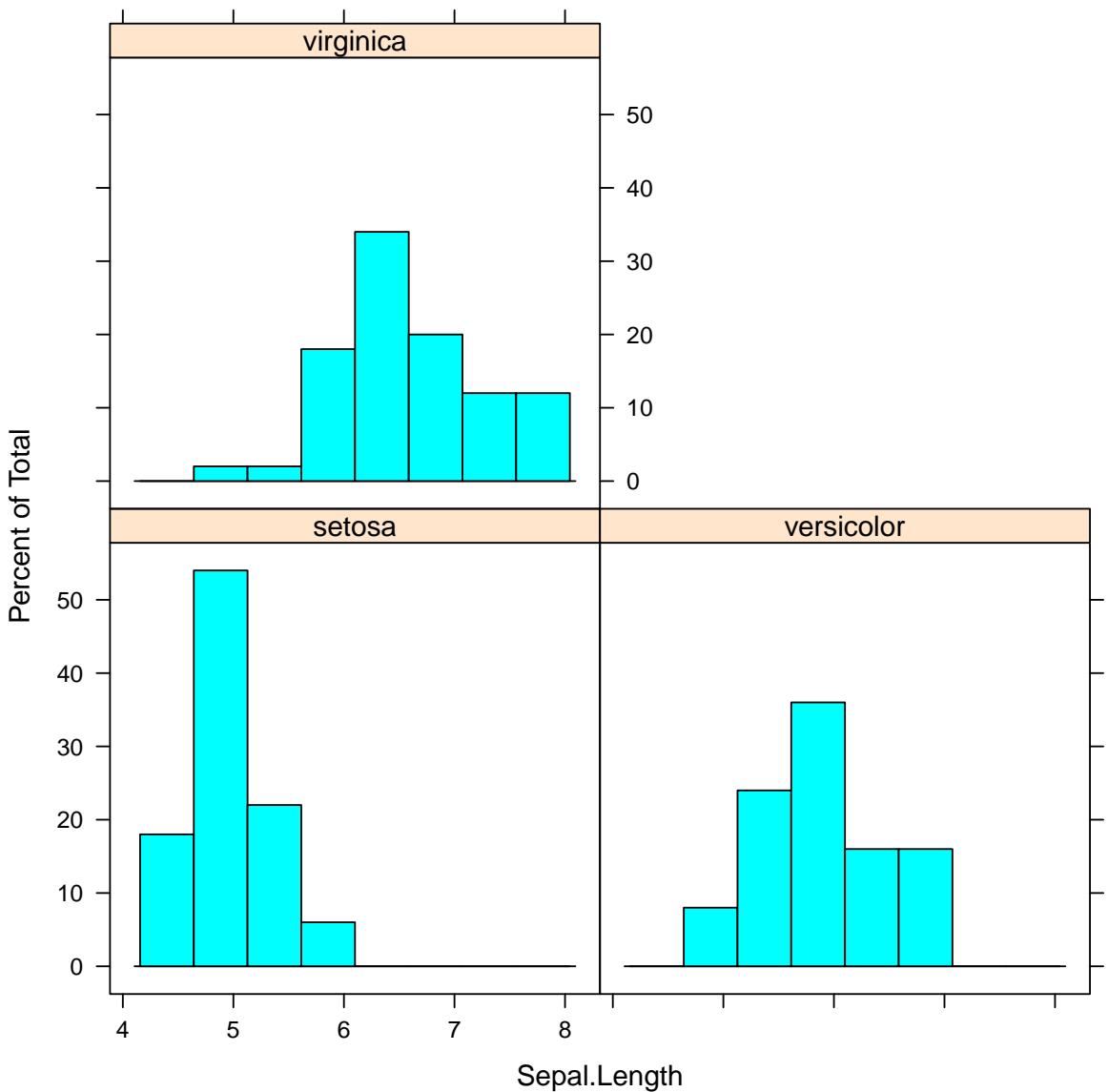
```
dotplot(Sepal.Length ~ Species, data = aggregate(subset(iris,
  select = -Species), list(Species = iris$Species),
  mean), xlab = NULL, ylab = "Mean sepal length")
```



```
