

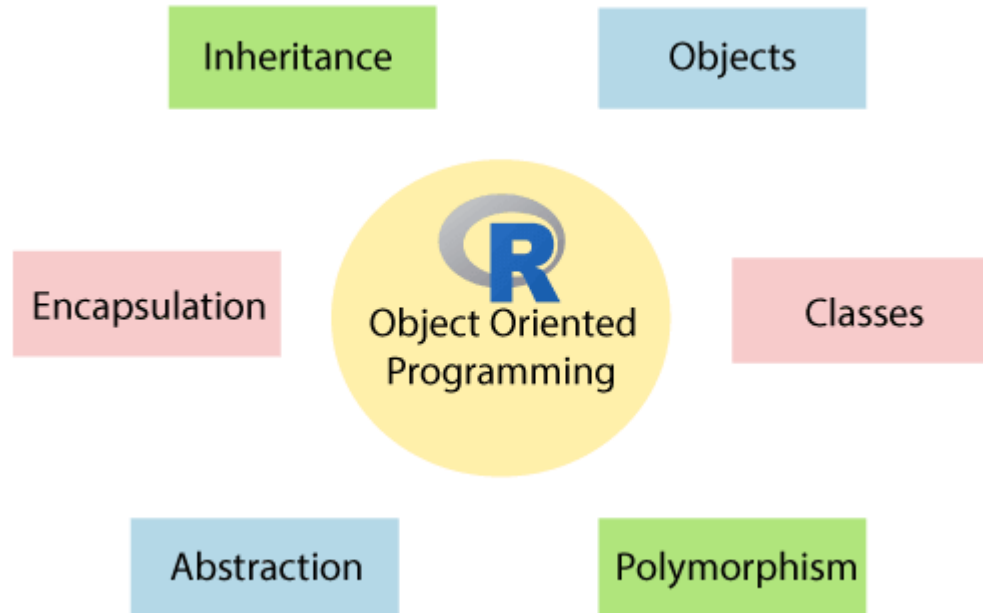
Программирование и статистический анализ данных на языке R

Лекция 2 (Основы программирования на языке R)



Петровский Михаил (ВМК МГУ), michael@cs.msu.su

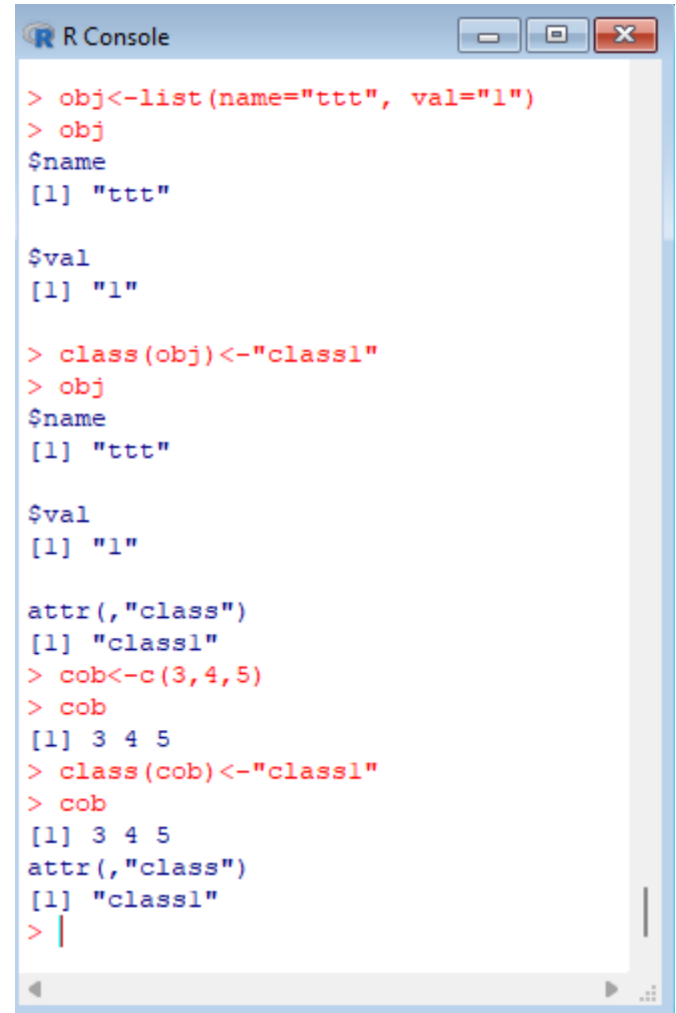
ООП в R



- Несколько «подходов» в R: S3, S4 (Reference Classes иногда считают частью S4, иногда отдельным подходом)
- S3 – «старый», простой, гибкий, менее формальный и структурированный, очень важный с точки зрения понимания R (много на нем)
- S4 – «новый», более формальный и мощный

