

# R, Quick start to data analysis

Alex Shlemov

Anton Korobeynikov

12 октября 2014 г.

## Содержание

<b>1</b>	<b>Справка, workspaces, запуск скриптов, пакеты</b>	<b>2</b>
1.1	Справка . . . . .	2
1.2	Переменные, рабочие пространства (workspaces), история команд, выход	2
1.3	Запуск скриптов . . . . .	3
1.4	Пакеты . . . . .	3
<b>2</b>	<b>Вектора, матрицы, массивы</b>	<b>4</b>
2.1	Вектора, основные операции . . . . .	4
2.2	Вектора, доступ к элементам (subscripting, индексная техника) . . . . .	7
2.2.1	Числовой вектор индексов . . . . .	8
2.2.2	Логический вектор-маска . . . . .	8
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 . . . . .	12
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
<b>3</b>	<b>Списки</b>	<b>15</b>
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
<b>4</b>	<b>Материалы с занятия 3 октября</b>	<b>19</b>
4.1	Toothgrowth . . . . .	19
4.2	Графики residuals-vs-fitted . . . . .	25
4.3	Линейная регрессия (Университеты) . . . . .	34
4.3.1	Advertising, окончательный результат . . . . .	47

<b>5</b>	<b>Материалы с занятия 10 октября</b>	<b>54</b>
5.1	LDA и tune . . . . .	57
5.2	Default . . . . .	62
5.3	Smarket . . . . .	66
5.4	banknote . . . . .	74
<b>6</b>	<b>Рисование</b>	<b>80</b>

# 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 вельюенед 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("as1/rssa")
# 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 понадобиться 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
> 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:<sup>1</sup>

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

### 2.2.2 Логический вектор-маска

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

---

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



```

> 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 самой на себя:  $x^T x$  и  $xx^T$ 

diag(m) # для матрицы, возвращает вектор главной диагонали, при этом
        доступна 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, ...) # Склейка матриц по столбцам:  $[a : b : c : \dots]$ 
rbind(a, b, c, ...) # Склейка 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

схлопывания 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: 0x2dbabd8>
## <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 подписка

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

```

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

ll

```

```
## $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:

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

## 4 Материалы с занятия 3 октября

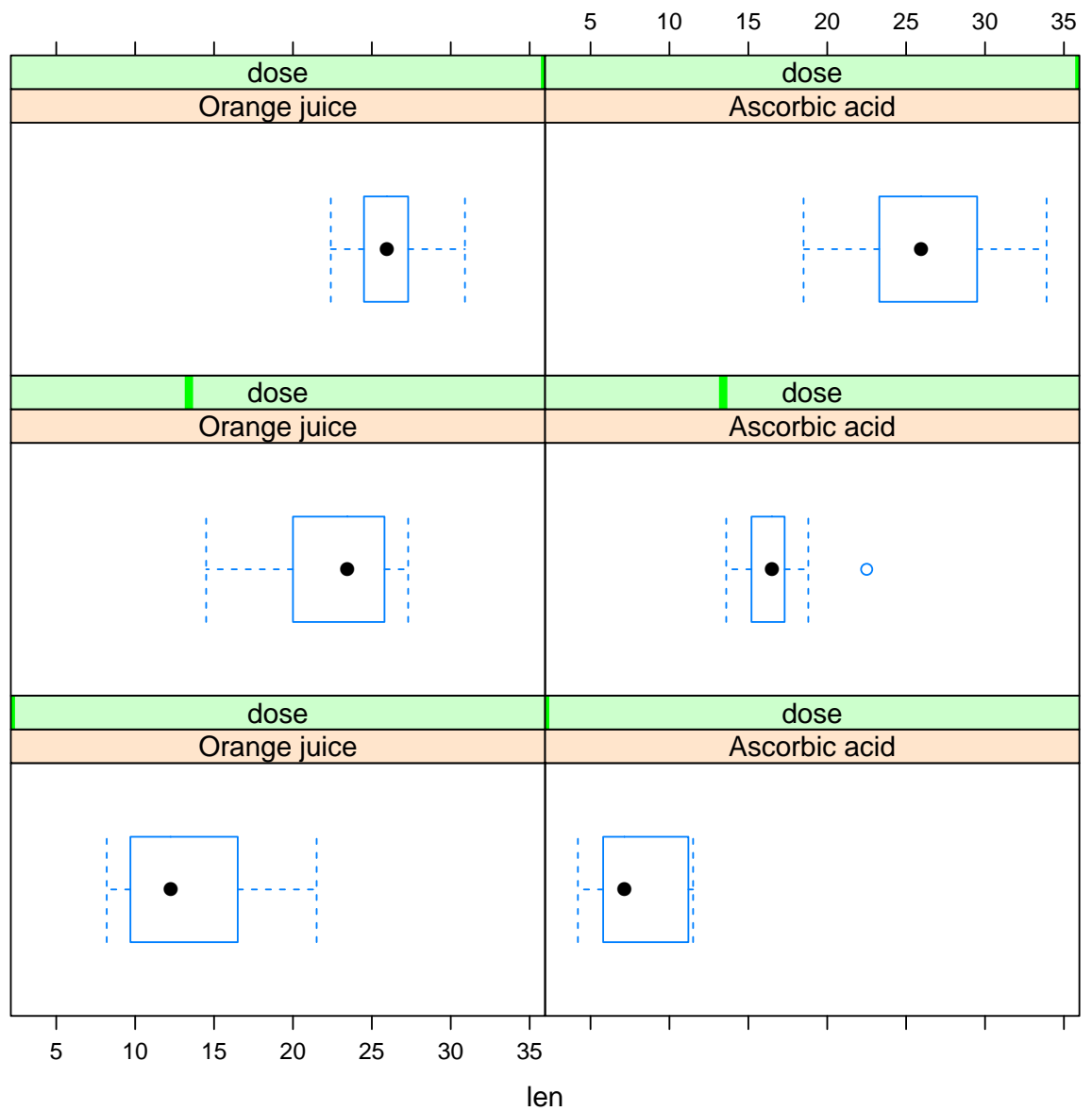
### 4.1 Toothgrowth

```
library(lattice)
library(latticeExtra)
library(MASS)

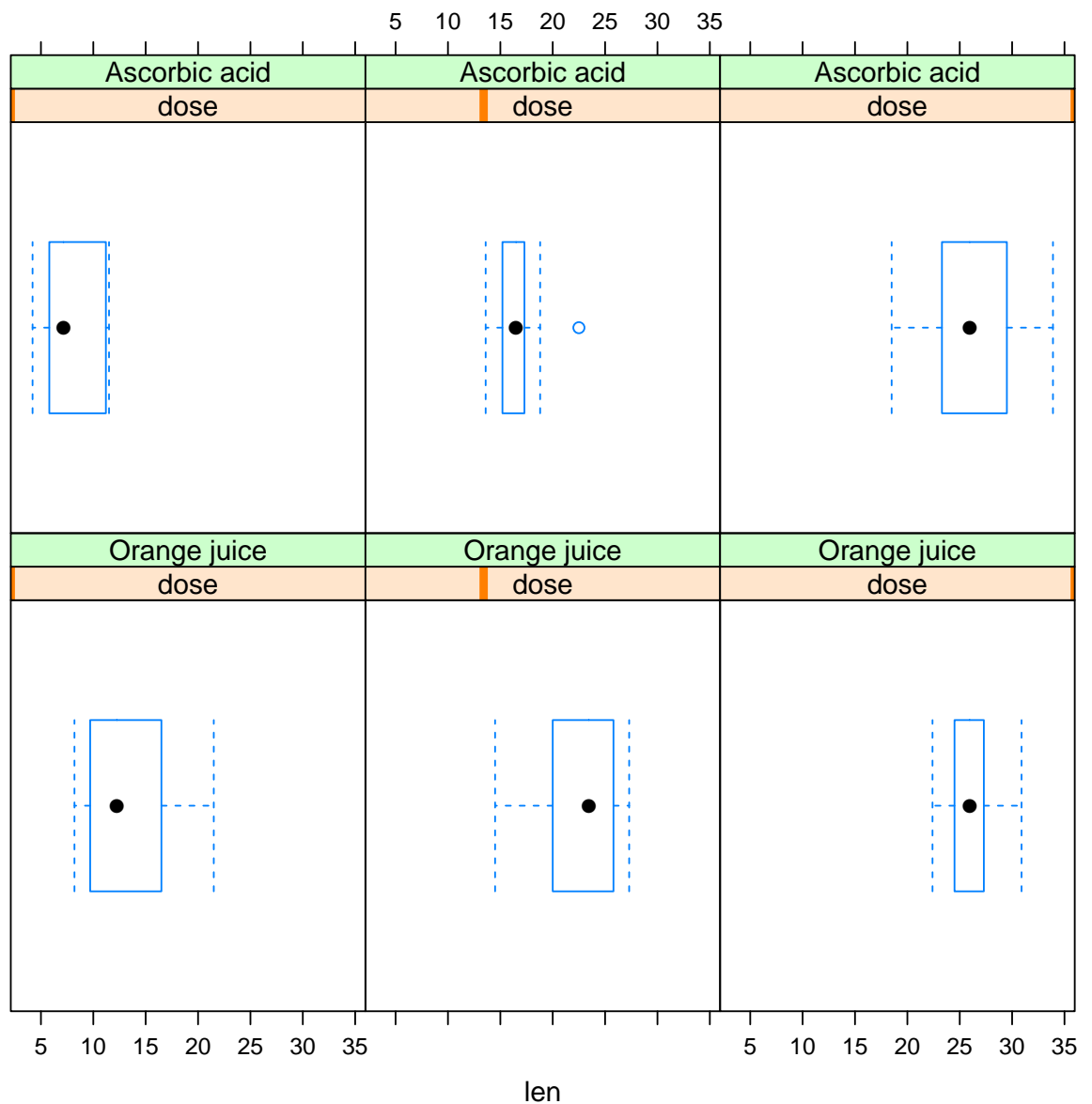
tooth <- read.table("toothgrowth.txt")

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)    7.422      1.160     6.40 3.2e-08 ***
## suppl         1.850      0.547     3.38 0.0013 **
## dose          9.764      0.877    11.14 6.3e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 4.24 on 57 degrees of freedom
## Multiple R-squared:  0.704, Adjusted R-squared:  0.693
## F-statistic: 67.7 on 2 and 57 DF,  p-value: 8.72e-16

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

##
## Call:
## lm(formula = len ~ supp * dose, data = tooth)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -8.23  -2.85   0.05   2.29   7.94
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept)    7.422      1.118     6.64 1.4e-08 ***
## suppl         4.127      1.118     3.69 0.00051 ***
## dose          9.764      0.845    11.55 < 2e-16 ***
## suppl:dose    -1.952      0.845    -2.31 0.02463 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 4.08 on 56 degrees of freedom
## Multiple R-squared:  0.73, Adjusted R-squared:  0.715
## F-statistic: 50.4 on 3 and 56 DF,  p-value: 6.52e-16

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

##              .L      .Q
## [1,] -7.071e-01  0.4082
```

```
## [2,] -7.850e-17 -0.8165
## [3,]  7.071e-01  0.4082

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)   18.813     0.469   40.13 < 2e-16 ***
## supp1         1.850     0.469    3.95 0.00023 ***
## dose1         4.565     0.574    7.95 1.2e-10 ***
## dose2         3.643     0.332   10.99 2.2e-15 ***
## supp1:dose1    0.170     0.574    0.30 0.76831
## supp1:dose2   -0.945     0.332   -2.85 0.00617 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 3.63 on 54 degrees of freedom
## Multiple R-squared:  0.794, Adjusted R-squared:  0.775
## F-statistic: 41.6 on 5 and 54 DF, p-value: <2e-16

stepAIC(l)

## Start:  AIC=160.4
## len ~ supp * dose
##
##              Df Sum of Sq RSS AIC
## <none>                712 160
## - supp:dose  2          108 820 165
##
## Call:
## lm(formula = len ~ supp * dose, data = tooth)
##
```

```
## Coefficients:
## (Intercept)      supp1      dose1      dose2  supp1:dose1
##      18.813      1.850      4.565      3.643      0.170
## supp1:dose2
##      -0.945

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

# the smaller AIC/BIC, the better the fit
AIC(l, l.lin)

##      df    AIC
## 1      7 332.7
## l.lin  5 337.2

BIC(l, l.lin)

##      df    BIC
## 1      7 347.4
## l.lin  5 347.7

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
## 2      56 820 -2      -108 4.11  0.022 *
## ---
## 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,
```

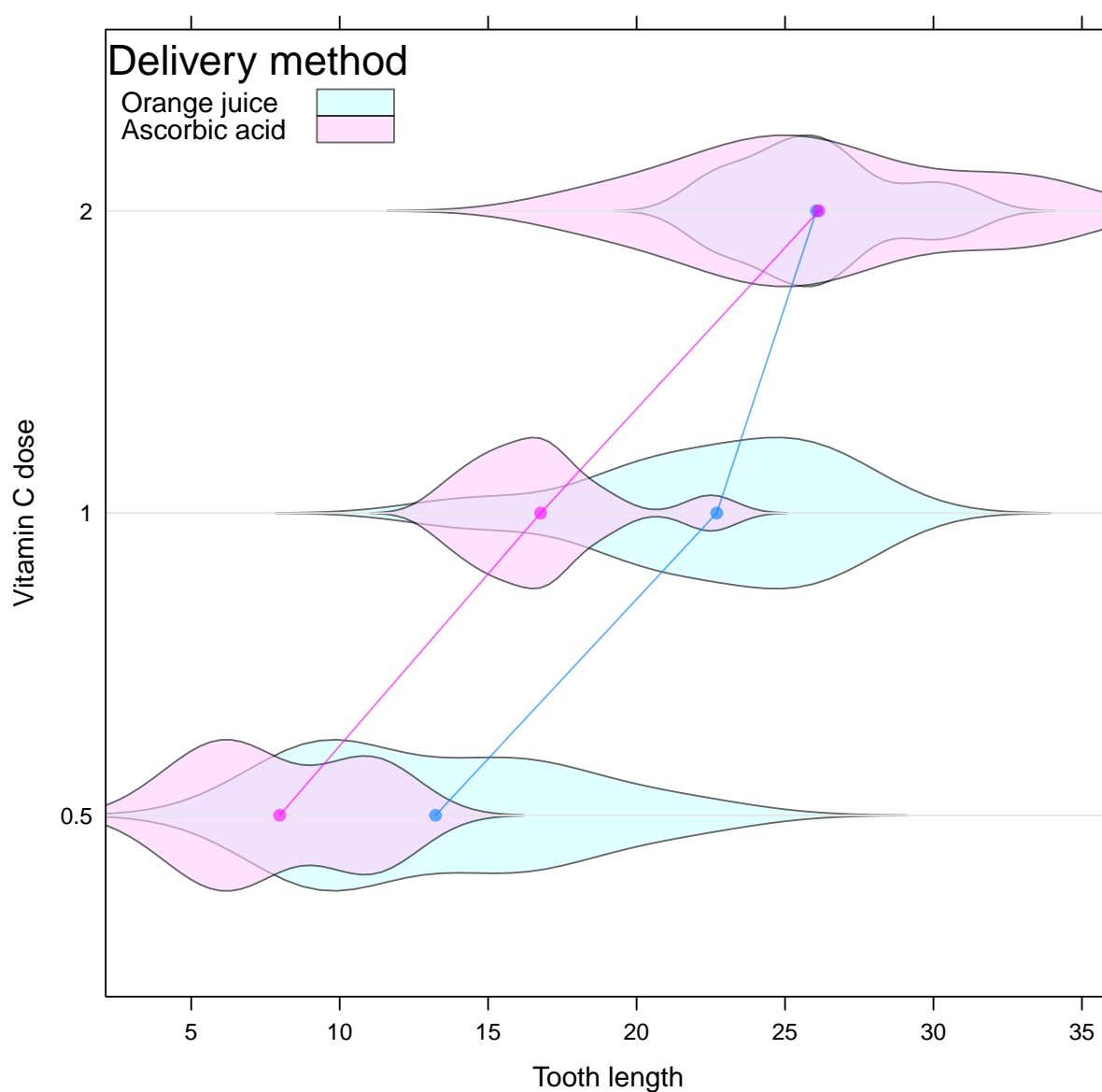


```

        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)

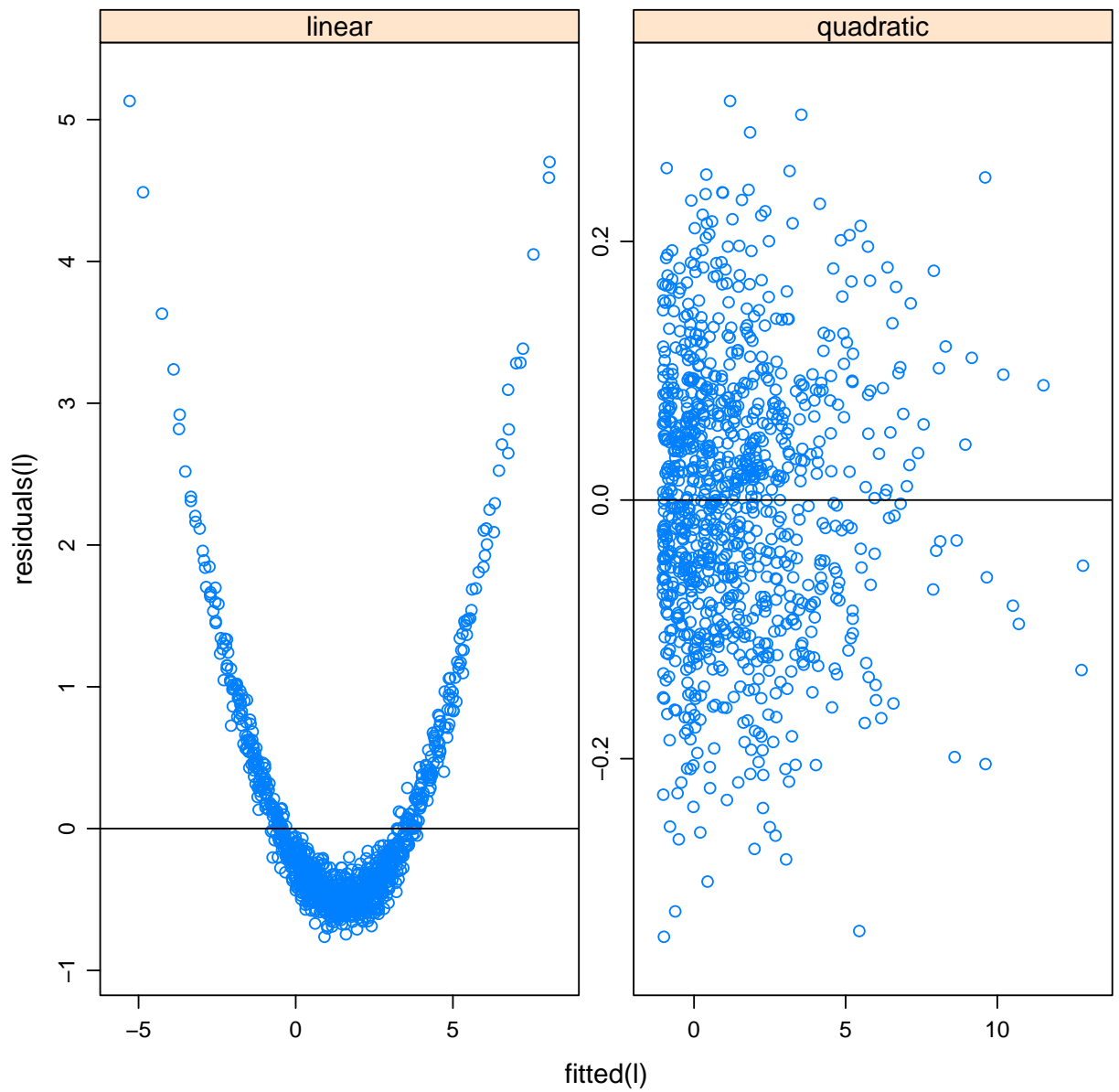
df <- data.frame(y = y, x = x)

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))