densityplot(~Sepal.Length, groups = Species,  
           data = iris)
```

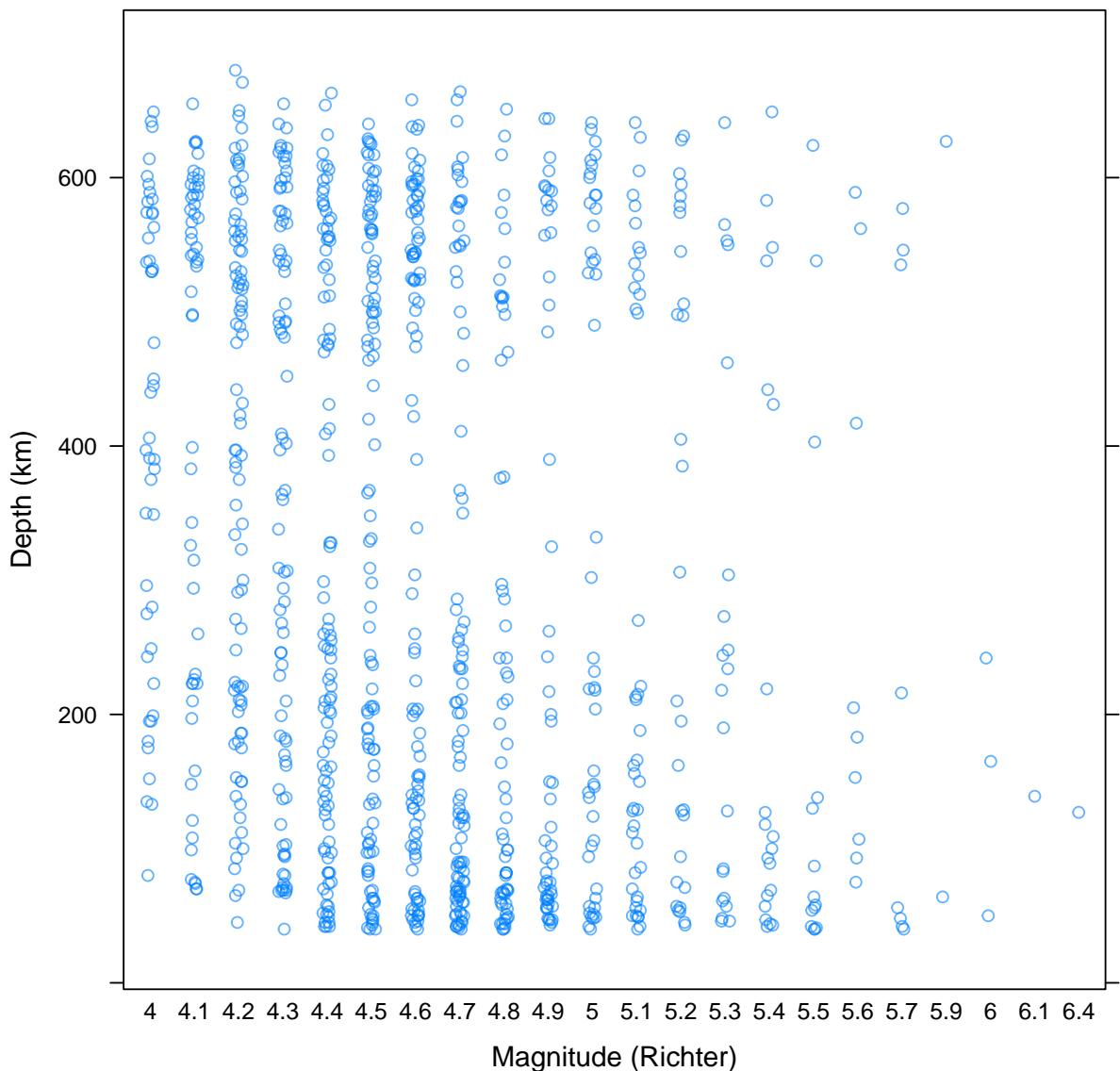


```
histogram(~Sepal.Length | Species, data = iris)
```



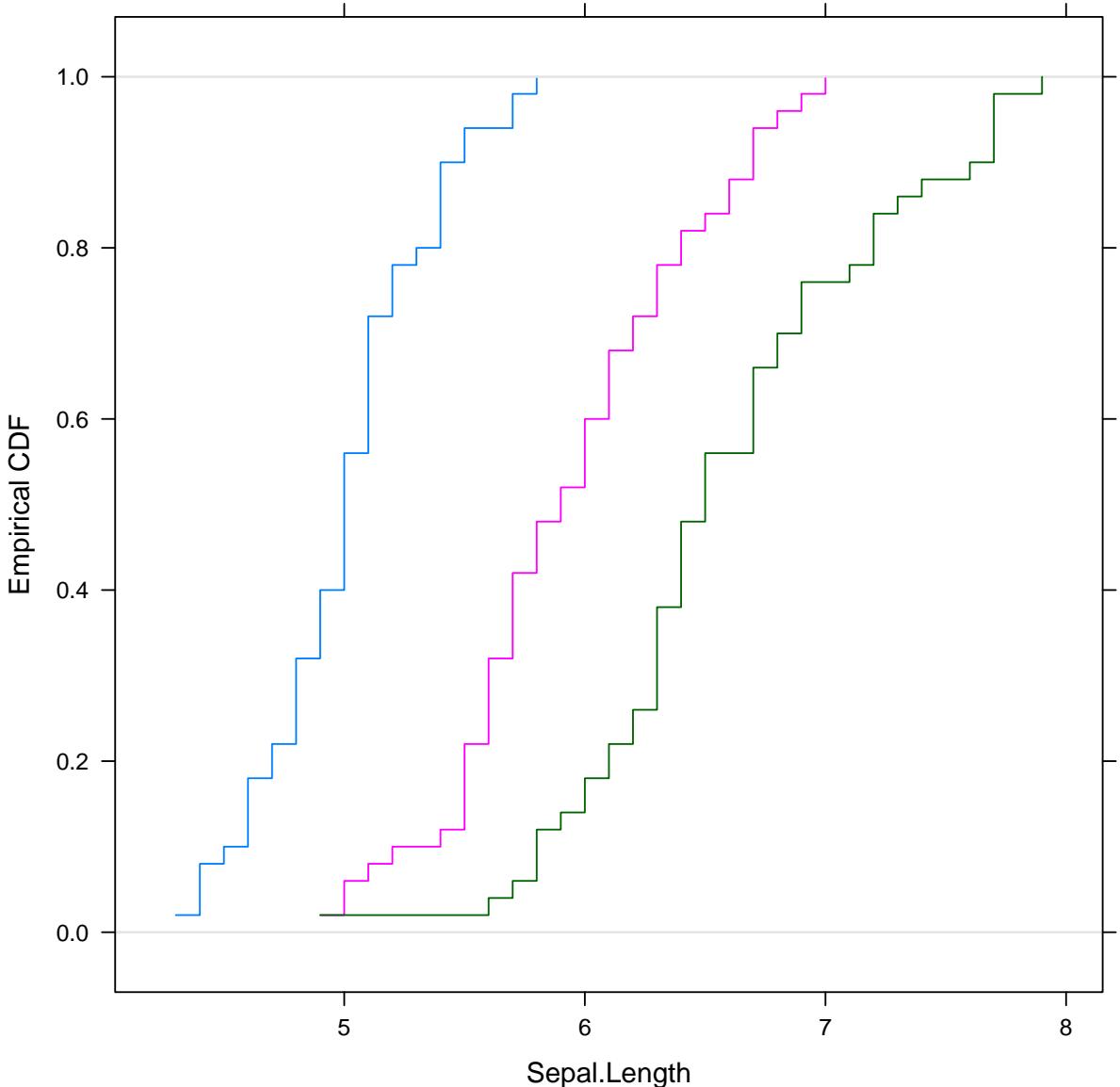
```