Особенности S3 классов

- У каждого объекта есть тип и class
- class – пользовательская строка или вектор, описывающие принадлежность классам
- Доступ к полям объекта через \$
- Создание объекта заданного класса (внимание!!!) просто присваивание значения атрибуту class, по сути нет никаких структурированных описаний объектов
- Нет контроля структуры классов
- Проверка принадлежности классу `is`
`is(объект, "class")`



```
R Console

> obj<-list(name="ttt", val="1")
> obj
$name
[1] "ttt"

$val
[1] "1"

> class(obj)<-"class1"
> obj
$name
[1] "ttt"

$val
[1] "1"

attr(,"class")
[1] "class1"

> cob<-c(3,4,5)
> cob
[1] 3 4 5

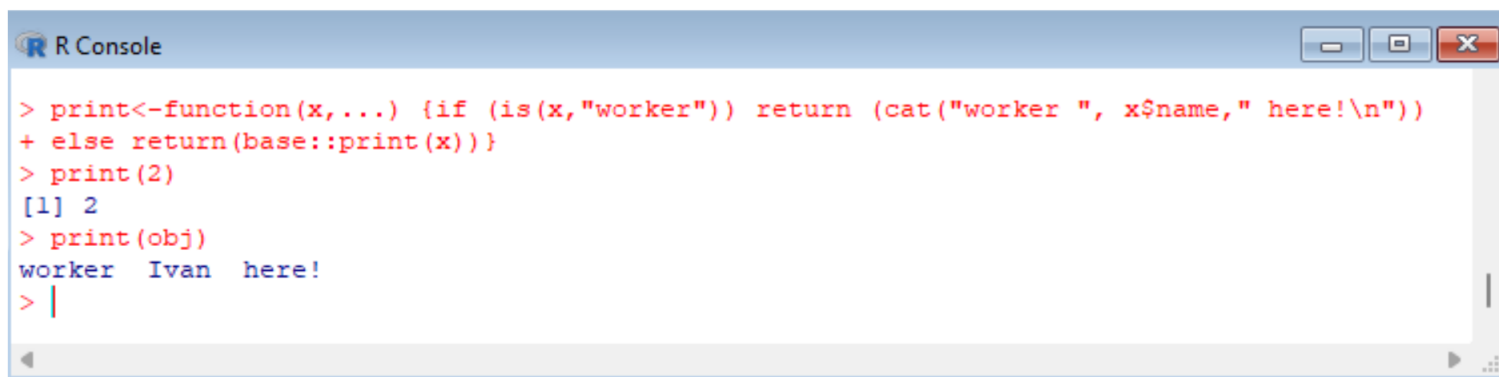
> class(cob)<-"class1"
> cob
[1] 3 4 5

attr(,"class")
[1] "class1"

> |
```

Переопределение функций - работающий, но «плохой» путь

- Проверять класс через `is` и вызывать базовый метод через `base::method`



```
R Console

> print<-function(x,...) {if (is(x,"worker")) return (cat("worker ", x$name," here!\n"))
+ else return(base::print(x))}
> print(2)
[1] 2
> print(obj)
worker Ivan here!
> |
```

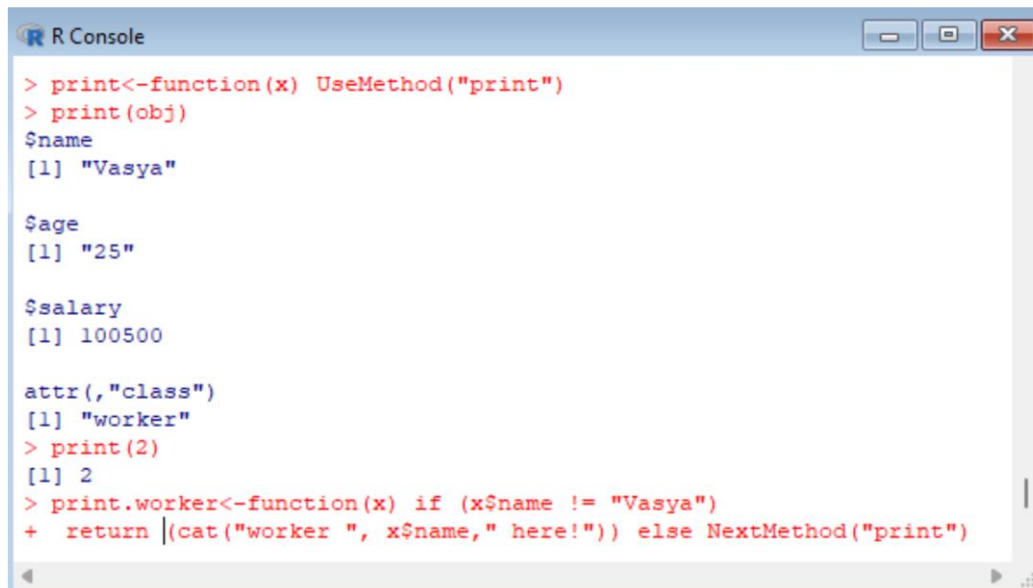
- Почему плохо? Источник ошибок, много переписывать для поддержки новых классов, если вдруг совпадают имена, то есть зависимость от последовательности загрузки пакетов ...