```



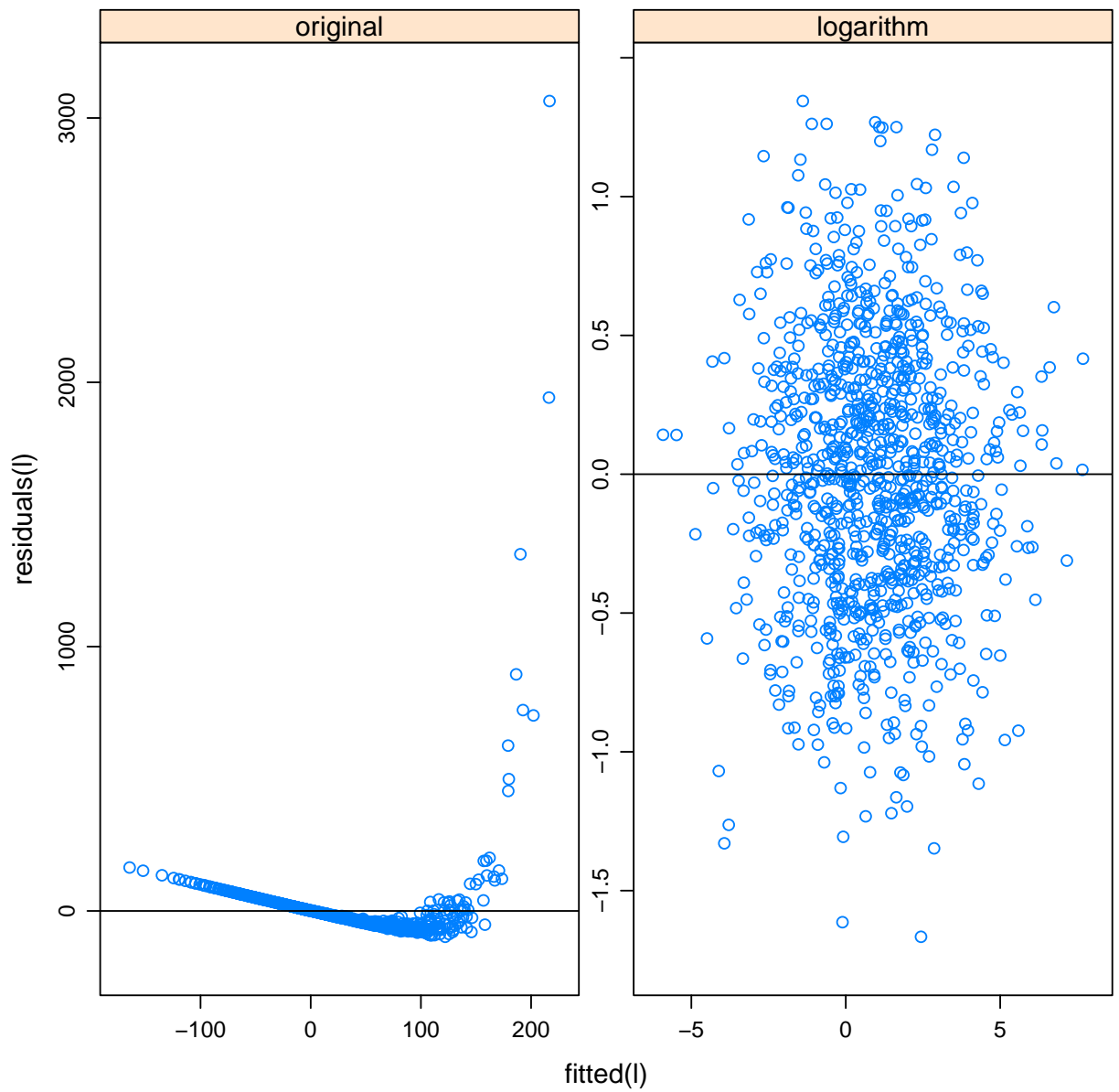
```

y <- exp(beta0 + beta1 * x + rnorm(N, sd = 0.5))

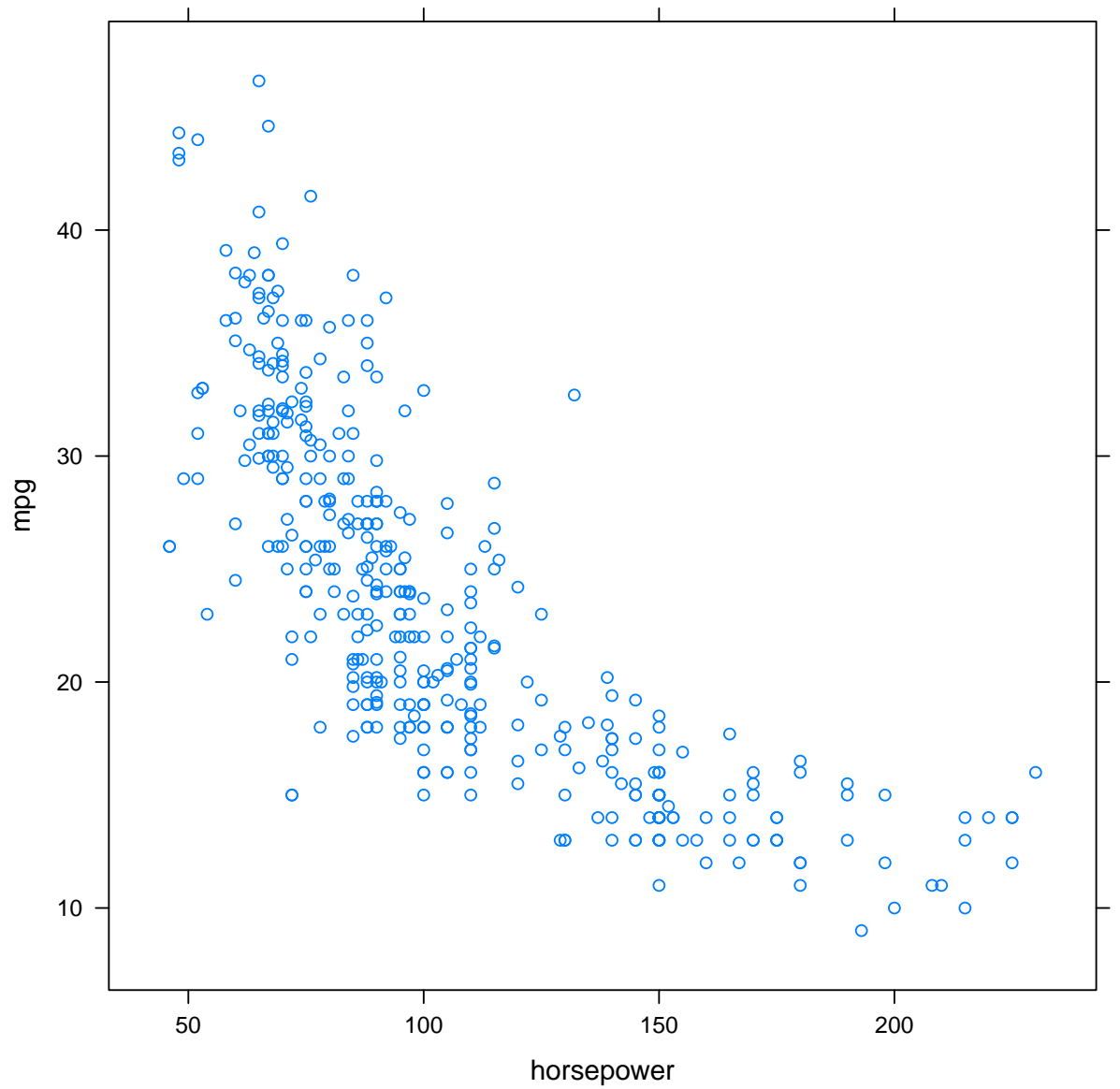
df <- data.frame(y = y, x = x)

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))

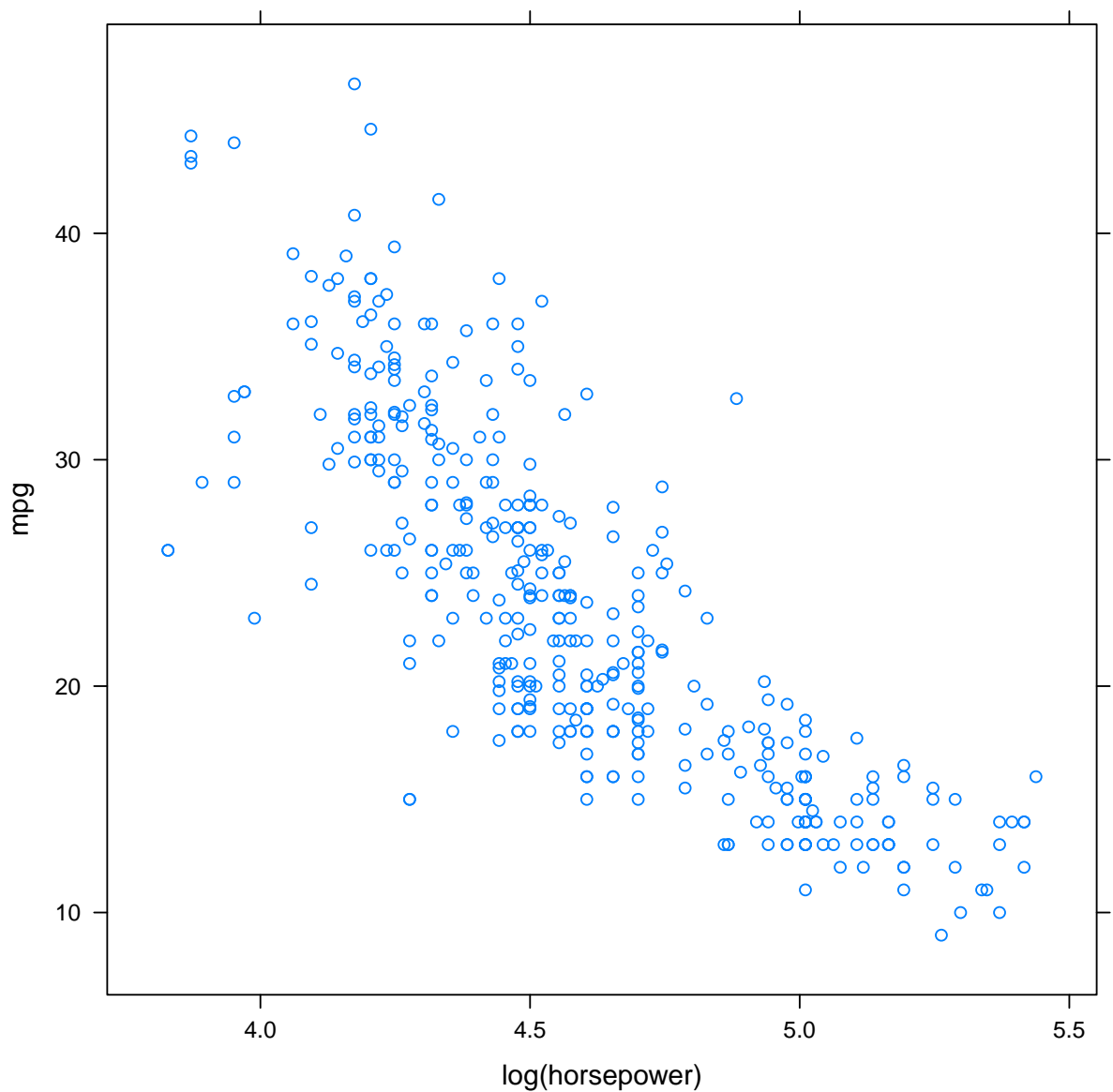
```



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



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



```
l1 <- lm(mpg ~ horsepower, data = Auto)
l2 <- lm(mpg ~ poly(horsepower, degree = 2), data = Auto)
llog <- lm(mpg ~ log(horsepower), data = Auto)
l5 <- lm(mpg ~ poly(horsepower, degree = 5), data = Auto)
l6 <- lm(mpg ~ poly(horsepower, degree = 6), data = Auto)
summary(l1)

##
## Call:
## lm(formula = mpg ~ horsepower, data = Auto)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -13.571  -3.259  -0.344   2.763  16.924
##
```

```
## Coefficients:
##           Estimate Std. Error t value Pr(>|t|)
## (Intercept) 39.93586    0.71750   55.7   <2e-16 ***
## horsepower -0.15784    0.00645  -24.5   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 4.91 on 390 degrees of freedom
## Multiple R-squared:  0.606, Adjusted R-squared:  0.605
## F-statistic: 600 on 1 and 390 DF, p-value: <2e-16

summary(llog)

##
## Call:
## lm(formula = mpg ~ log(horsepower), data = Auto)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -14.230  -2.782  -0.232   2.666  15.470
##
## Coefficients:
##           Estimate Std. Error t value Pr(>|t|)
## (Intercept)    108.700      3.050   35.6   <2e-16 ***
## log(horsepower) -18.582      0.663  -28.0   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 4.5 on 390 degrees of freedom
## Multiple R-squared:  0.668, Adjusted R-squared:  0.667
## F-statistic: 786 on 1 and 390 DF, p-value: <2e-16

summary(l5)

##
## Call:
## lm(formula = mpg ~ poly(horsepower, degree = 5), data = Auto)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -15.433  -2.529  -0.293   2.175  15.973
##
## Coefficients:
##           Estimate Std. Error t value
## (Intercept)      23.446      0.218  107.31
## poly(horsepower, degree = 5)1 -120.138      4.326  -27.77
## poly(horsepower, degree = 5)2  44.090      4.326   10.19
## poly(horsepower, degree = 5)3  -3.949      4.326   -0.91
## poly(horsepower, degree = 5)4  -5.188      4.326   -1.20
```

```
## poly(horsepower, degree = 5)5    13.272      4.326    3.07
##                                Pr(>|t|)
## (Intercept)                    <2e-16 ***
## poly(horsepower, degree = 5)1    <2e-16 ***
## poly(horsepower, degree = 5)2    <2e-16 ***
## poly(horsepower, degree = 5)3    0.3619
## poly(horsepower, degree = 5)4    0.2312
## poly(horsepower, degree = 5)5    0.0023 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 4.33 on 386 degrees of freedom
## Multiple R-squared:  0.697, Adjusted R-squared:  0.693
## F-statistic: 177 on 5 and 386 DF, p-value: <2e-16

summary(l6)

##
## 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)                   23.446     0.218  107.72
## poly(horsepower, degree = 6)1 -120.138     4.310  -27.88
## poly(horsepower, degree = 6)2   44.090     4.310   10.23
## poly(horsepower, degree = 6)3   -3.949     4.310   -0.92
## poly(horsepower, degree = 6)4   -5.188     4.310   -1.20
## poly(horsepower, degree = 6)5   13.272     4.310    3.08
## poly(horsepower, degree = 6)6   -8.546     4.310   -1.98
##                                Pr(>|t|)
## (Intercept)                    <2e-16 ***
## poly(horsepower, degree = 6)1    <2e-16 ***
## poly(horsepower, degree = 6)2    <2e-16 ***
## poly(horsepower, degree = 6)3    0.3601
## poly(horsepower, degree = 6)4    0.2294
## poly(horsepower, degree = 6)5    0.0022 **
## poly(horsepower, degree = 6)6    0.0481 *
## ---
## 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:  0.7, Adjusted R-squared:  0.695
## F-statistic: 150 on 6 and 385 DF, p-value: <2e-16
```



```

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

##      df  AIC
## l1    3 2363
## l2    4 2274
## llog   3 2296
## l5    7 2269
## l6    8 2267

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

##      df  BIC
## l1    3 2375
## l2    4 2290
## llog   3 2308
## l5    7 2296
## l6    8 2298

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 9386
## 2      389 7442  1      1944 102 <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 9386
## 2      386 7223  4      2163 28.9 <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 9386
## 2      385 7150  5      2236 24.1 <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
## 2      385 7150  1         73 3.93  0.048 *
## ---
## 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 вдоволь 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?)
#
# Итааааааак, 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 H.S. 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 общежитии
#
# ADD_FEE - Дополнительные сборы (сверх платы за обучение,
# койку и учебные материалы)
#
# PH_D - Процент кандидатов наук среди педагогического состава.
#
# GRADUAT - Процент выпускников. 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.

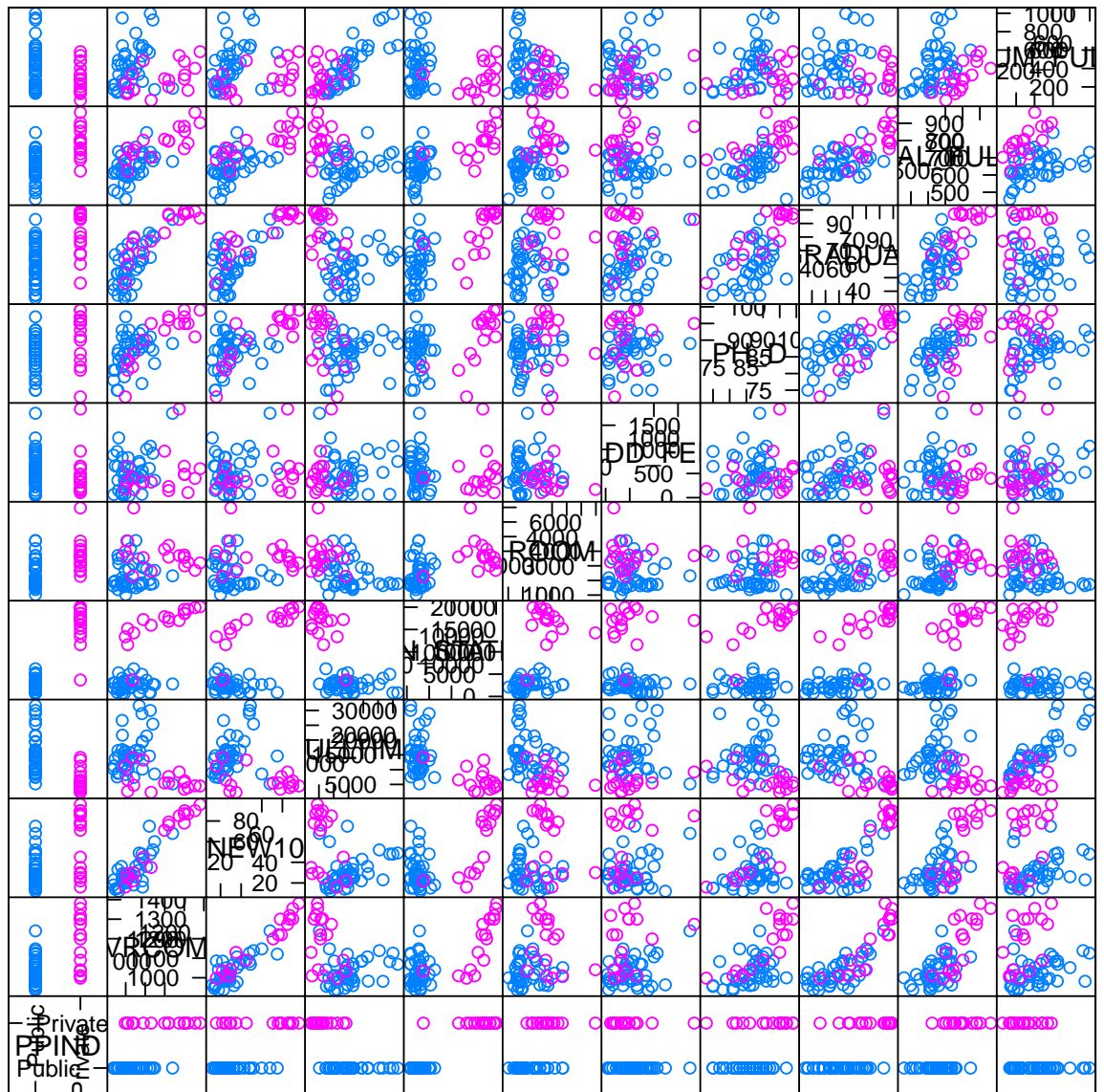
# latticist(df)

# 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)

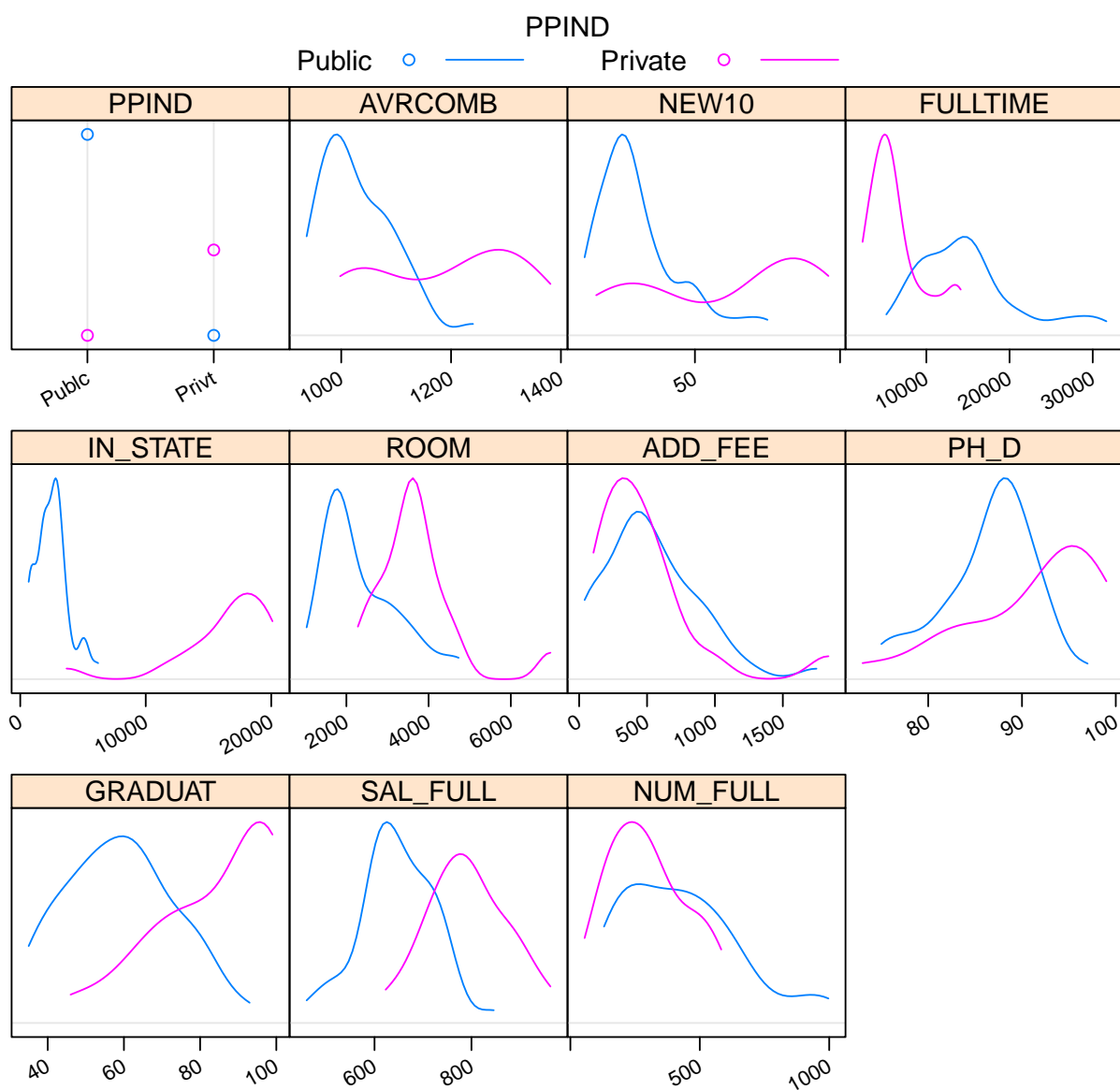
```