# from
# http://lattice.r-forge.r-project.org/Vignettes/src/lattice-intro/lattice-intro.pdf
stripplot(depth ~ factor(mag), data = quakes,
          jitter.data = TRUE, alpha = 0.6, main = "Depth of earthquake epicenters by magnitude",
          xlab = "Magnitude (Richter)", ylab = "Depth (km)")
```

Depth of earthquake epicenters by magnitude

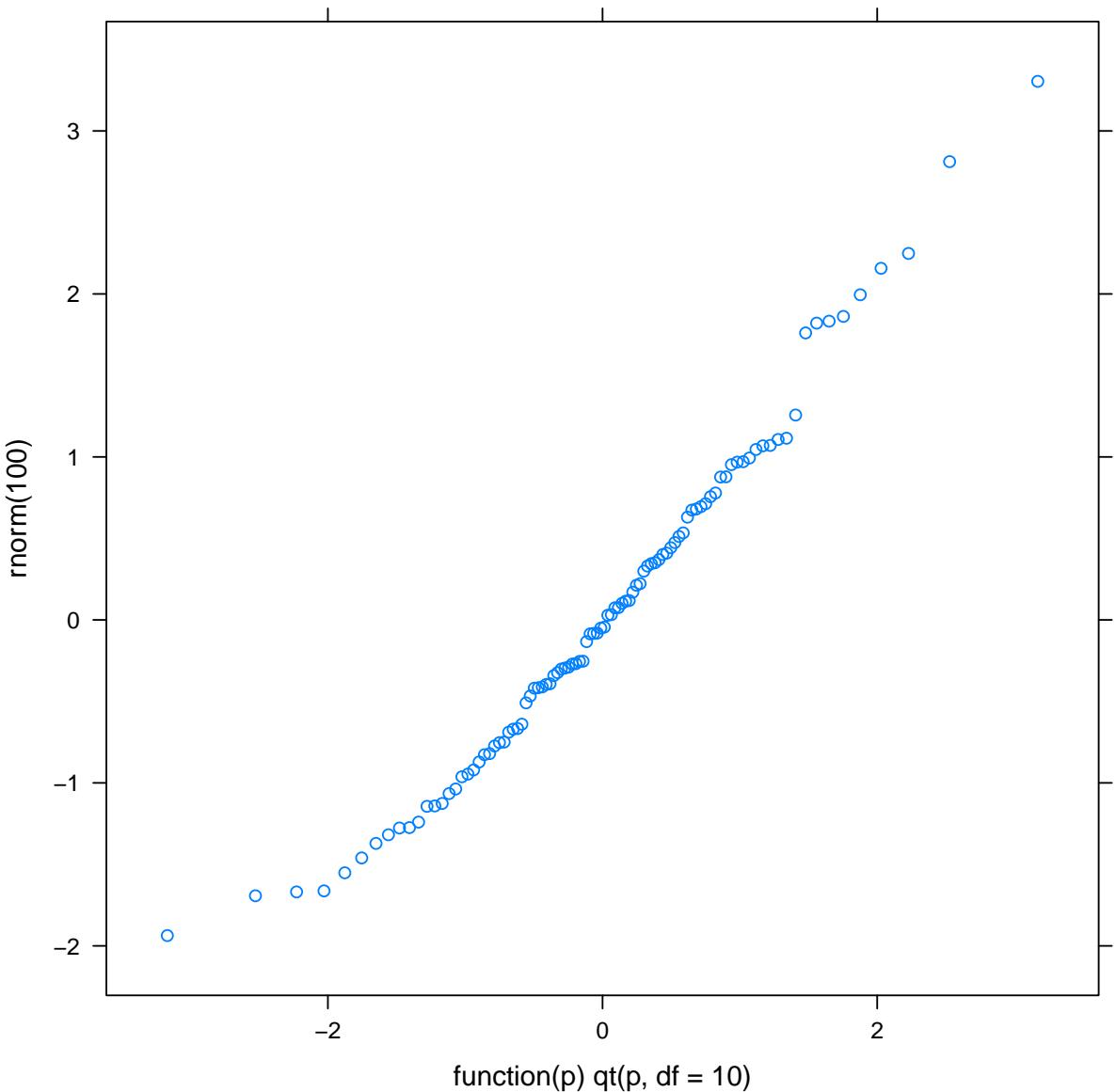


7.3 Densityplots, ecdfplots, qqmath (draft)