Scatter Plot Matrix

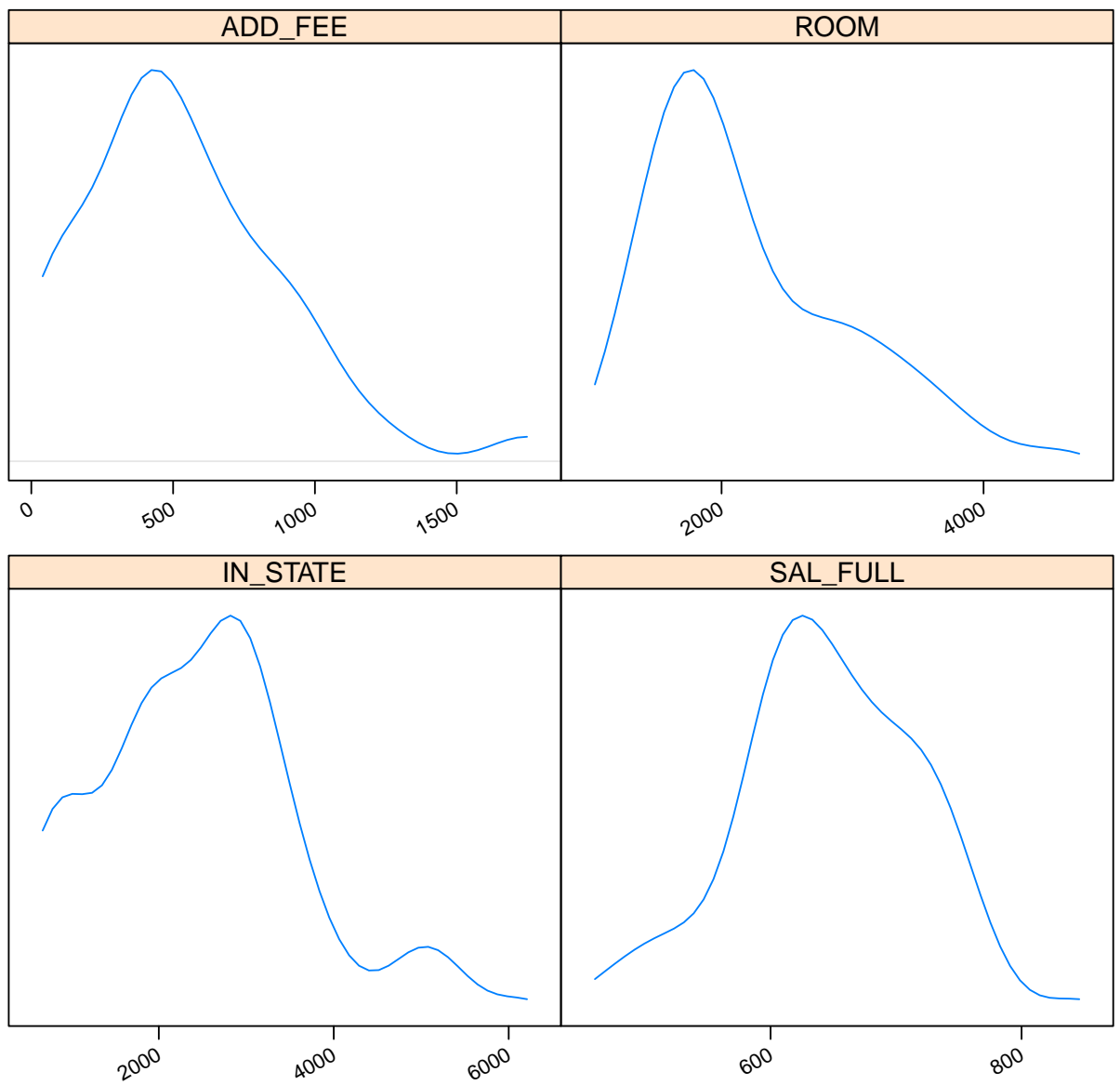
```
# Гвардия, в бой!
# latticist(df)

# На этом графике хорошо видно, что данные неоднородны.
# Нам надо будет выбрать, кого оставить ---
# частные или государственные университеты.
# Я оставлю государственные, потому что их больше.
marginal.plot(df, data = df,
              groups = PPIND, auto.key = list(lines = TRUE,
                                              title = "PPIND",
                                              cex.title = 1,
                                              columns = 2))
```

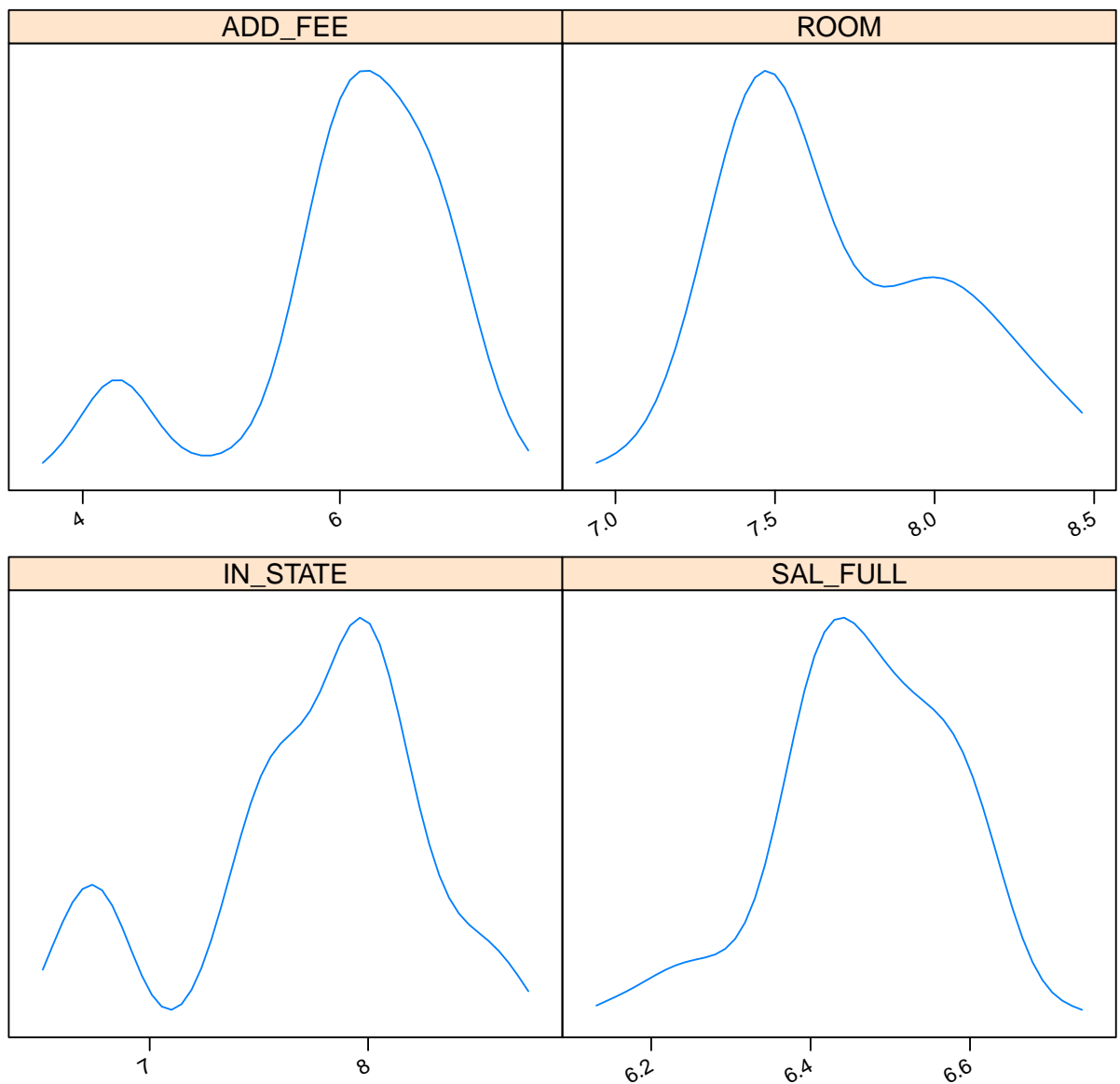


```
df.pub <- subset(df, PPIND == "Public")
# Теперь посмотрим на нормальность,
# может, что-то стоит прологарифмировать?...
# latticist(df.pub)
# Подозрение падает на денежные признаки. Они часто логнормальны.

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 мультимодальность,  
# 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
## -22.77   -3.41    1.11    3.59   15.07
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept)  -4.40e+01   8.49e+01  -0.52   0.6083
## AVRCOMB       1.47e-01   2.82e-02   5.20  1.3e-05 ***
## FULLTIME     -2.46e-04   3.80e-04  -0.65   0.5223
## log(IN_STATE) -9.06e+00   2.65e+00  -3.42   0.0018 **
## log(ROOM)     -4.01e+00   4.08e+00  -0.98   0.3338
## log(ADD_FEE)  -3.65e+00   1.63e+00  -2.24   0.0325 *
## log(SAL_FULL) 1.11e+01   1.46e+01   0.76   0.4509
## PH_D         -4.98e-01   2.84e-01  -1.75   0.0897 .
## GRADUAT       2.65e-01   1.47e-01   1.80   0.0817 .
## NUM_FULL      1.22e-02   1.17e-02   1.05   0.3043
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 7.75 on 30 degrees of freedom
## Multiple R-squared:  0.787, Adjusted R-squared:  0.724
## F-statistic: 12.3 on 9 and 30 DF,  p-value: 6.86e-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 тавтологией

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)  4.53e+01  1.13e+02   0.40  0.69069
## FULLTIME     -3.02e-04  5.15e-04  -0.59  0.56136
## log(IN_STATE) -1.31e+01  3.43e+00  -3.83  0.00059 ***
## log(ROOM)     -2.31e+00  5.52e+00  -0.42  0.67830
## log(ADD_FEE)  -4.26e+00  2.20e+00  -1.93  0.06226 .
## log(SAL_FULL) 1.67e+01  1.97e+01   0.85  0.40380
## PH_D         -2.90e-01  3.82e-01  -0.76  0.45296
## GRADUAT       7.37e-01  1.57e-01   4.70   5e-05 ***
## NUM_FULL     1.92e-02  1.58e-02   1.21  0.23418
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 10.5 on 31 degrees of freedom
## Multiple R-squared:  0.596, Adjusted R-squared:  0.491
## F-statistic: 5.71 on 8 and 31 DF, p-value: 0.000172

# 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
## -26.23  -5.18  -0.44   6.80  19.58
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  113.062    28.269   4.00  0.00030 ***
## log(IN_STATE)  -13.187     3.228  -4.09  0.00023 ***
## log(ADD_FEE)   -4.830     2.149  -2.25  0.03080 *
## GRADUAT         0.819     0.138   5.95  8.1e-07 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 10.5 on 36 degrees of freedom
## Multiple R-squared:  0.53, Adjusted R-squared:  0.491
## F-statistic: 13.5 on 3 and 36 DF, p-value: 4.56e-06
```

```

fit2.aic <- stepAIC(fit2)

## Start:  AIC=196.1
## 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 3451 194
## - FULLTIME       1      38 3470 194
## - PH_D           1      64 3496 195
## - log(SAL_FULL)  1      79 3511 195
## - NUM_FULL       1     163 3595 196
## <none>                      3432 196
## - log(ADD_FEE)   1     414 3846 199
## - log(IN_STATE)  1    1623 5055 210
## - GRADUAT        1    2450 5883 216
##
## Step:  AIC=194.3
## NEW10 ~ FULLTIME + log(IN_STATE) + log(ADD_FEE) + log(SAL_FULL) +
##      PH_D + GRADUAT + NUM_FULL
##
##              Df Sum of Sq  RSS AIC
## - FULLTIME       1      37 3488 193
## - log(SAL_FULL)  1      60 3511 193
## - PH_D           1      61 3513 193
## <none>                      3451 194
## - NUM_FULL       1     194 3645 194
## - log(ADD_FEE)   1     451 3903 197
## - log(IN_STATE)  1    1729 5181 209
## - GRADUAT        1    2629 6081 215
##
## Step:  AIC=192.7
## NEW10 ~ log(IN_STATE) + log(ADD_FEE) + log(SAL_FULL) + PH_D +
##      GRADUAT + NUM_FULL
##
##              Df Sum of Sq  RSS AIC
## - PH_D           1      43 3531 191
## - log(SAL_FULL)  1      65 3554 192
## <none>                      3488 193
## - NUM_FULL       1     250 3739 194
## - log(ADD_FEE)   1     433 3921 195
## - log(IN_STATE)  1    1702 5190 207
## - GRADUAT        1    2592 6081 213
##
## Step:  AIC=191.2
## NEW10 ~ log(IN_STATE) + log(ADD_FEE) + log(SAL_FULL) + GRADUAT +
##      NUM_FULL
##

```

```

##           Df Sum of Sq  RSS AIC
## - log(SAL_FULL)  1          60 3591 190
## <none>                                3531 191
## - NUM_FULL      1          212 3743 192
## - log(ADD_FEE)   1          458 3989 194
## - log(IN_STATE)  1         1724 5255 205
## - GRADUAT       1         2551 6082 211
##
## Step:  AIC=189.9
## NEW10 ~ log(IN_STATE) + log(ADD_FEE) + GRADUAT + NUM_FULL
##
##           Df Sum of Sq  RSS AIC
## <none>                                3591 190
## - NUM_FULL      1          397 3988 192
## - log(ADD_FEE)   1          446 4037 193
## - log(IN_STATE)  1         1664 5255 203
## - GRADUAT       1         3291 6882 214

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
## -22.987  -5.665  -0.508   5.308  23.992
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept)  102.45261    27.73302     3.69  0.00075 ***
## log(IN_STATE) -12.57406     3.12200    -4.03  0.00029 ***
## log(ADD_FEE)  -4.34406     2.08263    -2.09  0.04435 *
## GRADUAT       0.76556     0.13517     5.66  2.1e-06 ***
## NUM_FULL      0.01423     0.00723     1.97  0.05702 .
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 10.1 on 35 degrees of freedom
## Multiple R-squared:  0.577, Adjusted R-squared:  0.528
## F-statistic: 11.9 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  5 307.6
## fit2.aic     6 305.4

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

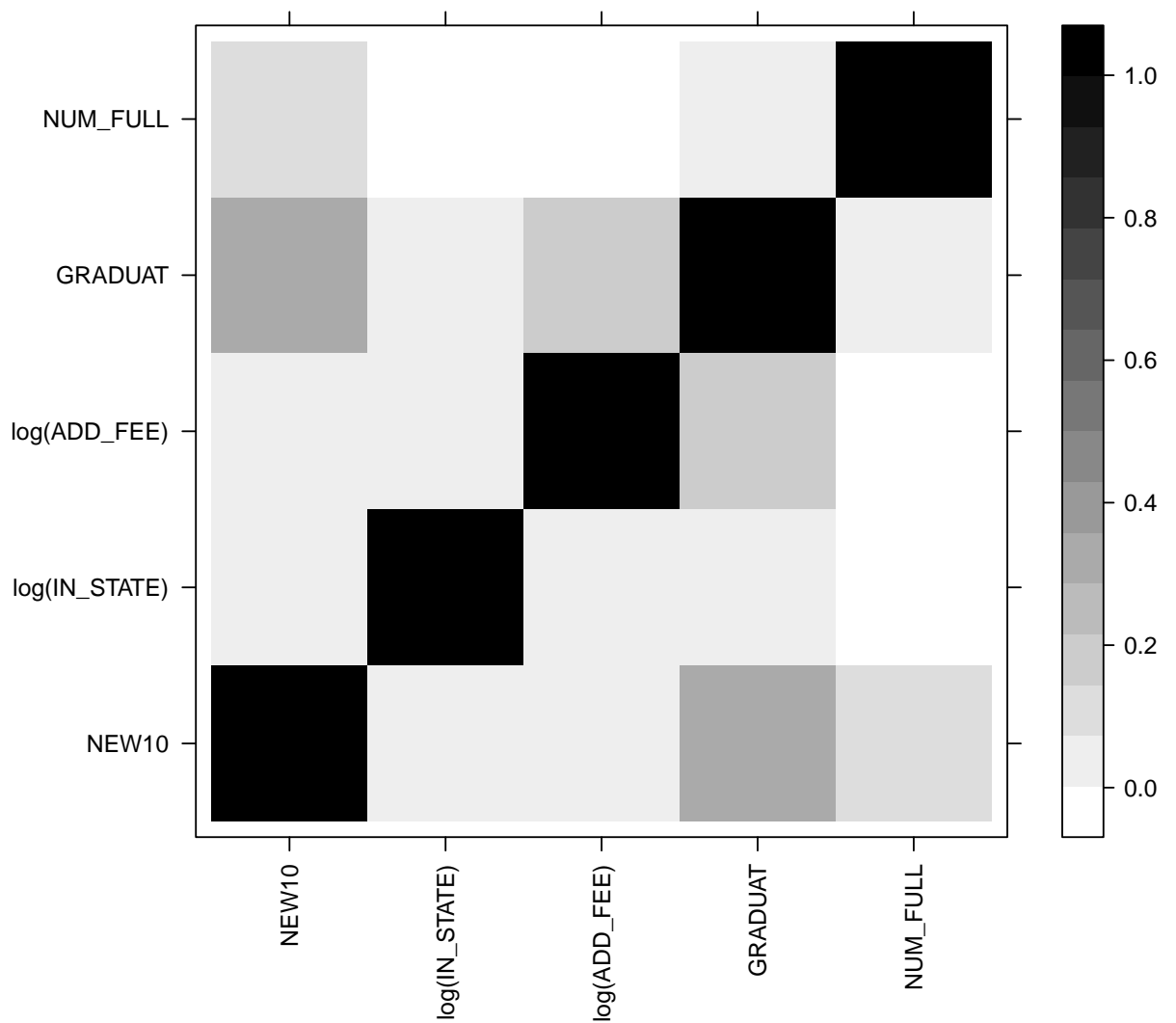
anova(fit2.manual, fit2.aic)

## 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
## 2       35 3591  1       397 3.87 0.057 .
## ---
## 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) GRADUAT
## NEW10          1.0000      -0.25001      0.11170  0.5450
## log(IN_STATE) -0.2500      1.00000     -0.15425  0.2645
## log(ADD_FEE)  0.1117     -0.15425      1.00000  0.4106
## GRADUAT        0.5450      0.26450      0.41065  1.0000
## NUM_FULL       0.3517     -0.02384     -0.02152  0.1516
##          NUM_FULL
## NEW10          0.35167
## log(IN_STATE) -0.02384
## log(ADD_FEE)  -0.02152
## GRADUAT        0.15156
## NUM_FULL       1.00000

levelplot(cor(fit2.aic$model)^2,
           par.settings = list(regions = list(col = colorRampPalette(grey(1:0)))),
           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
## -22.987  -5.665  -0.508   5.308  23.992
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept)  102.45261    27.73302     3.69  0.00075 ***
## log(IN_STATE) -12.57406     3.12200    -4.03  0.00029 ***
```

```

## log(ADD_FEE)    -4.34406    2.08263    -2.09    0.04435 *
## GRADUAT        0.76556    0.13517    5.66    2.1e-06 ***
## NUM_FULL       0.01423    0.00723    1.97    0.05702 .
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 10.1 on 35 degrees of freedom
## Multiple R-squared:  0.577, Adjusted R-squared:  0.528
## F-statistic: 11.9 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: 125.4

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

##
## Error estimation of 'lm' using 35-fold cross validation: 145.1

# 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
## -24.091  -4.681   0.547   3.576  24.793
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept) -1.86698     7.50975  -0.25  0.80512
## IN_STATE    -0.00528     0.00142  -3.73  0.00069 ***

```

```
## ADD_FEE      -0.00599    0.00549   -1.09  0.28269
## GRADUAT      0.72609    0.14410    5.04  1.4e-05 ***
## NUM_FULL     0.01533    0.00746    2.06  0.04738 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 10.4 on 35 degrees of freedom
## Multiple R-squared:  0.55, Adjusted R-squared:  0.498
## F-statistic: 10.7 on 4 and 35 DF,  p-value: 9.13e-06

AIC(fit.nolog, fit2.aic)

##           df    AIC
## fit.nolog  6 307.9
## fit2.aic   6 305.4

# 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          0 170 -20.4
## <none>                        170 -18.6
## - TV:Newspaper      1          4 174 -15.5
## - TV:Radio          1        350 520 203.0
##
## Step:  AIC=-20.36
## Sales ~ TV + Radio + Newspaper + TV:Radio + TV:Newspaper
```

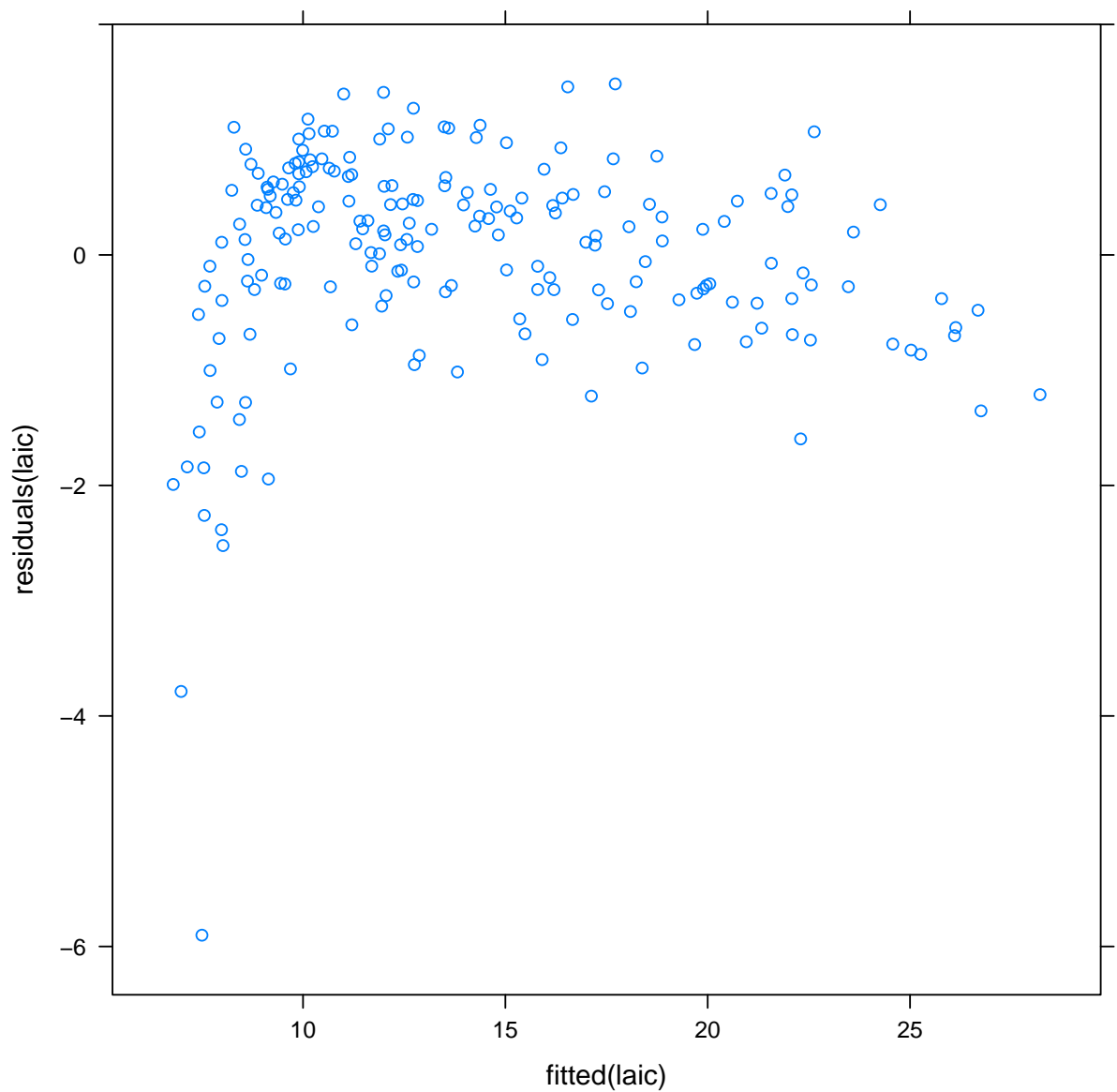
```
##
##              Df Sum of Sq RSS    AIC
## <none>              170 -20.4
## - TV:Newspaper    1         4 174 -17.5
## - TV:Radio        1       353 523 202.2

summary(laic)

##
## Call:
## lm(formula = Sales ~ TV + Radio + Newspaper + TV:Radio + TV:Newspaper,
##     data = Advertising)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -5.902 -0.382  0.194  0.574  1.484
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept)  6.54e+00   2.65e-01   24.67  <2e-16 ***
## TV           2.03e-02   1.61e-03   12.67  <2e-16 ***
## Radio        2.02e-02   9.73e-03    2.07   0.039 *
## Newspaper    1.34e-02   6.38e-03    2.10   0.037 *
## TV:Radio      1.14e-03   5.66e-05   20.06  <2e-16 ***
## TV:Newspaper -7.72e-05   3.53e-05   -2.19   0.030 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.936 on 194 degrees of freedom
## Multiple R-squared:  0.969, Adjusted R-squared:  0.968
## F-statistic: 1.2e+03 on 5 and 194 DF,  p-value: <2e-16

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



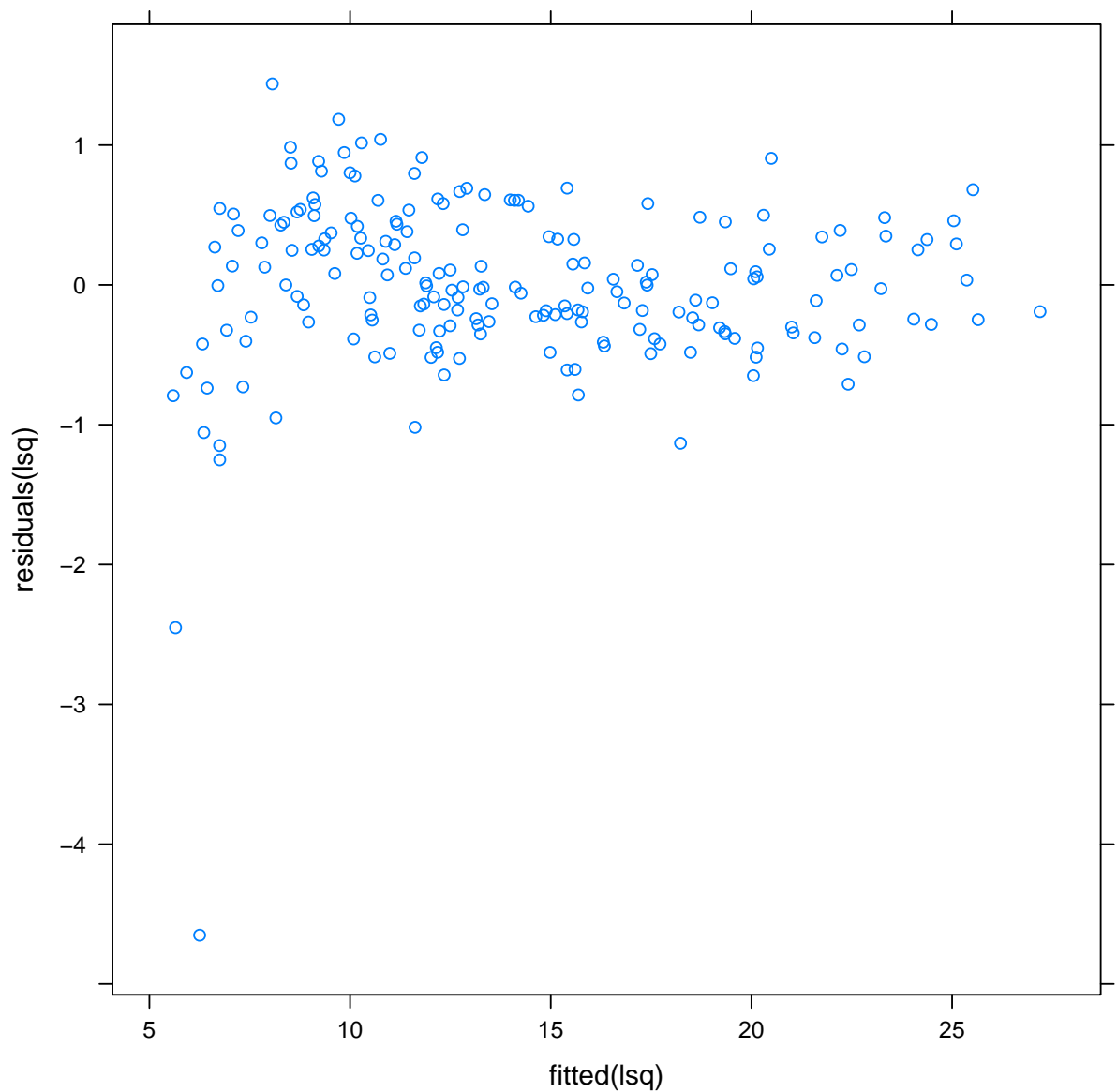


```
lsq <- lm(Sales ~ poly(TV, Radio, Newspaper, degree = 2), data = Advertising)
summary(lsq)
```

```
##
## Call:
## lm(formula = Sales ~ poly(TV, Radio, Newspaper, degree = 2),
##     data = Advertising)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -4.651  -0.294  -0.006   0.383   1.438
##
## Coefficients:
##                                     Estimate Std. Error
## (Intercept)                        13.9410     0.0481
```

```
## poly(TV, Radio, Newspaper, degree = 2)1.0.0  53.7303      0.6258
## poly(TV, Radio, Newspaper, degree = 2)2.0.0  -9.9799      0.6325
## poly(TV, Radio, Newspaper, degree = 2)0.1.0  40.1048      0.6704
## poly(TV, Radio, Newspaper, degree = 2)1.1.0 280.3575      9.6532
## poly(TV, Radio, Newspaper, degree = 2)0.2.0   0.2973      0.6486
## poly(TV, Radio, Newspaper, degree = 2)0.0.1   0.9437      0.7414
## poly(TV, Radio, Newspaper, degree = 2)1.0.1 -16.9358      8.8441
## poly(TV, Radio, Newspaper, degree = 2)0.1.1   5.3188     10.9533
## poly(TV, Radio, Newspaper, degree = 2)0.0.2   0.1071      0.6600
##
##                                     t value Pr(>|t|)
## (Intercept)                        290.08   <2e-16 ***
## poly(TV, Radio, Newspaper, degree = 2)1.0.0   85.86   <2e-16 ***
## poly(TV, Radio, Newspaper, degree = 2)2.0.0  -15.78   <2e-16 ***
## poly(TV, Radio, Newspaper, degree = 2)0.1.0   59.82   <2e-16 ***
## poly(TV, Radio, Newspaper, degree = 2)1.1.0   29.04   <2e-16 ***
## poly(TV, Radio, Newspaper, degree = 2)0.2.0    0.46    0.647
## poly(TV, Radio, Newspaper, degree = 2)0.0.1    1.27    0.205
## poly(TV, Radio, Newspaper, degree = 2)1.0.1   -1.91    0.057 .
## poly(TV, Radio, Newspaper, degree = 2)0.1.1    0.49    0.628
## poly(TV, Radio, Newspaper, degree = 2)0.0.2    0.16    0.871
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.62 on 190 degrees of freedom
## Multiple R-squared:  0.987, Adjusted R-squared:  0.986
## F-statistic: 1.54e+03 on 9 and 190 DF,  p-value: <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  RSS  AIC
## <none>                                73 -181
## - poly(TV, Radio, Newspaper, degree = 2)  9      5344 5417  662

summary(lsqaic)

##
## Call:
## lm(formula = Sales ~ poly(TV, Radio, Newspaper, degree = 2),
##     data = Advertising)
```

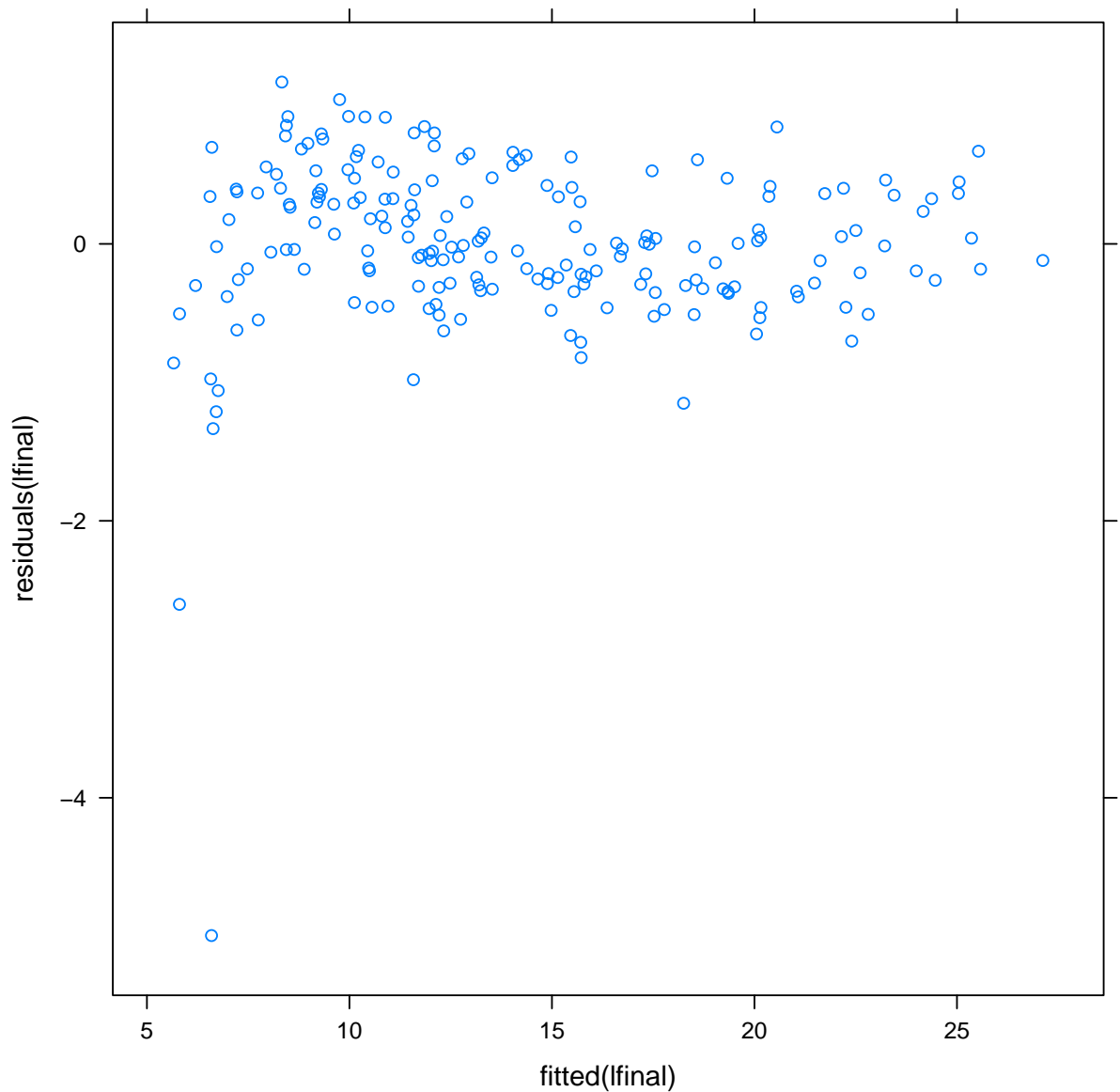
```
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -4.651 -0.294 -0.006  0.383  1.438
##
## Coefficients:
##                                Estimate Std. Error
## (Intercept)                   13.9410     0.0481
## poly(TV, Radio, Newspaper, degree = 2)1.0.0  53.7303     0.6258
## poly(TV, Radio, Newspaper, degree = 2)2.0.0  -9.9799     0.6325
## poly(TV, Radio, Newspaper, degree = 2)0.1.0  40.1048     0.6704
## poly(TV, Radio, Newspaper, degree = 2)1.1.0 280.3575     9.6532
## poly(TV, Radio, Newspaper, degree = 2)0.2.0   0.2973     0.6486
## poly(TV, Radio, Newspaper, degree = 2)0.0.1   0.9437     0.7414
## poly(TV, Radio, Newspaper, degree = 2)1.0.1 -16.9358     8.8441
## poly(TV, Radio, Newspaper, degree = 2)0.1.1   5.3188    10.9533
## poly(TV, Radio, Newspaper, degree = 2)0.0.2   0.1071     0.6600
##                                t value Pr(>|t|)
## (Intercept)                   290.08   <2e-16 ***
## poly(TV, Radio, Newspaper, degree = 2)1.0.0   85.86   <2e-16 ***
## poly(TV, Radio, Newspaper, degree = 2)2.0.0  -15.78   <2e-16 ***
## poly(TV, Radio, Newspaper, degree = 2)0.1.0   59.82   <2e-16 ***
## poly(TV, Radio, Newspaper, degree = 2)1.1.0   29.04   <2e-16 ***
## poly(TV, Radio, Newspaper, degree = 2)0.2.0    0.46    0.647
## poly(TV, Radio, Newspaper, degree = 2)0.0.1    1.27    0.205
## poly(TV, Radio, Newspaper, degree = 2)1.0.1   -1.91    0.057 .
## poly(TV, Radio, Newspaper, degree = 2)0.1.1    0.49    0.628
## poly(TV, Radio, Newspaper, degree = 2)0.0.2    0.16    0.871
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.62 on 190 degrees of freedom
## Multiple R-squared:  0.987, Adjusted R-squared:  0.986
## F-statistic: 1.54e+03 on 9 and 190 DF, p-value: <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
## -4.995 -0.297 -0.007  0.380  1.169
##
## Coefficients:
##                                Estimate Std. Error t value Pr(>|t|)
```

```
## (Intercept)  5.14e+00  1.93e-01  26.66 < 2e-16 ***
## TV          5.09e-02  2.23e-03  22.81 < 2e-16 ***
## Radio       3.52e-02  5.90e-03   5.96 1.2e-08 ***
## I(TV^2)     -1.10e-04  6.89e-06 -15.92 < 2e-16 ***
## TV:Radio    1.08e-03  3.47e-05  31.06 < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.624 on 195 degrees of freedom
## Multiple R-squared:  0.986, Adjusted R-squared:  0.986
## F-statistic: 3.43e+03 on 4 and 195 DF,  p-value: <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: 0.1662

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

##
## Error estimation of 'lm' using fixed training/validation set: 3.069

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

##
## Error estimation of 'lm' using fixed training/validation set: 1.704

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

##
## Error estimation of 'lm' using fixed training/validation set: 0.6504

```

## 5 Материалы с занятия 10 октября

Вспомогательный код для классификации

```

library(MASS) # AIC(), BIC(), lda(), qda()
library(lattice) # xyplot(), densityplot()
library(latticeExtra) # layer()
library(ROCR) # performance(), prediction()
# 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
##      0.3333      0.3333      0.3333
##
## Group means:
##              Sepal.Length Sepal.Width Petal.Length Petal.Width
## setosa              5.006      3.428      1.462      0.246
## versicolor          5.936      2.770      4.260      1.326
## virginica           6.588      2.974      5.552      2.026
##
## Coefficients of linear discriminants:
##              LD1      LD2
## Sepal.Length  0.8294  0.0241
## Sepal.Width   1.5345  2.1645
## Petal.Length -2.2012 -0.9319
## Petal.Width  -2.8105  2.8392
##
## 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
##      0.3333      0.3333      0.3333
```

```
##
## Group means:
##           Sepal.Length Sepal.Width Petal.Length Petal.Width
## setosa           4.985         3.408         1.481         0.2192
## versicolor       6.016         2.803         4.375         1.3562
## virginica        6.522         2.959         5.513         2.0187
##
## Coefficients of linear discriminants:
##           LD1      LD2
## Sepal.Length  1.070  0.1976
## Sepal.Width   1.259 -2.0364
## Petal.Length -2.402  0.9645
## Petal.Width  -3.077 -2.9905
##
## Proportion of trace:
##      LD1      LD2
## 0.9938 0.0062

table(predicted = predict(ld, iris.test)$class,
      actual = iris.test$Species)

##           actual
## predicted   setosa versicolor virginica
## setosa      24         0         0
## versicolor   0        17         1
## virginica    0         1        17

table(predicted = predict(ld, iris.test, prior = c(0.5, 0.25, 0.25))$class,
      actual = iris.test$Species)

##           actual
## predicted   setosa versicolor virginica
## setosa      24         0         0
## versicolor   0        17         1
## virginica    0         1        17

cm <- table(predicted = predict(ld, iris.test)$class,
      actual = iris.test$Species)
chisq.test(cm)

##
## Pearson's Chi-squared test
##
## data:  cm
## X-squared = 107.4, df = 4, p-value < 2.2e-16

# test-train
tune(lda, Species ~ ., prior = c(1/3, 1/3, 1/3),
      data = iris,
```

```

predict.func = function(...) predict(...)$class,
tunecontrol = tune.control(sampling = "fix",
                           fix = 2/3))

##
## Error estimation of 'lda' using fixed training/validation set: 0.02

simple.predict.da <- function(...) predict(...)$class

# bootstrap
tune(lda, Species ~ ., prior = c(1/3, 1/3, 1/3),
     data = iris,
     predict.func = simple.predict.da,
     tunecontrol = tune.control(sampling = "bootstrap",
                               nboot = 10,
                               boot.size = 9/10))

##
## Error estimation of 'lda' using bootstrapping: 0.02318

# 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
##      0.3333      0.3333      0.3333
##
## Group means:

```

```
##           Sepal.Length Sepal.Width Petal.Length Petal.Width
## setosa           5.006           3.428           1.462           0.246
## versicolor       5.936           2.770           4.260           1.326
## virginica        6.588           2.974           5.552           2.026
##
## Coefficients of linear discriminants:
##           LD1      LD2
## Sepal.Length  0.8294  0.0241
## Sepal.Width   1.5345  2.1645
## Petal.Length -2.2012 -0.9319
## Petal.Width  -2.8105  2.8392
##
## Proportion of trace:
##      LD1      LD2
## 0.9912 0.0088

tn$performance

##      dummyparameter error dispersion
## 1              0  0.02      0.0322

# 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 Petal.Length
## versicolor          18.69         -5.458         -8.707          14.24
## virginica          -23.84         -7.924         -15.371          23.66
##           Petal.Width
## versicolor         -3.098
## virginica          15.135
##
## Std. Errors:
##           (Intercept) Sepal.Length Sepal.Width Petal.Length
## versicolor          34.97          89.89          157.0          60.19
## virginica           35.77          89.91          157.1          60.47
```

```

##              Petal.Width
## versicolor      45.49
## virginica       45.93
##
## Residual Deviance: 11.9
## AIC: 31.9

mln.aic <- stepAIC(mln)

## Start:  AIC=31.9
## Species ~ Sepal.Length + Sepal.Width + Petal.Length + Petal.Width
##
##              Df  AIC
## - Sepal.Length  2 29.3
## - Sepal.Width   2 31.5
## <none>          31.9
## - Petal.Width   2 39.8
## - Petal.Length  2 41.9
##
## Step:  AIC=29.27
## Species ~ Sepal.Width + Petal.Length + Petal.Width
##
##              Df  AIC
## <none>          29.3
## - Sepal.Width   2 32.6
## - Petal.Length  2 39.4
## - Petal.Width   2 43.5

summary(mln.aic)