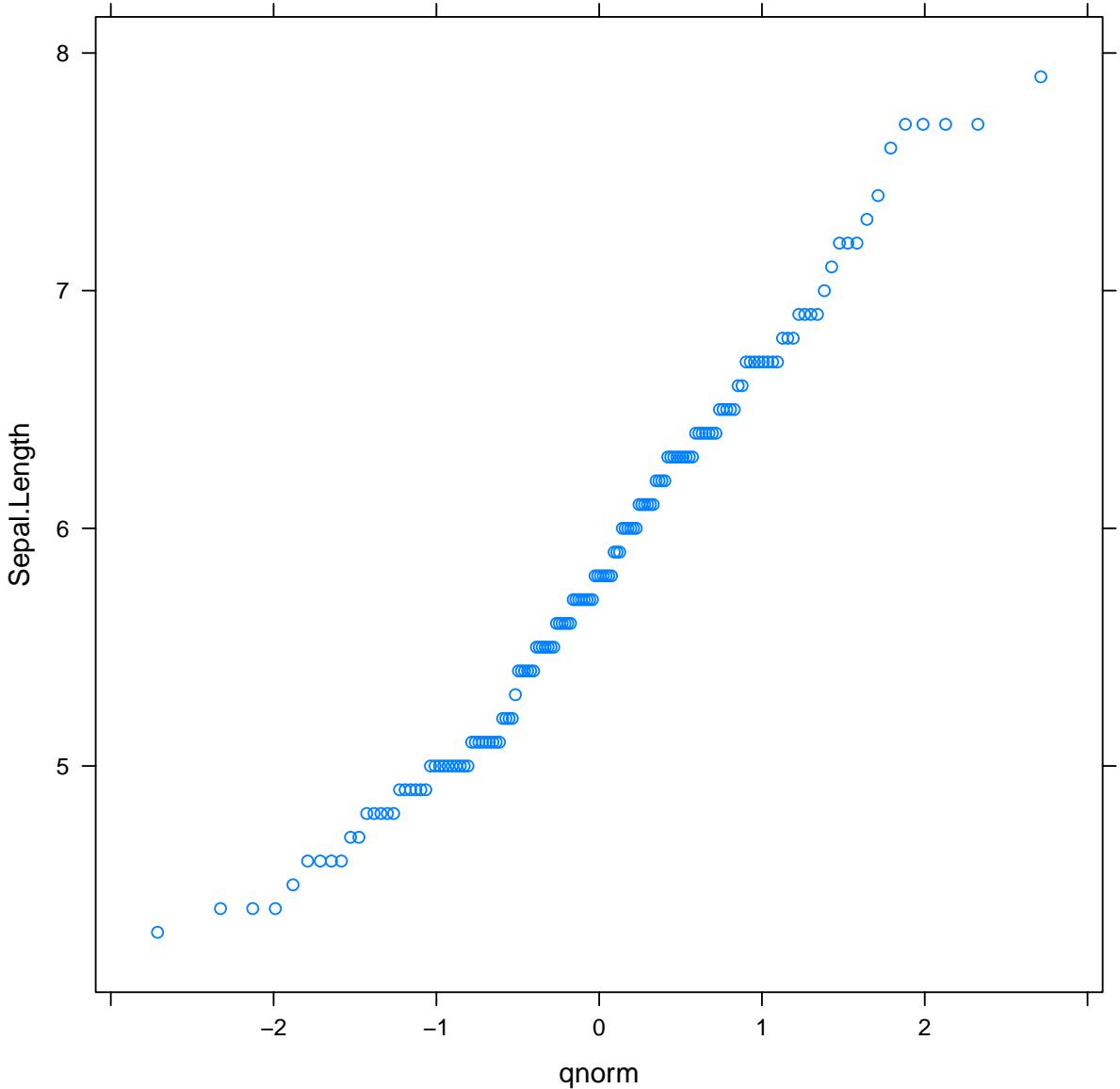
```
ecdfplot(~Sepal.Length, groups = Species,  
        data = iris)
```



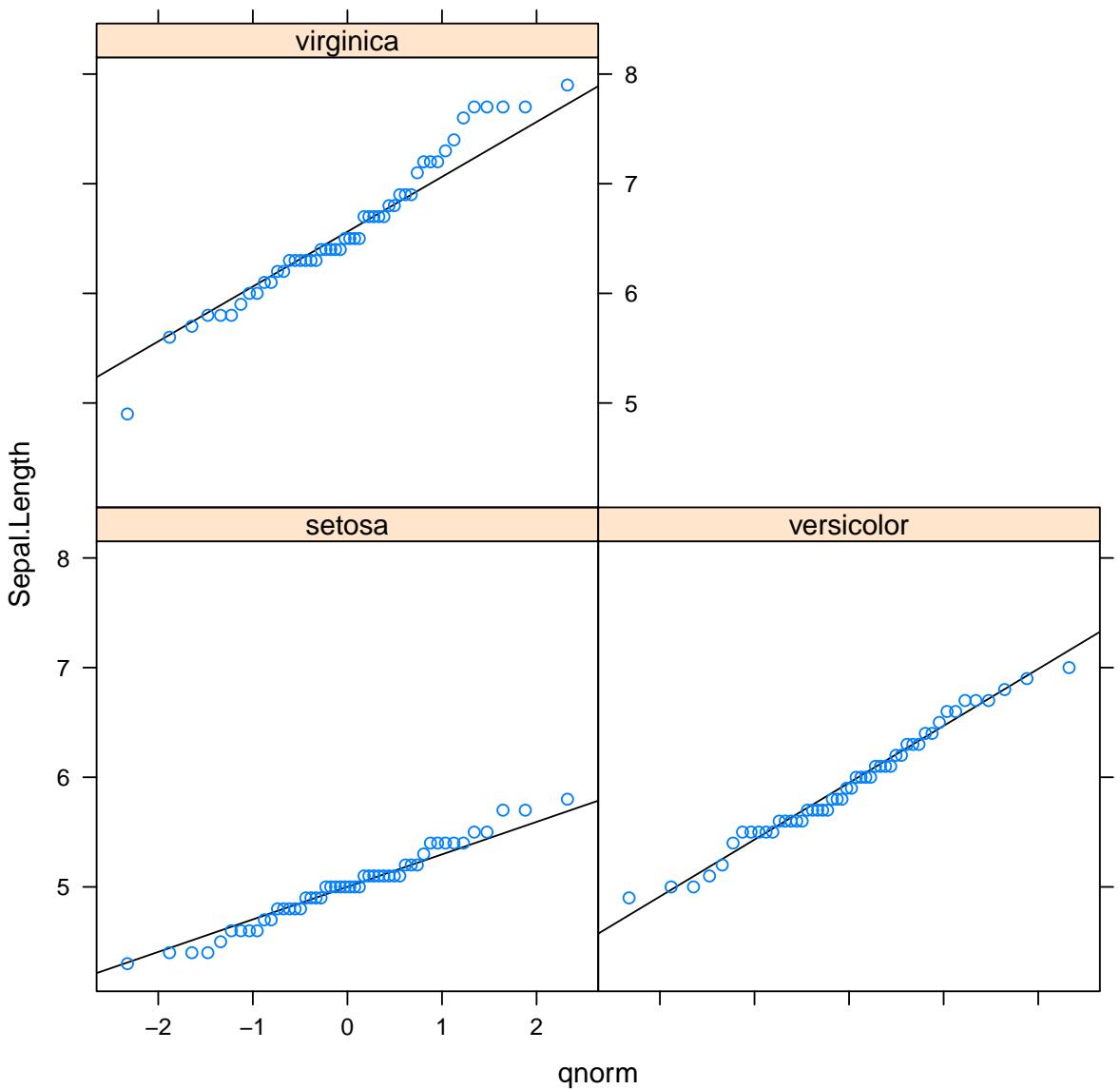
```
qqmath(~rnorm(100), distribution = function(p) qt(p,  
df = 10))
```



```
qqmath(~Sepal.Length, distribution = qnorm,  
      data = iris)
```



```
qqmath(~Sepal.Length | Species, distribution = qnorm,  
      data = iris) + layer(panel.qqmathline(...))
```



TODO: marginal.plot, countourplot