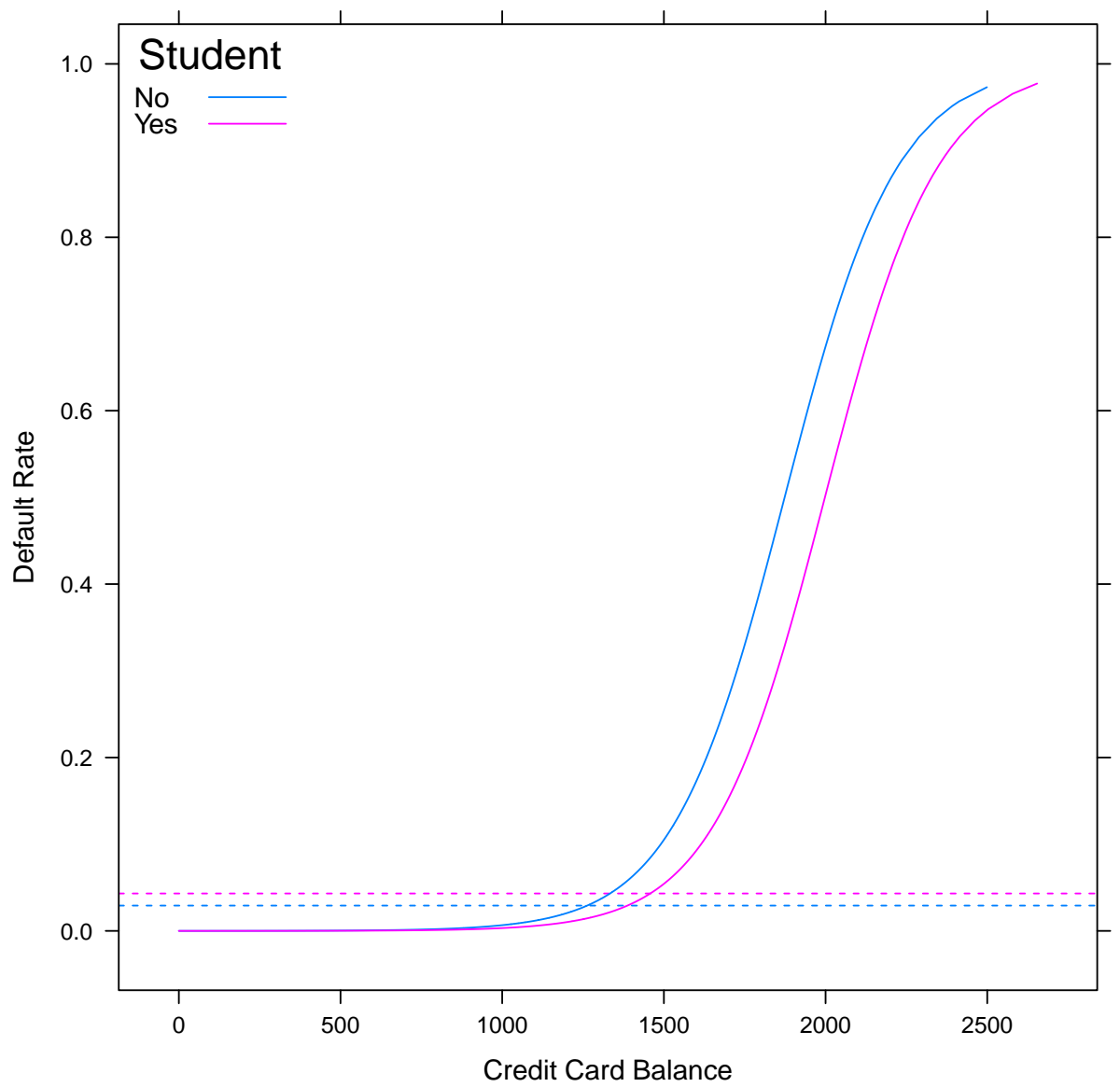
## Call:
## multinom(formula = Species ~ Sepal.Width + Petal.Length + Petal.Width,
##          data = iris, trace = FALSE)
##
## Coefficients:
##              (Intercept) Sepal.Width Petal.Length Petal.Width
## versicolor      14.16      -17.32         14.10      -2.696
## virginica       -36.44      -25.71         21.98      18.766
##
## Std. Errors:
##              (Intercept) Sepal.Width Petal.Length Petal.Width
## versicolor      29.66       47.48         68.58       39.08
## virginica       32.19       48.00         68.77       39.75
##
## Residual Deviance: 13.27
## AIC: 29.27

```

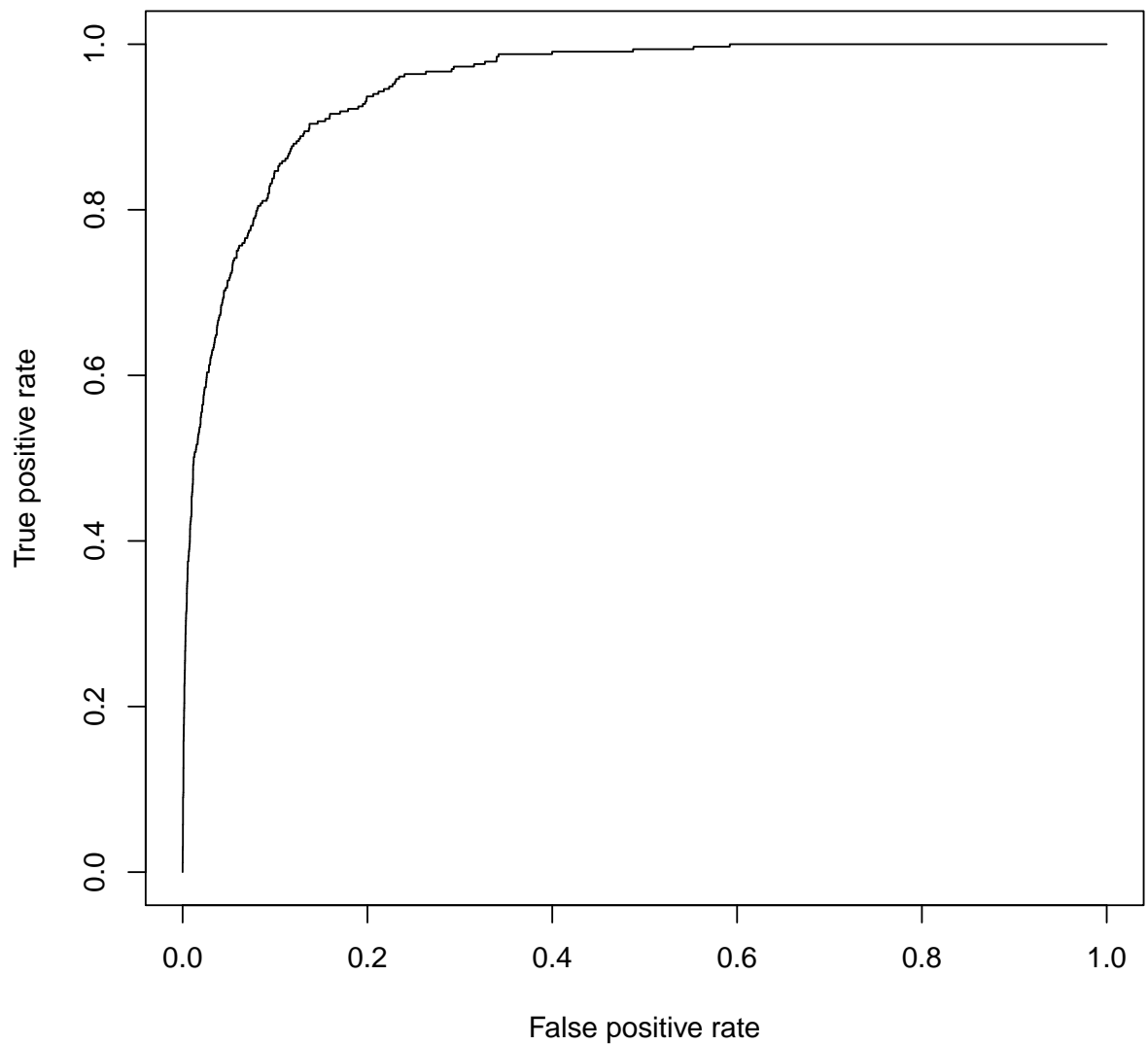
## 5.2 Default

```
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(gl, Default), Default$default)

## [1] 0.9495

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: -0.9488
```



```

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

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

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

##           actual
## predicted   No  Yes
##        No 8340   33
##        Yes 1327  300

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

##           actual
## predicted   No  Yes
##        No 9613  209
##        Yes   54  124

train <- sample(nrow(Default), size = 0.6 * nrow(Default))
gl <- glm(default ~ balance + student, data = Default, subset = train,
          family = binomial(link = "logit"))

table(predicted = my.predict.glm(gl, Default[-train, ], measure = "max"),
      actual = Default[-train, ]$default)

##           actual
## predicted   No  Yes
##        No 3434   15
##        Yes  429  122

table(predicted = my.predict.glm(gl, Default[-train, ], measure = "mean"),
      actual = Default[-train, ]$default)

##           actual
## predicted   No  Yes
##        No 3434   15
##        Yes  429  122

table(predicted = my.predict.glm(gl, Default[-train, ], measure = "err"),
      actual = Default[-train, ]$default)

##           actual
## predicted   No  Yes
##        No 3836   84
##        Yes   27  53

```

## 5.3 Smarket

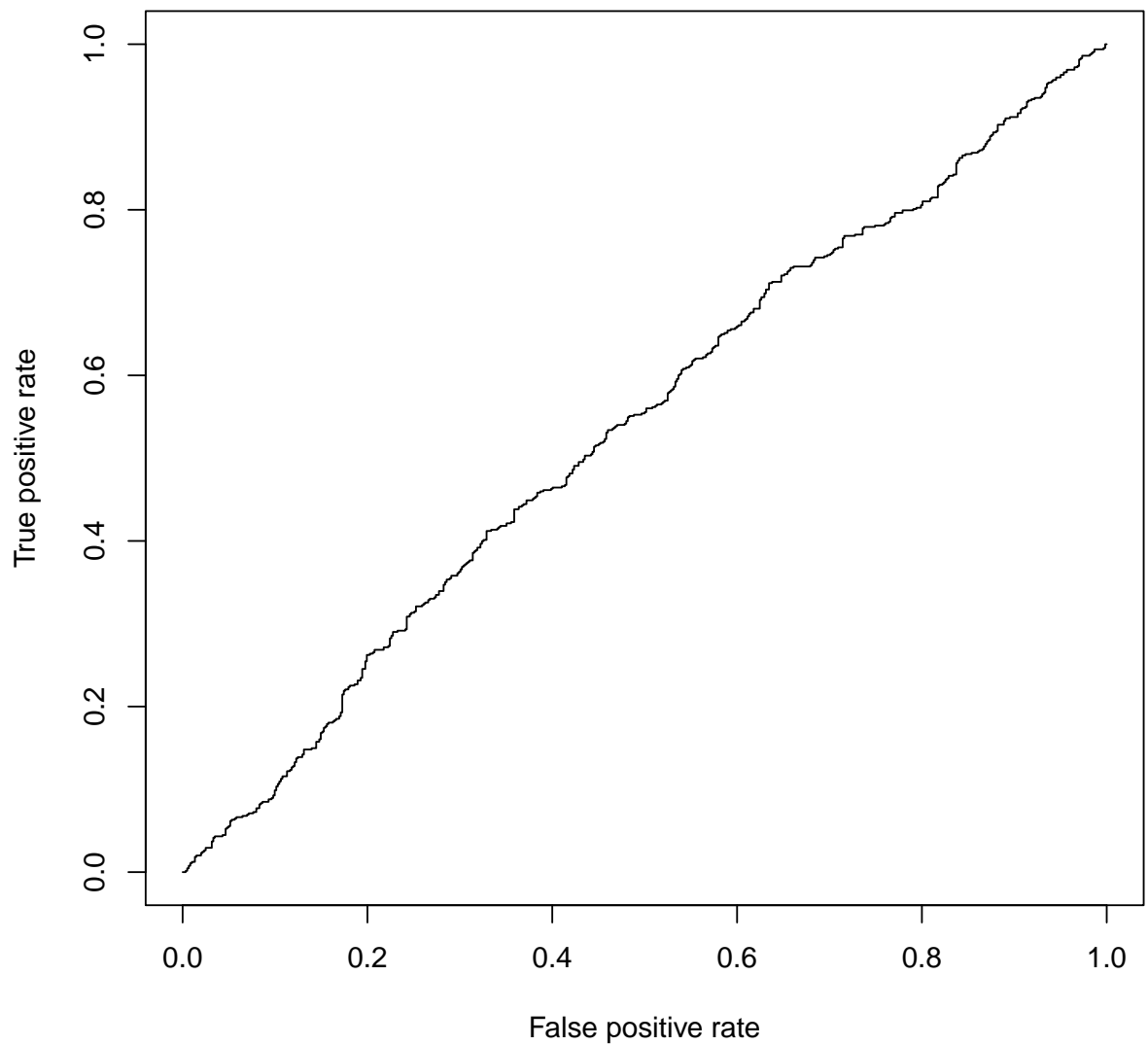
```
library(ISLR)
library(MASS)

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
## -1.45    -1.20     1.07     1.15     1.33
##
## Coefficients:
##              Estimate Std. Error z value Pr(>|z|)
## (Intercept) -0.12600    0.24074  -0.52    0.60
## Volume       0.13544    0.15836   0.86    0.39
## Lag1        -0.07307    0.05017  -1.46    0.15
## Lag2        -0.04230    0.05009  -0.84    0.40
## Lag3         0.01109    0.04994   0.22    0.82
## Lag4         0.00936    0.04997   0.19    0.85
## Lag5         0.01031    0.04951   0.21    0.83
##
## (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: 1742
##
## Number of Fisher Scoring iterations: 3

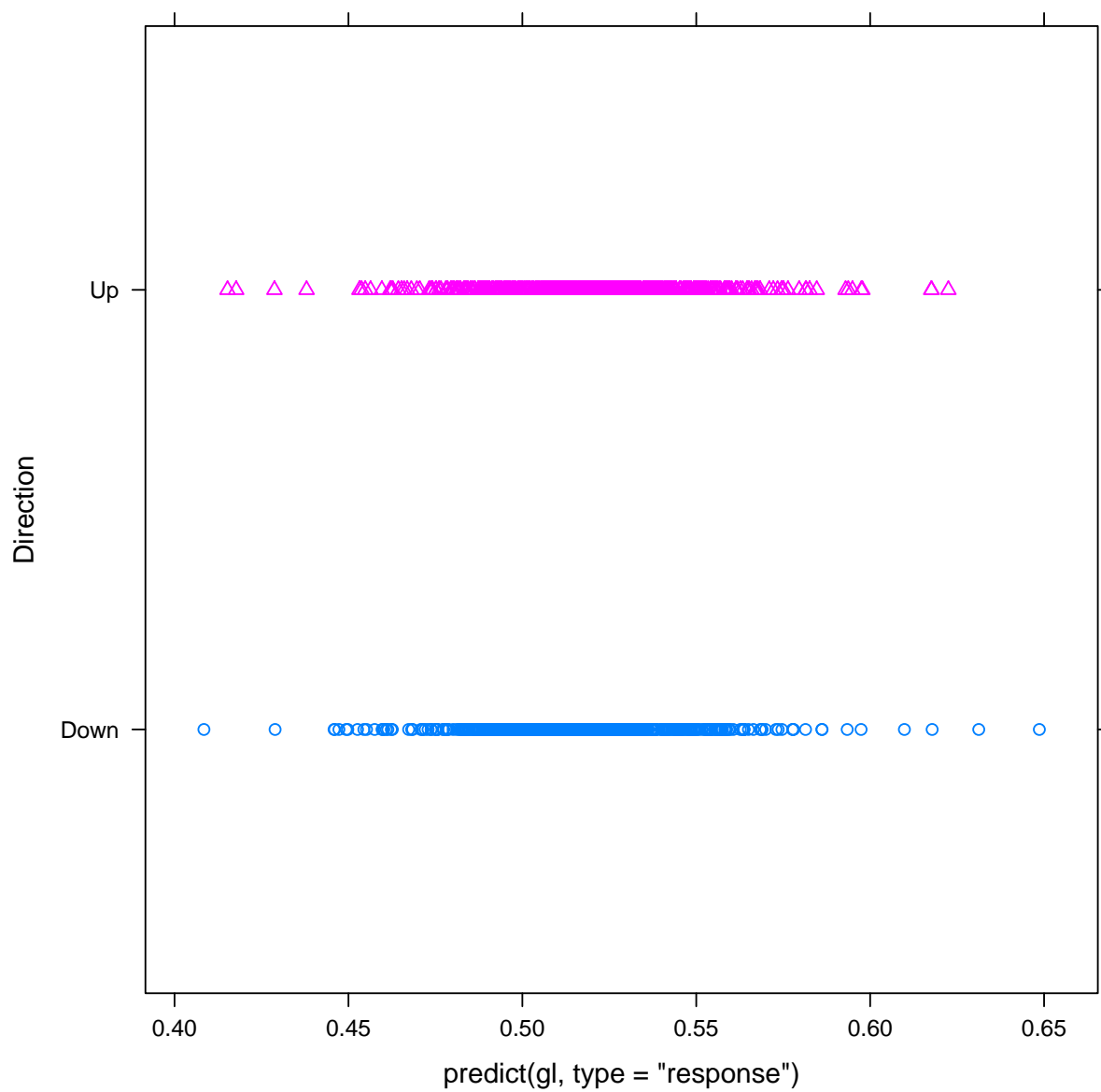
source("class.R")
roc <- ROC(predict(gl), Smarket$Direction)
plot(roc)
```



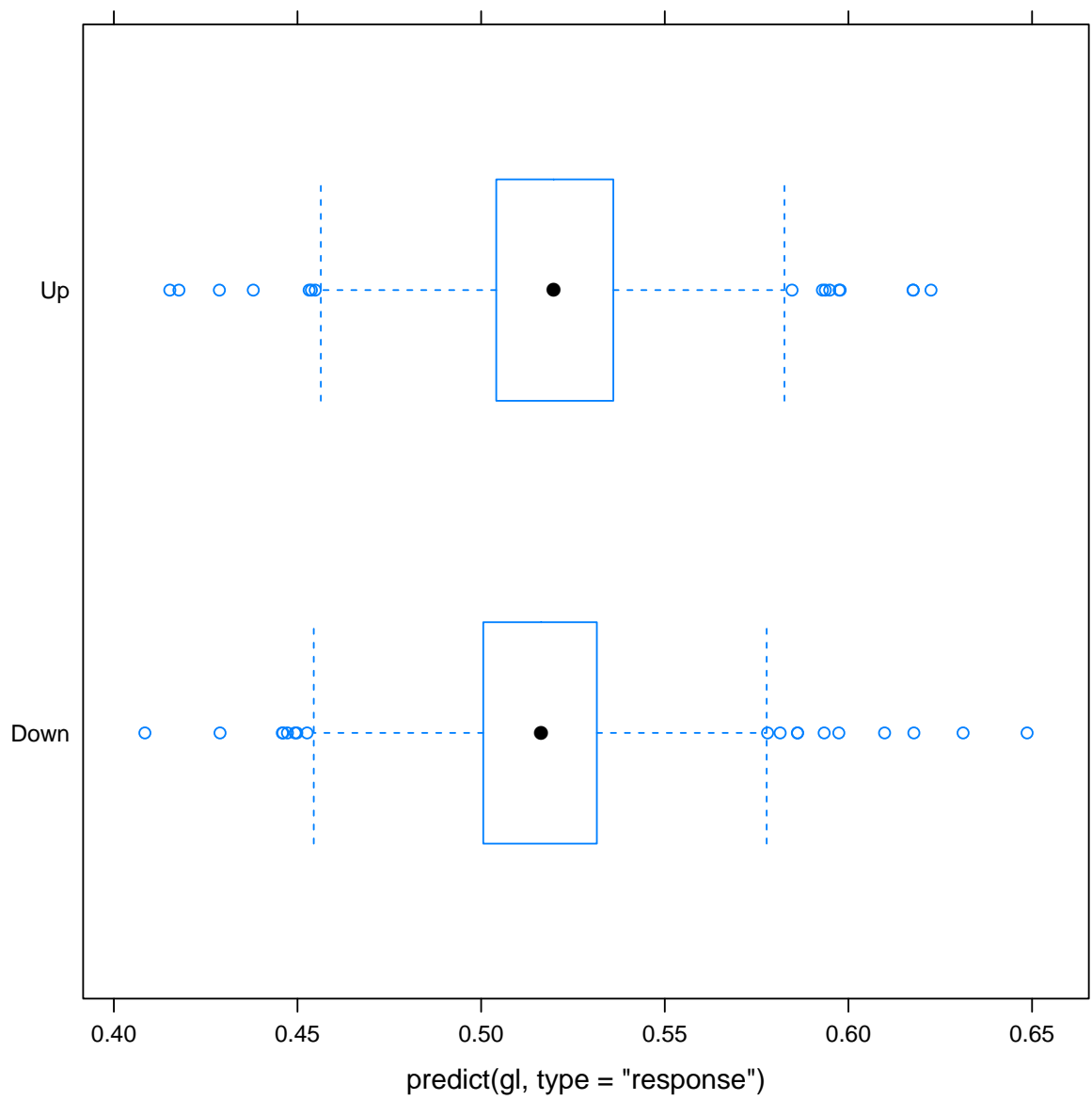
```
AUC(predict(g1), Smarket$Direction)

## [1] 0.5387

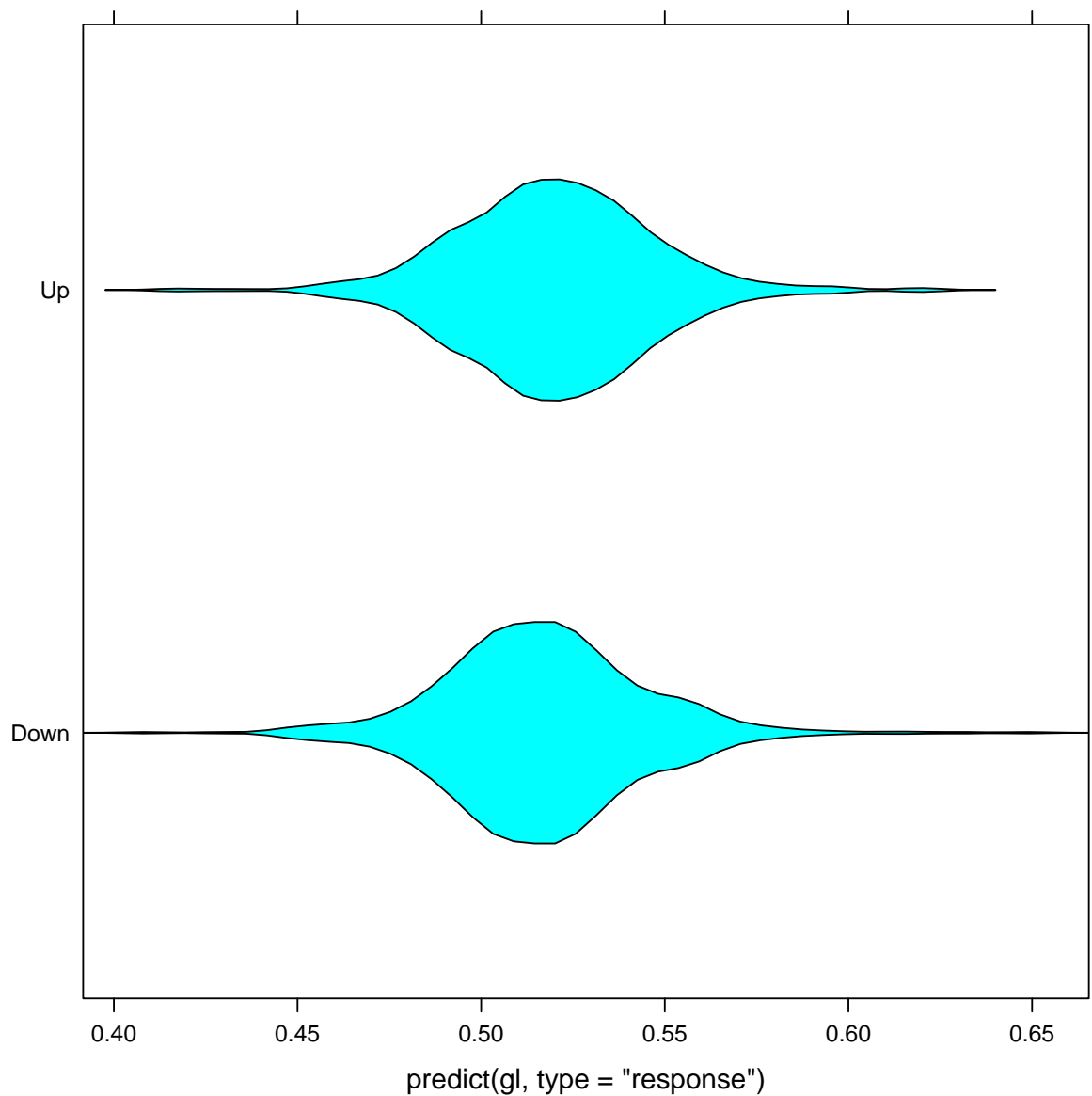
xyplot(Direction ~ predict(g1, type = "response"),
       data = Smarket,
       groups = Direction,
       par.settings = simpleTheme(pch = 1:2))
```



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



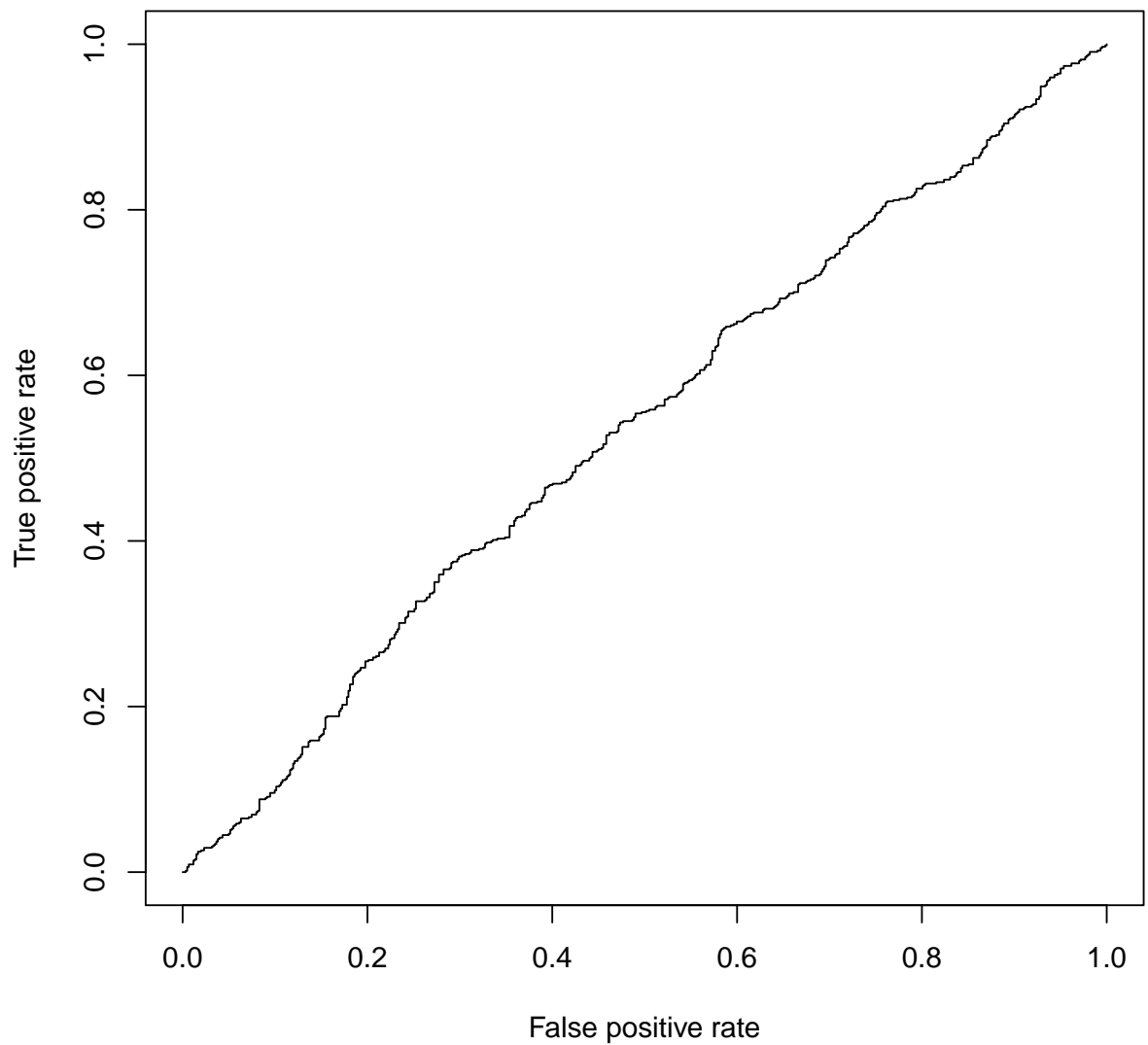
```
bwplot(Direction ~ predict(gl, type = "response"),
       data = Smarket,
       panel = panel.violin)
```



```
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   
## -1.45    -1.20     1.07     1.15     1.33   
##  
## Coefficients:
```

```
##           Estimate Std. Error z value Pr(>|z|)
## (Intercept) -0.1206    0.2402  -0.50    0.62
## Volume      0.1318    0.1580   0.83    0.40
## Lag1        -0.0733    0.0502  -1.46    0.14
## Lag2        -0.0428    0.0501  -0.85    0.39
##
## (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: 1736
##
## Number of Fisher Scoring iterations: 3

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

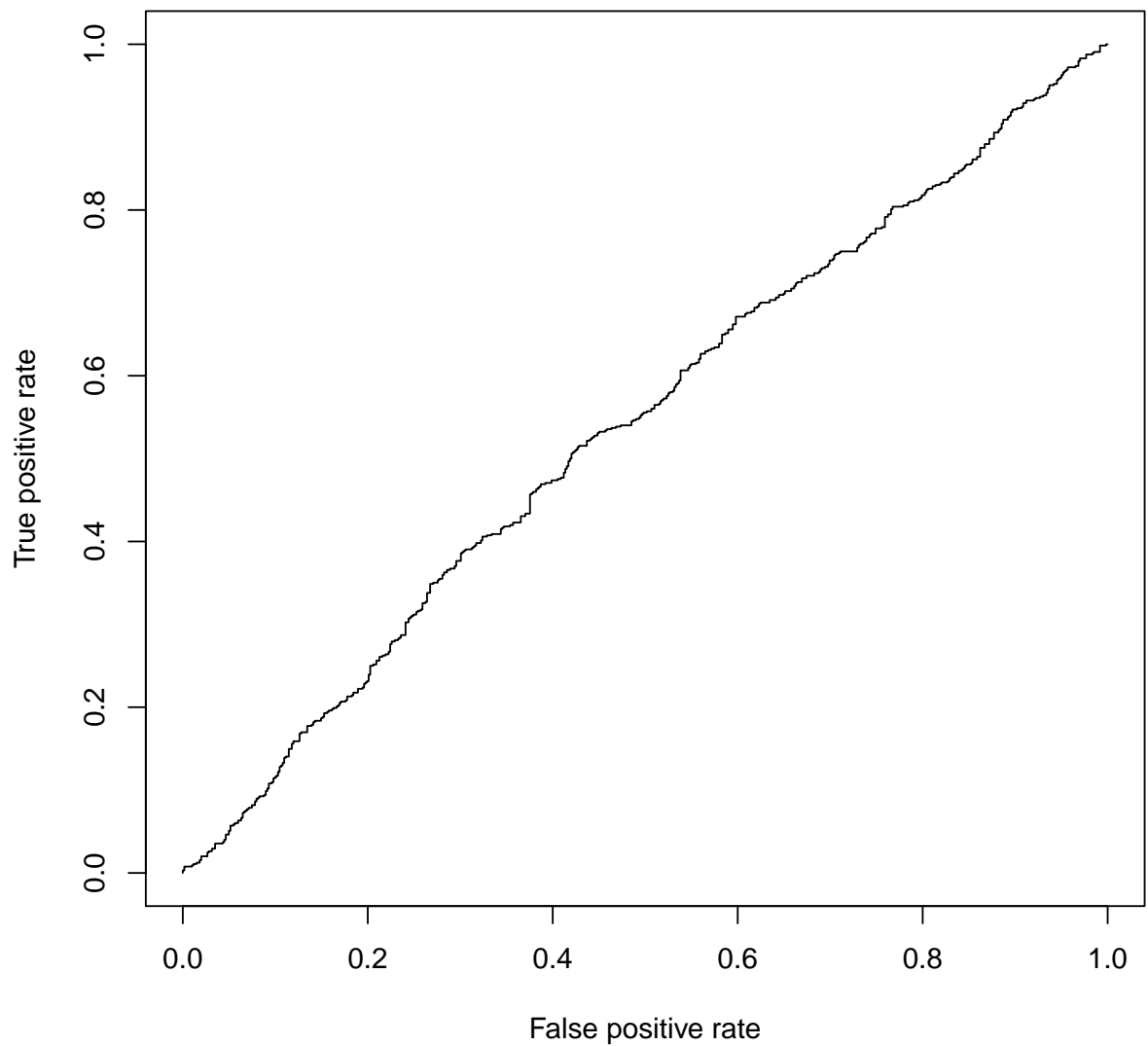


```
AUC(predict(g1), Smarket$Direction)

## [1] 0.5371

g1 <- glm(Direction ~ poly(Volume, Lag1, Lag2, degree = 2),
  data = Smarket,
  family = binomial(link = "logit"))
roc <- ROC(predict(g1), Smarket$Direction)
plot(roc)
```





```
AUC(predict(gl), Smarket$Direction)

## [1] 0.5401

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: 0.4886

tn.mglm

##
## Error estimation of 'glm' using 100-fold cross validation: 0.4944

tn.mglm$performances$dispersion

## [1] 0.1436

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: 0.5047

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 = 5.659, df = 1, p-value = 0.01737

```

## 5.4 banknote

```

source("class.R")

banknote <- read.csv("banknote/data_banknote_authentication.txt",
                    header = FALSE)
colnames(banknote) <- c("variance",
                        "skewness",
                        "curtosis",
                        "entropy",
                        "class")

## Error: 'names' attribute [5] must be the same length as the vector [1]

banknote$class <- factor(banknote$class, labels = c("genuine", "forged"))

## Error: invalid 'labels'; length 2 should be 1 or 0

summary(banknote)

##           V1
##  0           : 762
##  1           : 610
## -4.4552:     6
## -0.2957:     5
##  0.5706 :     5
## -0.9888:     5
## (Other):5495

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

## Error: object is not a matrix

ld <- lda(class ~ ., data = banknote)

## Error: object is not a matrix

qd <- qda(class ~ ., data = banknote)

## Error: object is not a matrix

gl <- glm(class ~ ., data = banknote, family = binomial(link = "logit"))

## Error: object is not a matrix

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

## Error: all arguments must have the same length

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

## Error: object is not a matrix

tn

```

```
##
## Error estimation of 'lda' using 10-fold cross validation: 0.02

tn$performances$dispersion

## [1] 0.0322

tn.glm <- tune(glm, class ~ . , data = banknote,
               family = binomial(link = "logit"),
               predict.func = simple.predict.glm)

## Error: object is not a matrix

tn.lda <- tune(lda, class ~ . , data = banknote,
               predict.func = simple.predict.da)

## Error: object is not a matrix

tn.qda <- tune(qda, class ~ . , data = banknote,
               predict.func = simple.predict.da)

## Error: object is not a matrix

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

## Error: object is not a matrix

tn.mglm <- tune(glm, class ~ . , data = banknote,
               family = binomial(link = "logit"),
               predict.func = function(...)
                 my.predict.glm(..., measure = "err"))

## Error: object is not a matrix

tn.mlda <- tune(my.lda, class ~ . , data = banknote,
               predict.func = function(...)
                 my.predict.da(..., measure = "err"))

## Error: object is not a matrix

tn.mqda <- tune(my.qda, class ~ . , data = banknote,
               predict.func = function(...)
                 my.predict.da(..., measure = "err"),
               tunecontrol = tune.control(cross = 100))

## Error: object is not a matrix

tn.lda

## Error: object 'tn.lda' not found

tn.qda
```

```
##
## Error estimation of 'qda' using 100-fold cross validation: 0.5047

tn.glm

##
## Error estimation of 'glm' using 100-fold cross validation: 0.4886

tn.nb

##
## Error estimation of 'naiveBayes' using 10-fold cross validation: 0.04667

tn.mlda

## Error: object 'tn.mlda' not found

tn.mqda

## Error: object 'tn.mqda' not found

summary(tn.mglm$best.model)

##
## Call:
## best.tune(method = glm, train.x = Direction ~ poly(Volume, Lag1,
##      Lag2, degree = 2), data = Smarket, predict.func = function(...) my.predict.g
##      measure = "err"), tunecontrol = tune.control(cross = 100),
##      family = binomial(link = "logit"))
##
## Deviance Residuals:
##      Min        1Q    Median        3Q        Max
## -1.35    -1.21     1.07     1.14     1.41
##
## Coefficients:
##                                     Estimate Std. Error
## (Intercept)                        0.0741     0.0568
## poly(Volume, Lag1, Lag2, degree = 2)1.0.0    1.9147     2.0301
## poly(Volume, Lag1, Lag2, degree = 2)2.0.0   -0.5865     2.0433
## poly(Volume, Lag1, Lag2, degree = 2)0.1.0   -3.3551     2.0676
## poly(Volume, Lag1, Lag2, degree = 2)1.1.0   39.7782    66.9372
## poly(Volume, Lag1, Lag2, degree = 2)0.2.0   -0.1326     2.1054
## poly(Volume, Lag1, Lag2, degree = 2)0.0.1   -1.9867     2.0329
## poly(Volume, Lag1, Lag2, degree = 2)1.0.1   53.0670    71.1294
## poly(Volume, Lag1, Lag2, degree = 2)0.1.1  -10.6615    55.7612
## poly(Volume, Lag1, Lag2, degree = 2)0.0.2   -1.1340     2.0500
##                                     z value Pr(>|z|)
## (Intercept)                        1.30      0.19
## poly(Volume, Lag1, Lag2, degree = 2)1.0.0    0.94      0.35
## poly(Volume, Lag1, Lag2, degree = 2)2.0.0   -0.29      0.77
```

```

## poly(Volume, Lag1, Lag2, degree = 2)0.1.0    -1.62    0.10
## poly(Volume, Lag1, Lag2, degree = 2)1.1.0     0.59    0.55
## poly(Volume, Lag1, Lag2, degree = 2)0.2.0    -0.06    0.95
## poly(Volume, Lag1, Lag2, degree = 2)0.0.1    -0.98    0.33
## poly(Volume, Lag1, Lag2, degree = 2)1.0.1     0.75    0.46
## poly(Volume, Lag1, Lag2, degree = 2)0.1.1    -0.19    0.85
## poly(Volume, Lag1, Lag2, degree = 2)0.0.2    -0.55    0.58
##
## (Dispersion parameter for binomial family taken to be 1)
##
##      Null deviance: 1731.2  on 1249  degrees of freedom
## Residual deviance: 1726.2  on 1240  degrees of freedom
## AIC: 1746
##
## Number of Fisher Scoring iterations: 3

summary(tn.mqda$best.model)

## Error: object 'tn.mqda' not found

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

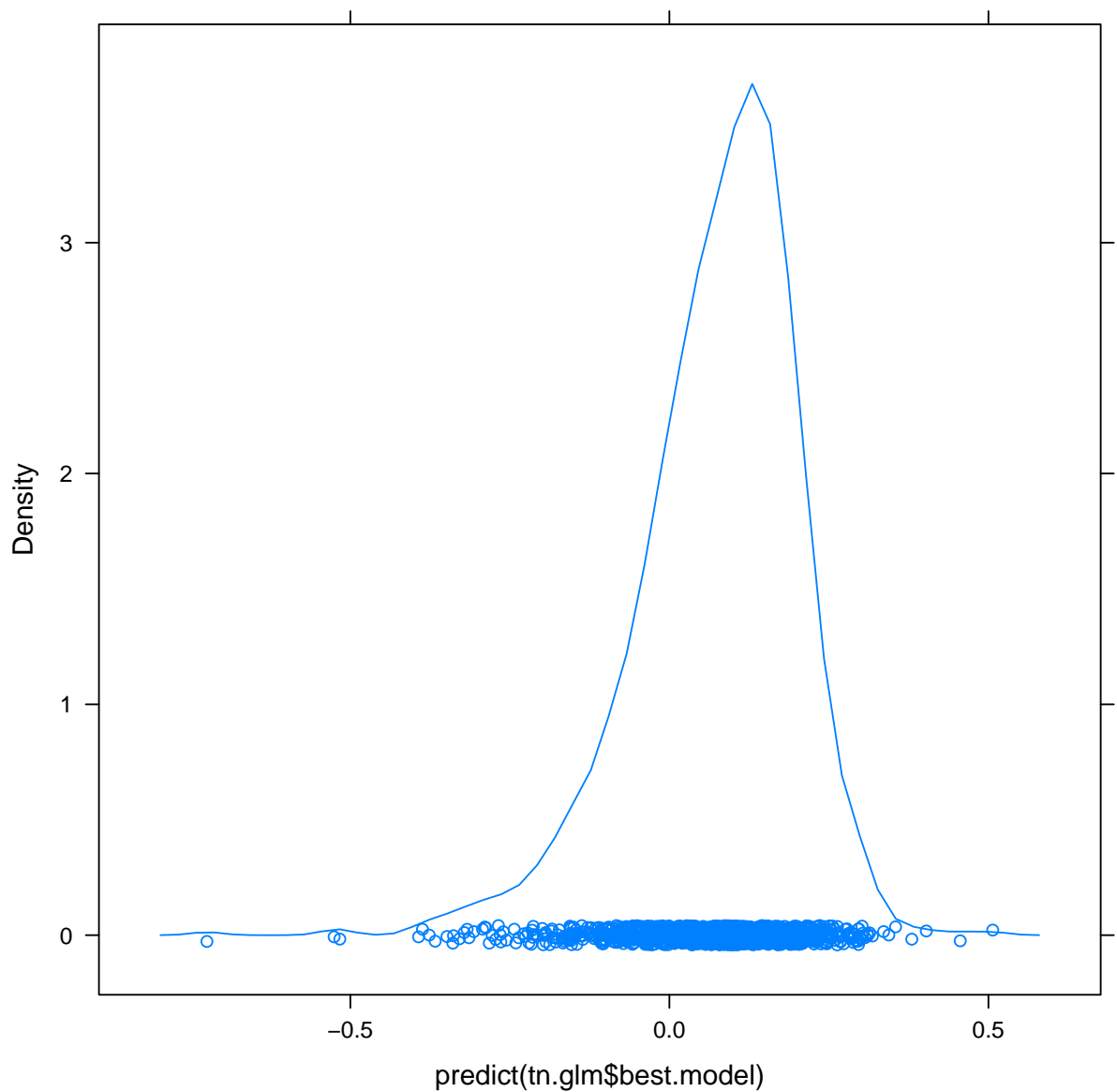
## Error: invalid argument to unary operator

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

## Error: invalid argument to unary operator

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)

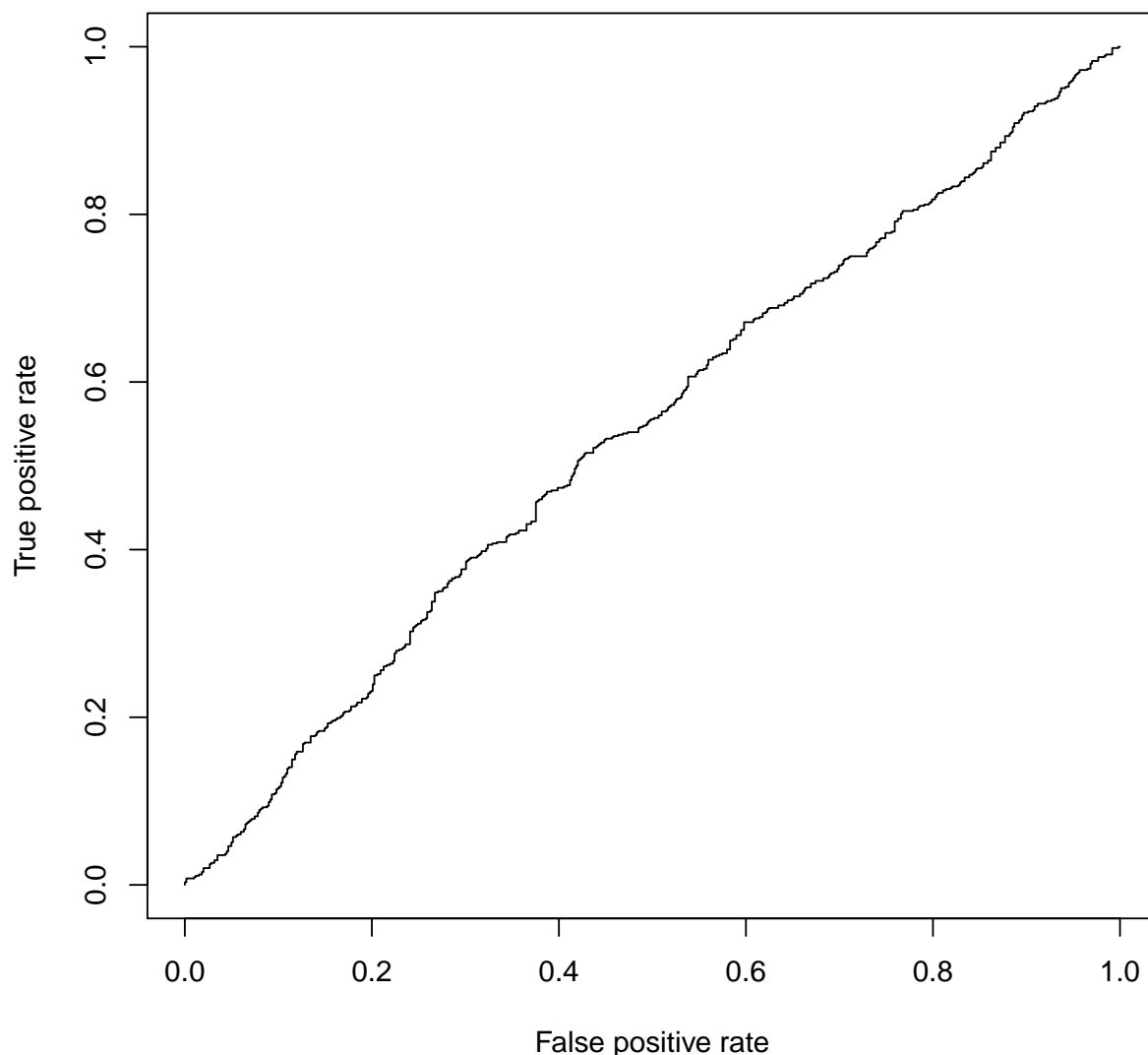
## Error: object is not a matrix

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

## Error: object 'Lag1' not found

plot(roc)

```



## 6 Рисование

Вообще в R существует по крайней мере три “школы” рисования. Во-первых, это классический пакет `graphics`. Пользоваться я 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).

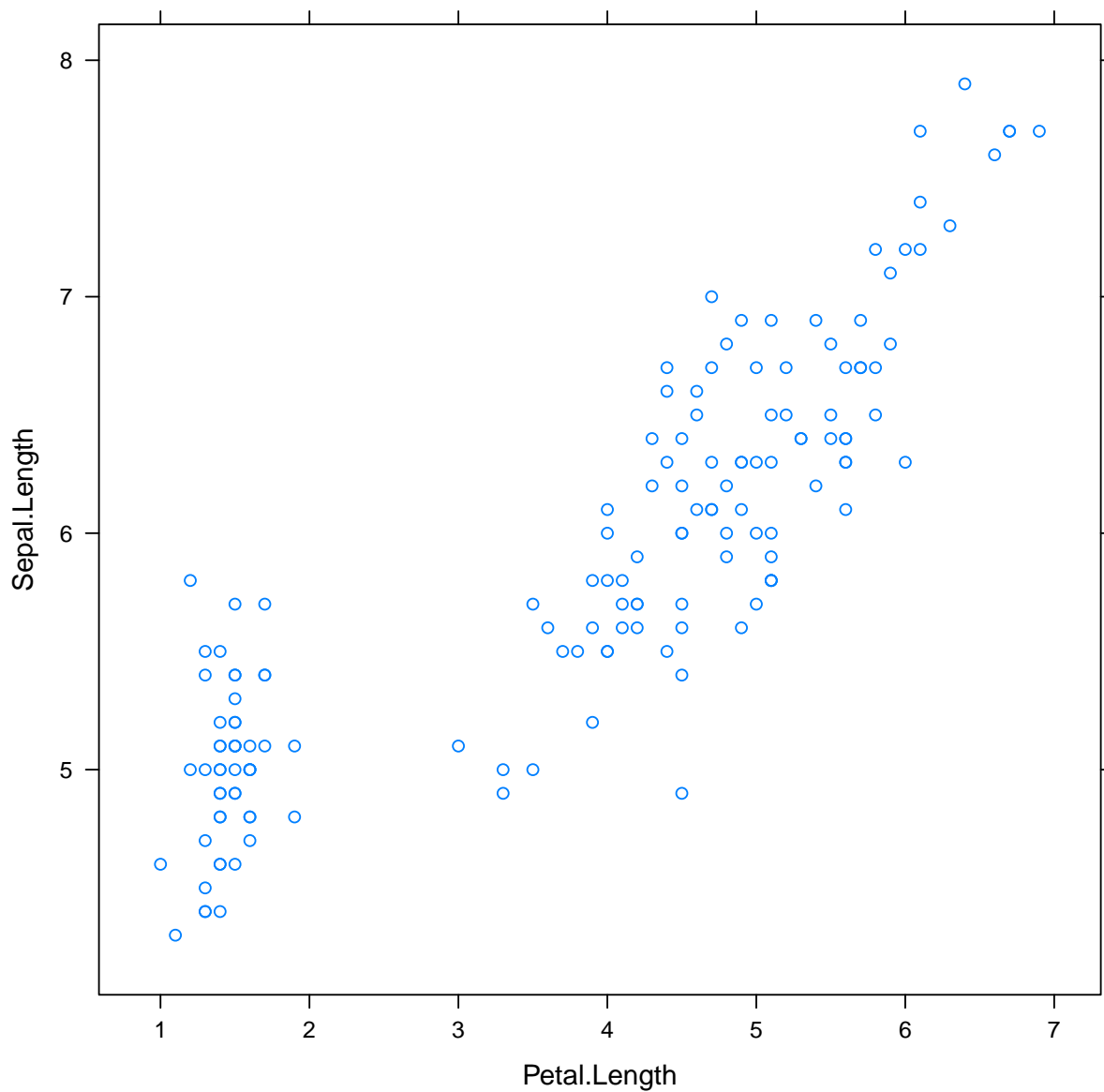
Все функции рисования в `lattice` принимают следующие параметры:

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:

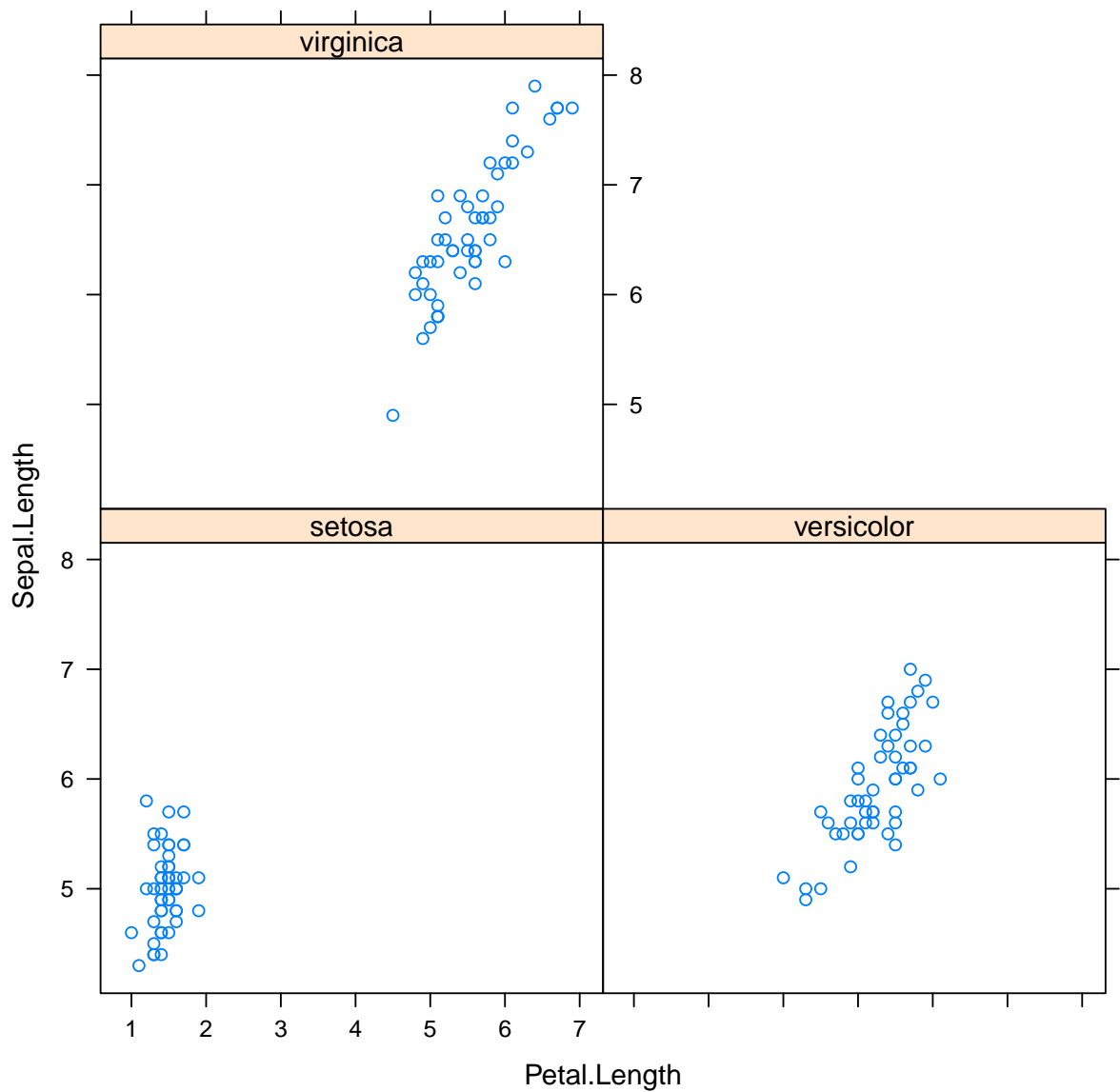
`xypplot()` — скаттерплот. Каждая строка датафрейма изображается отдельной точкой в координатах двух выбранных столбцов.

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



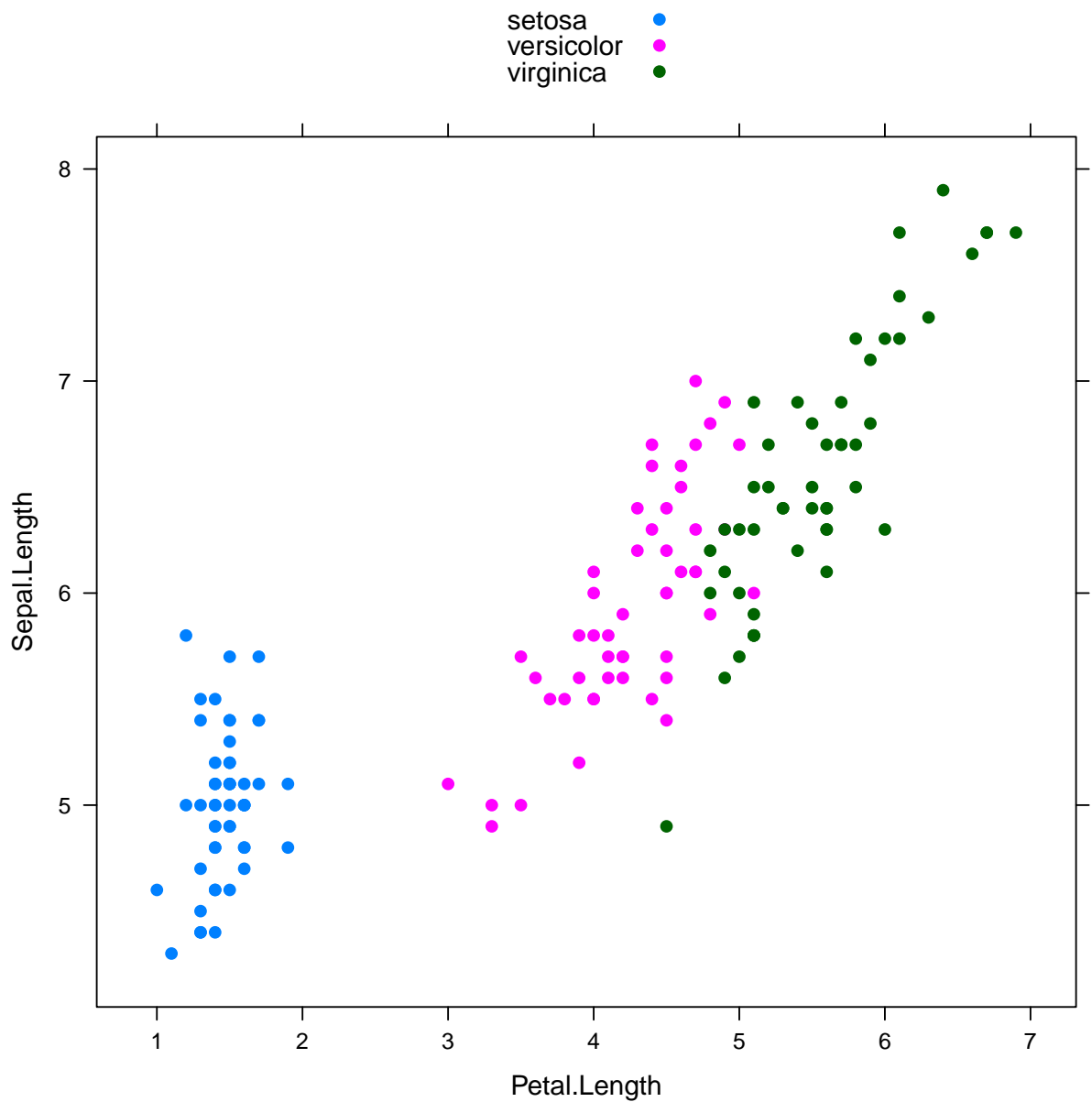
Не очень красиво и малоинформативно. Давайте нарисуем сорта на отдельных графиках:

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



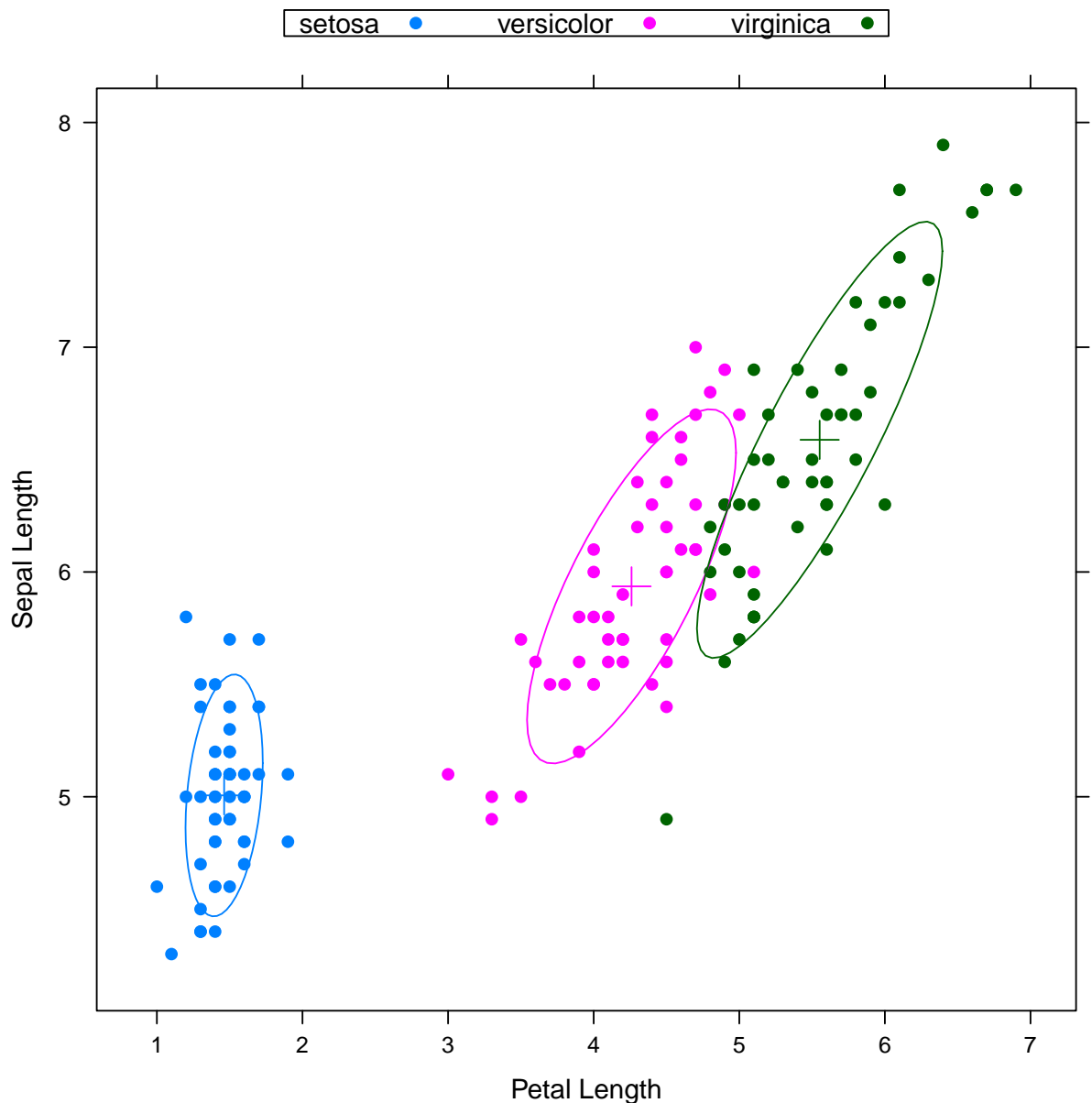
А может лучше все-таки на одном, но разными цветами?.. А еще я хочу сделать точки сплошным и добавить легенду:

```
xyplot(Sepal.Length ~ Petal.Length, groups = Species, data = iris, par.settings = s
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 `xuplot()`:

```
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)) {
```

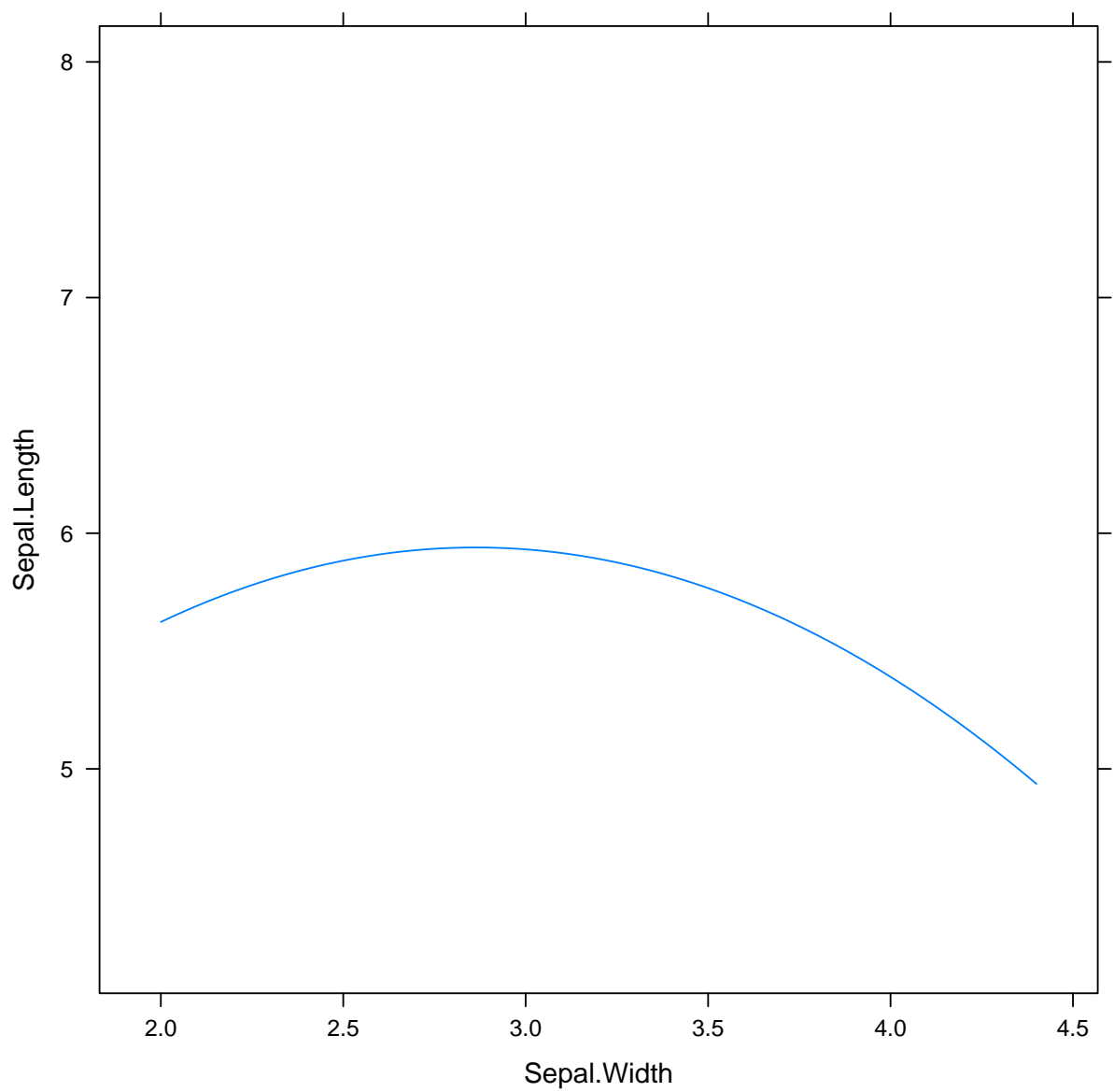
```

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))

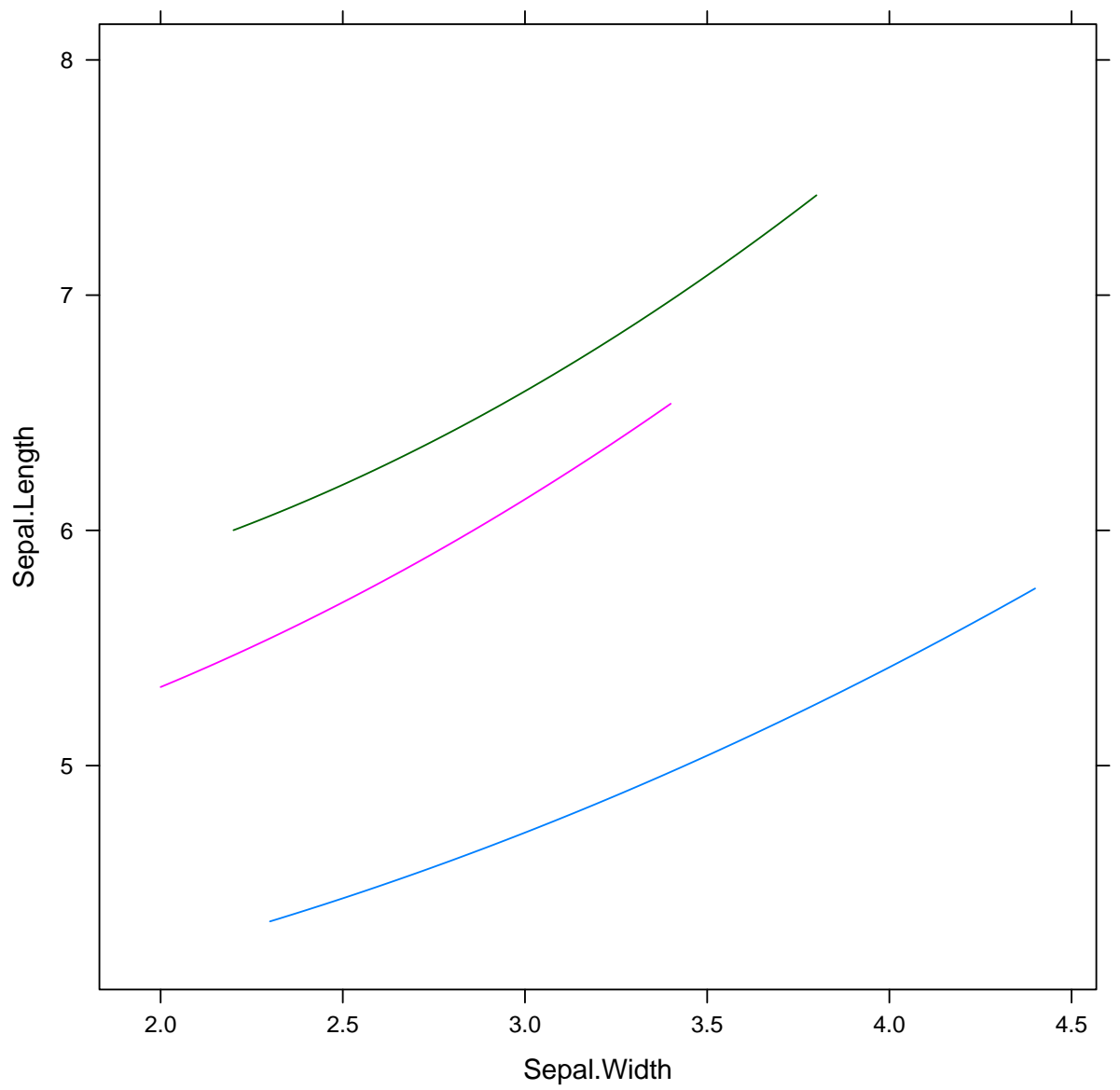
    par.line <- trellis.par.get("plot.line")
    panel.curve(predict(l, list(x = x)), from = min(x), to = max(x),
                col.line = col.line,
                lty = lty,
                lwd = lwd,
                alpha = alpha,
                ...,
                identifier = identifier)
  }
}
}

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

```

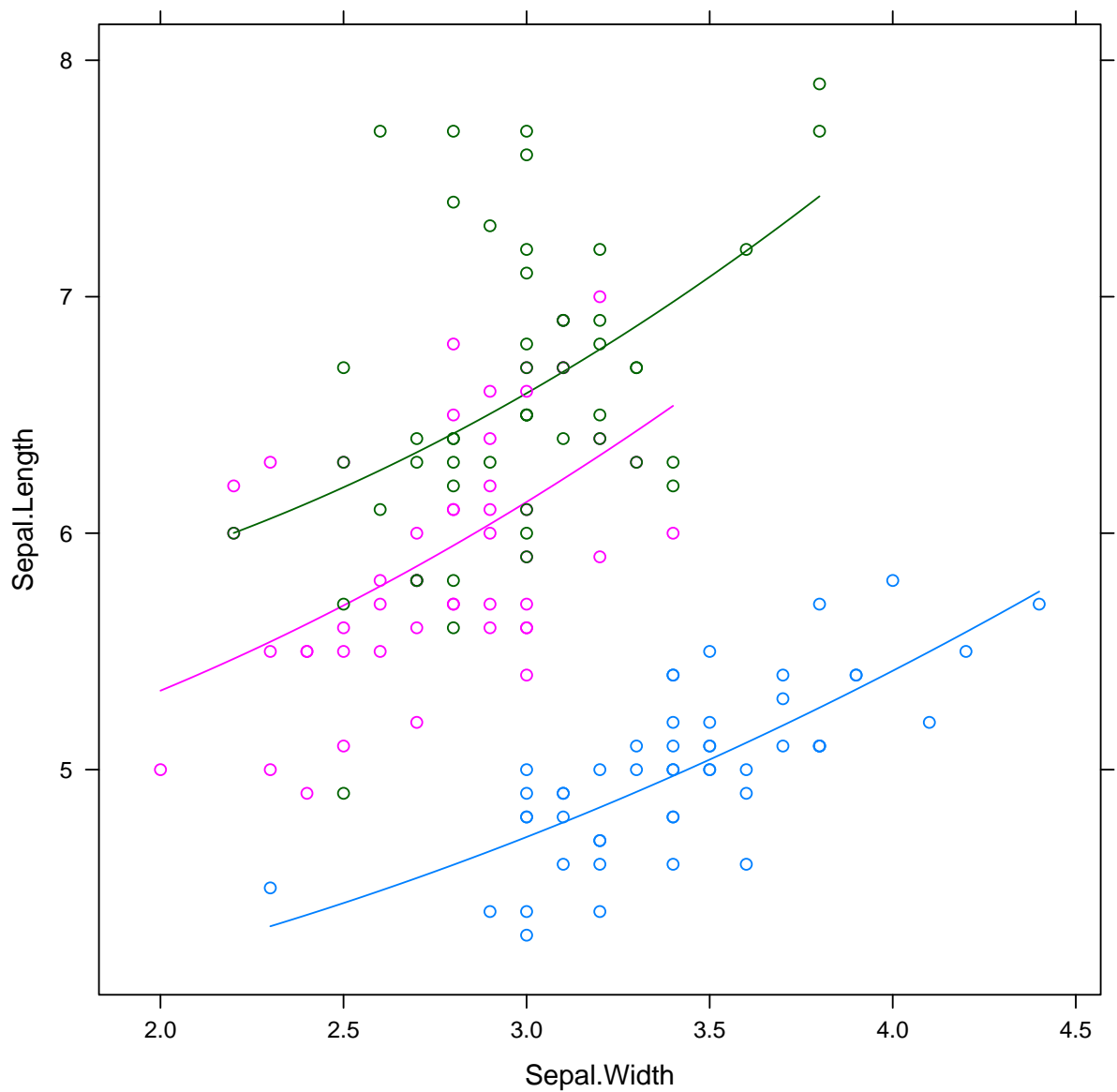


```
xyplot(Sepal.Length ~ Sepal.Width, groups = Species, data = iris, panel = panel.lmp
```



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





```
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)
```

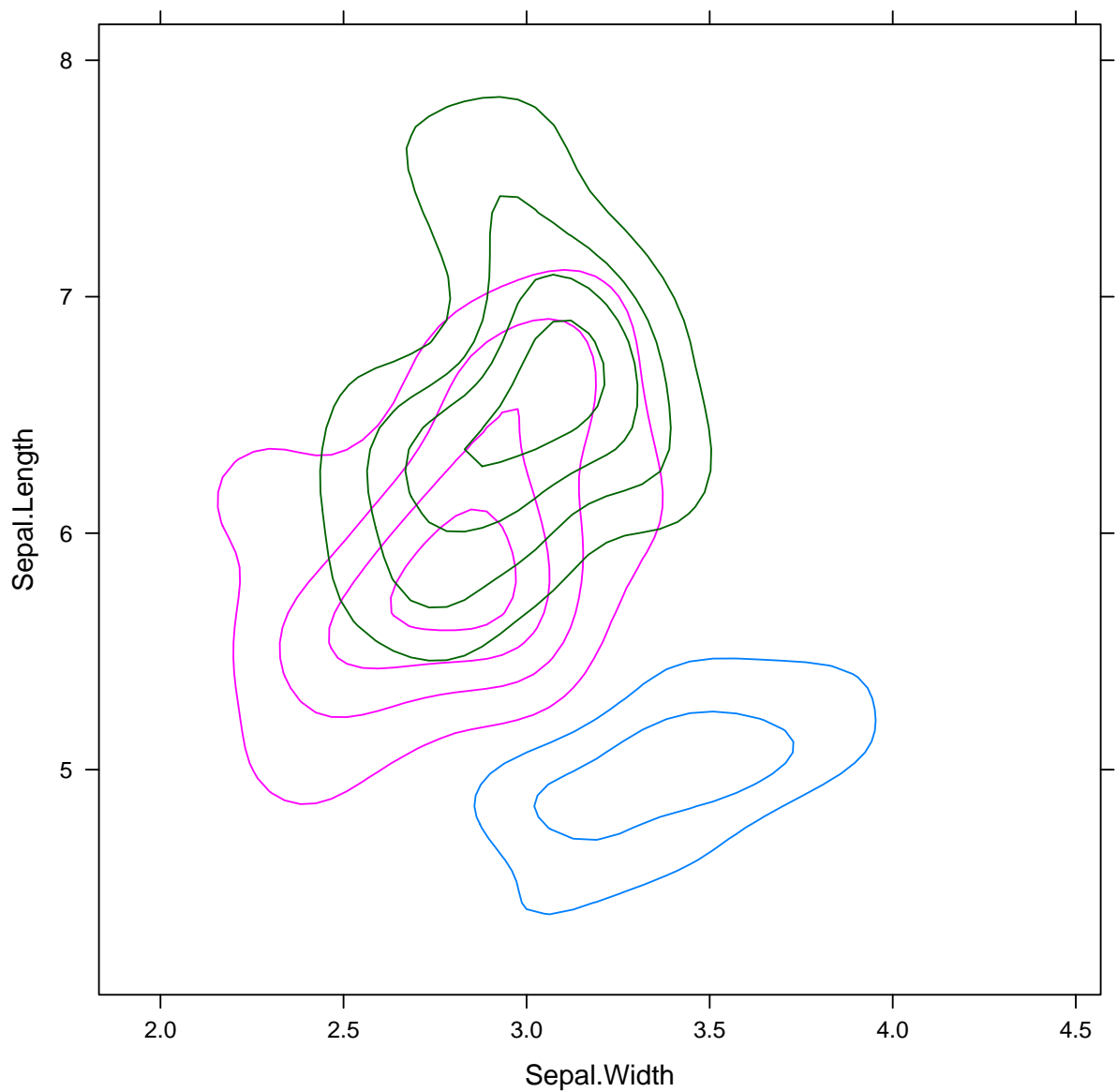
```

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)

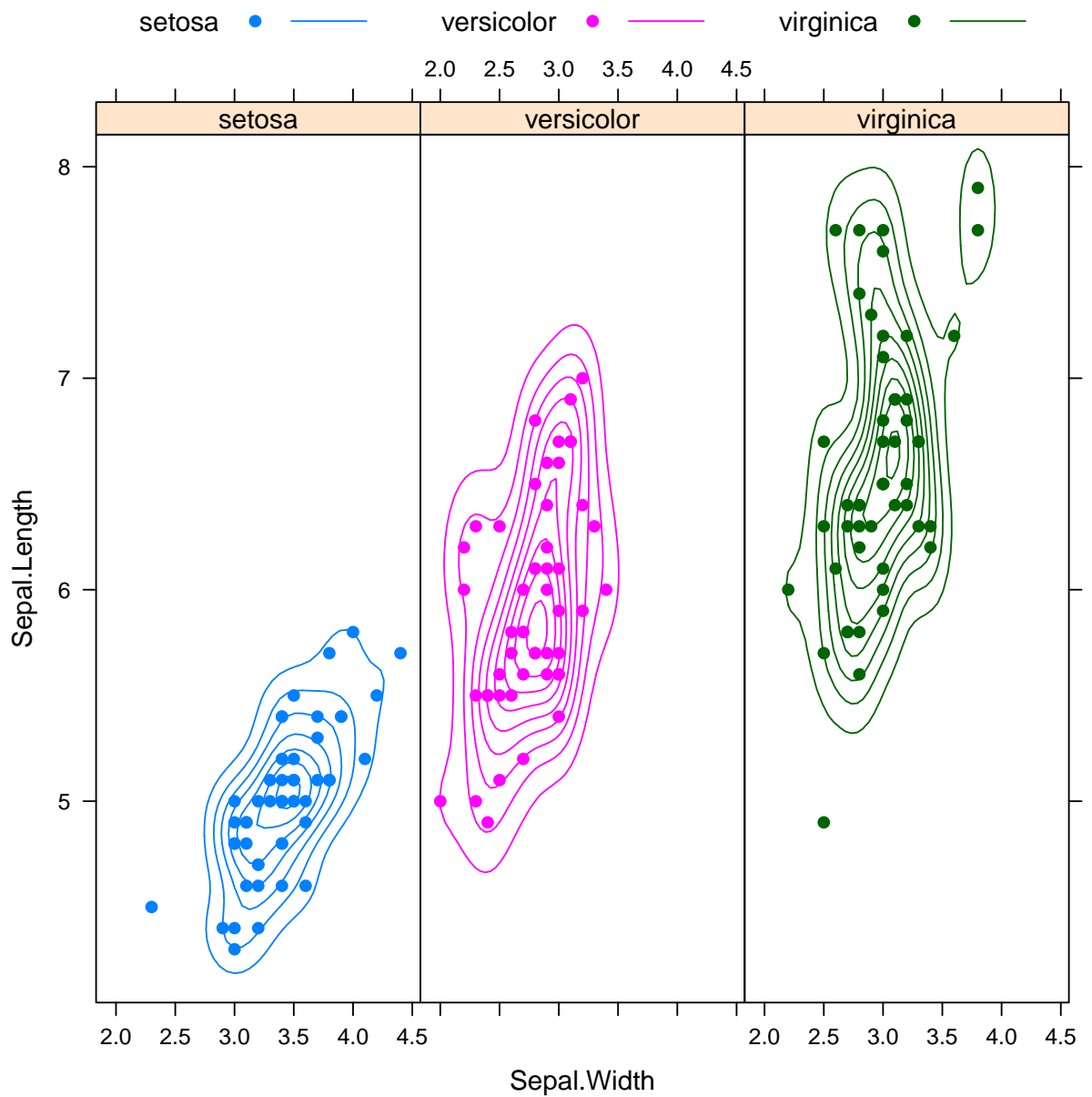
  plot.line <- trellis.par.get("plot.line")
  panel.contourplot(data$x, data$y, data$z,
    at = pretty(data$z, n = cuts),
    subscripts = seq_along(data$x),
    contour = TRUE, region = FALSE,
    col = col.line,
    lty = lty,
    lwd = lwd,
    alpha = alpha,
    ...,
    identifier = identifier)
}
}

xyplot(Sepal.Length ~ Sepal.Width, data = iris, groups = Species, panel = panel.kde

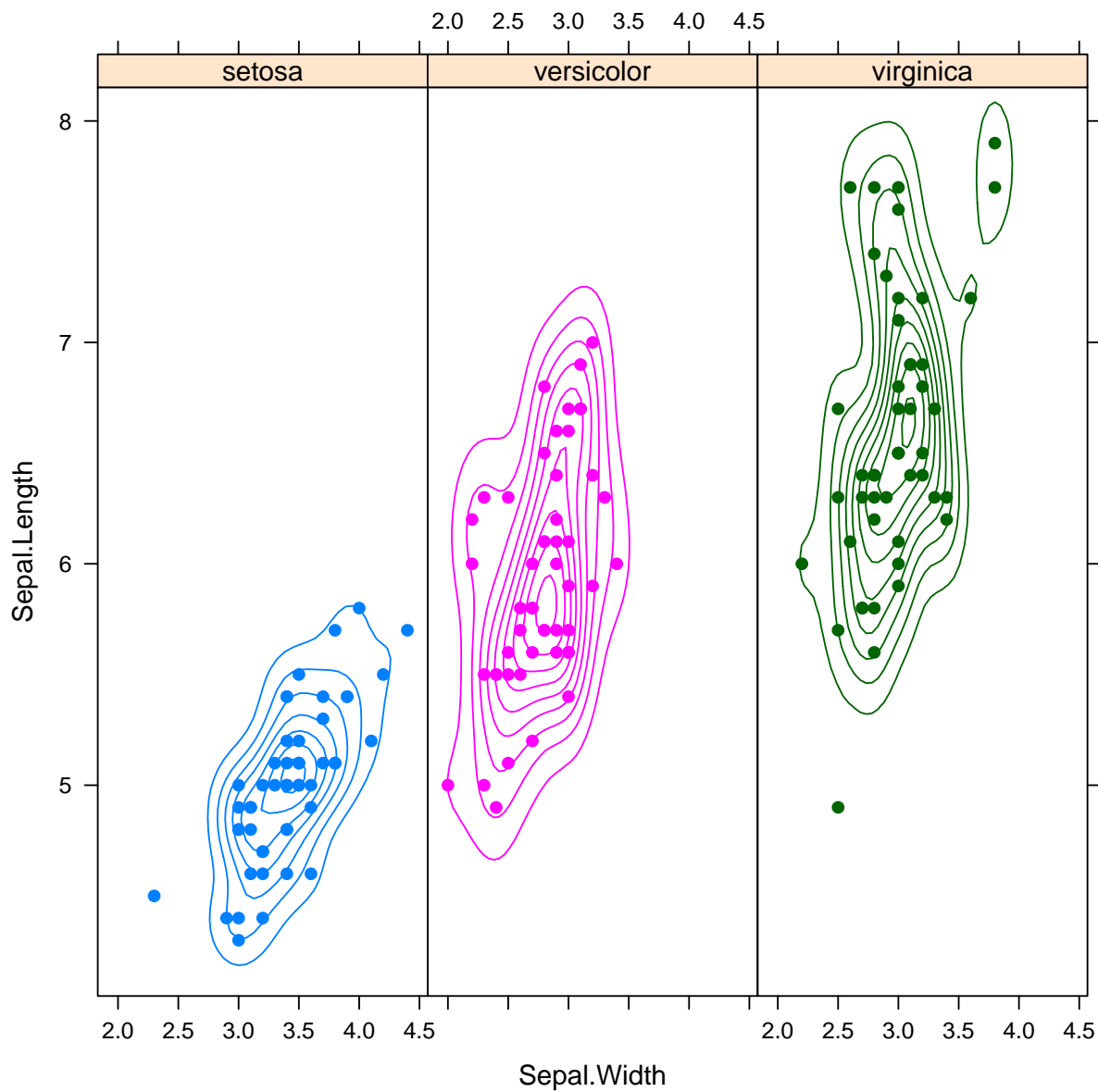
```



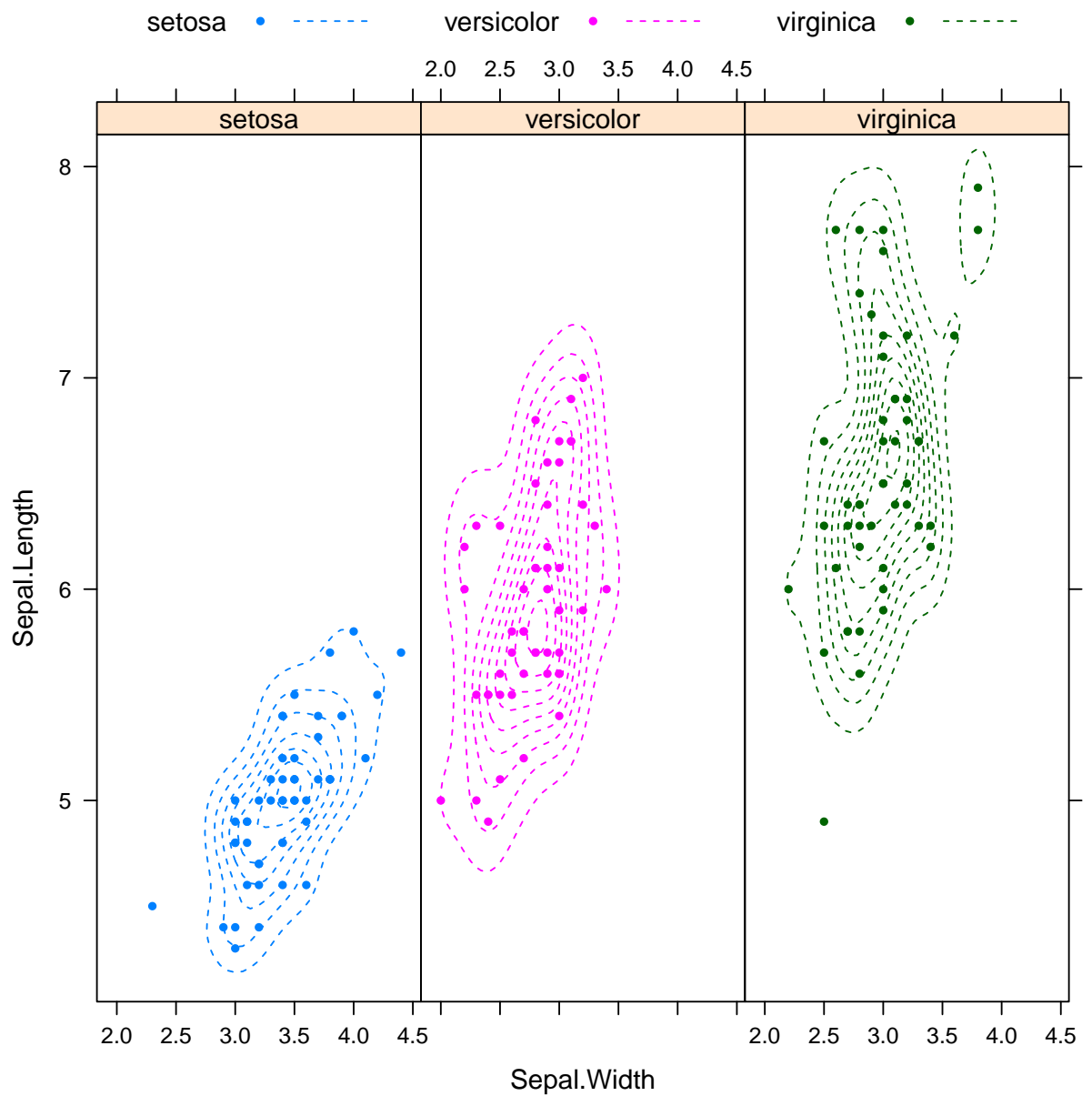
```
xyplot(Sepal.Length ~ Sepal.Width | Species, data = iris, groups = Species,
  par.settings = simpleTheme(pch = 19),
  auto.key = list(columns = 3, lines = TRUE),
  layout = c(3, 1)) + layer_(panel.kde2d(..., cuts = 10))
```



```
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))
```



bwplot violinplot densityplot marginal.plot

countourplot

Пример панельной функции для полиномиальной регрессии