Переопределение функций и вызов по имени через диспетчер

- «Хороший» путь – через диспетчерские вызовы, где вызываемый метод определяется классом первого аргумента

UseMethod(«метод»,«класс»),

- Вызов “родителя” **NextMethod(«метод»,«класс»)**
- Надо определить функцию **<generic name>.<class name>**
- По умолчанию вызов **<generic name>.default**

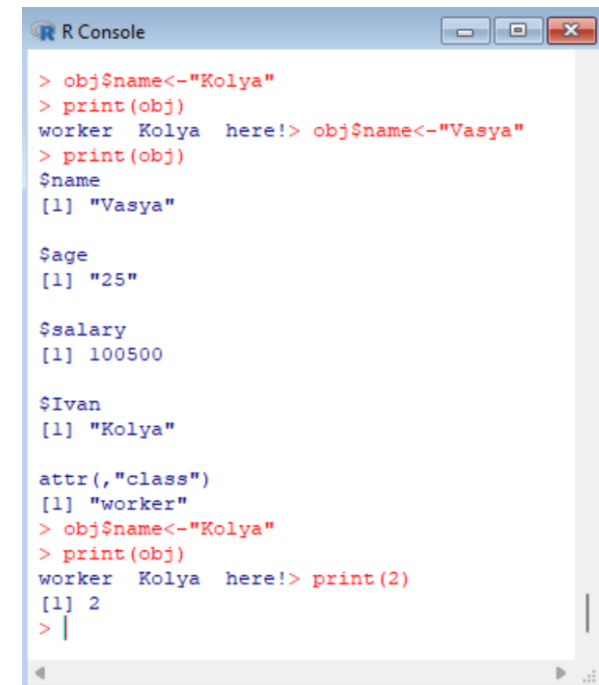


```
> print<-function(x) UseMethod("print")
> print(obj)
$name
[1] "Vasya"

$age
[1] "25"

$salary
[1] 100500

attr(,"class")
[1] "worker"
> print(2)
[1] 2
> print.worker<-function(x) if (x$name != "Vasya")
+   return (cat("worker ", x$name, " here!")) else NextMethod("print")
```



```
> obj$name<-"Kolya"
> print(obj)
worker Kolya here!> obj$name<-"Vasya"
> print(obj)
$name
[1] "Vasya"

$age
[1] "25"

$salary
[1] 100500

$Ivan
[1] "Kolya"

attr(,"class")
[1] "worker"
> obj$name<-"Kolya"
> print(obj)
worker Kolya here!> print(2)
[1] 2
> |
```

Наследование в S3

- По сути за счет включения дополнительных классов в атрибут `class` (порядок важен) , задается список, влияет только на поиск «родительского» метода
- и использования «родительских функций» при переопределении через **<generic name>.<class name>**
- Если функция **<generic name>** не переопределена, то при обращении к ней в объекте класса **<class name>** будет искаться следующая **<generic name>.<class name next>**, где **<class name next>** следующий класс в списке из атрибута `class`
- Полезная функция для проверки есть ли метод у класса
getS3method(<generic name>,<class name>)
- Как наследуются признаки? НИКАК!

Пример наследования в S3

```
R Console
>
> obj1<-list(name="Vaysa", salary=100, age=20)
> class(obj1)<-"worker"
> obj2<-list(name="Ivan", salary=150, age=30, prof="Wkr")
> class(obj2)<-c("superworker", "worker")
> print.worker<-function(x) if (x$name != "Vova") return (cat("worker ", x$name, " here!\n")) else NextMethod("print")
> print.superworker<-function(x) if (x$name != "Kolya") return (cat("supeworker", x$name, " here!\n")) else NextMethod("print")
> obj1
worker Vaysa here!
> obj2
supeworker Ivan here!
> obj2$name<-"Kolya"
> obj2
worker Kolya here!
> class(obj2)
[1] "superworker" "worker"
> getS3method("print", "superworker")
function(x) if (x$name != "Kolya") return (cat("supeworker", x$name, " here!\n")) else NextMethod("print")
<bytecode: 0x000001df15f38d10>
> |
```

Пример наследования в S3

```
> glm.mtcars = glm(mpg ~ ., data = mtcars)
> glm.mtcars

Call:  glm(formula = mpg ~ ., data = mtcars)

Coefficients:
(Intercept)      cyl      disp      hp      drat      wt      qsec      vs      am      gear      carb
  12.30337   -0.11144    0.01334   -0.02148    0.78711   -3.71530    0.82104    0.31776    2.52023    0.65541   -0.19942

Degrees of Freedom: 31 Total (i.e. Null);  21 Residual
Null Deviance:      1126
Residual Deviance: 147.5      AIC: 163.7

> class(glm.mtcars)
[1] "glm" "lm"
> is(glm.mtcars,"glm")
[1] TRUE
> is(glm.mtcars,"lm")
[1] TRUE
> lm.mtcars = lm(mpg ~ ., data = mtcars)
> lm.mtcars

Call:
lm(formula = mpg ~ ., data = mtcars)

Coefficients:
(Intercept)      cyl      disp      hp      drat      wt      qsec      vs      am      gear      carb
  12.30337   -0.11144    0.01334   -0.02148    0.78711   -3.71530    0.82104    0.31776    2.52023    0.65541   -0.19942

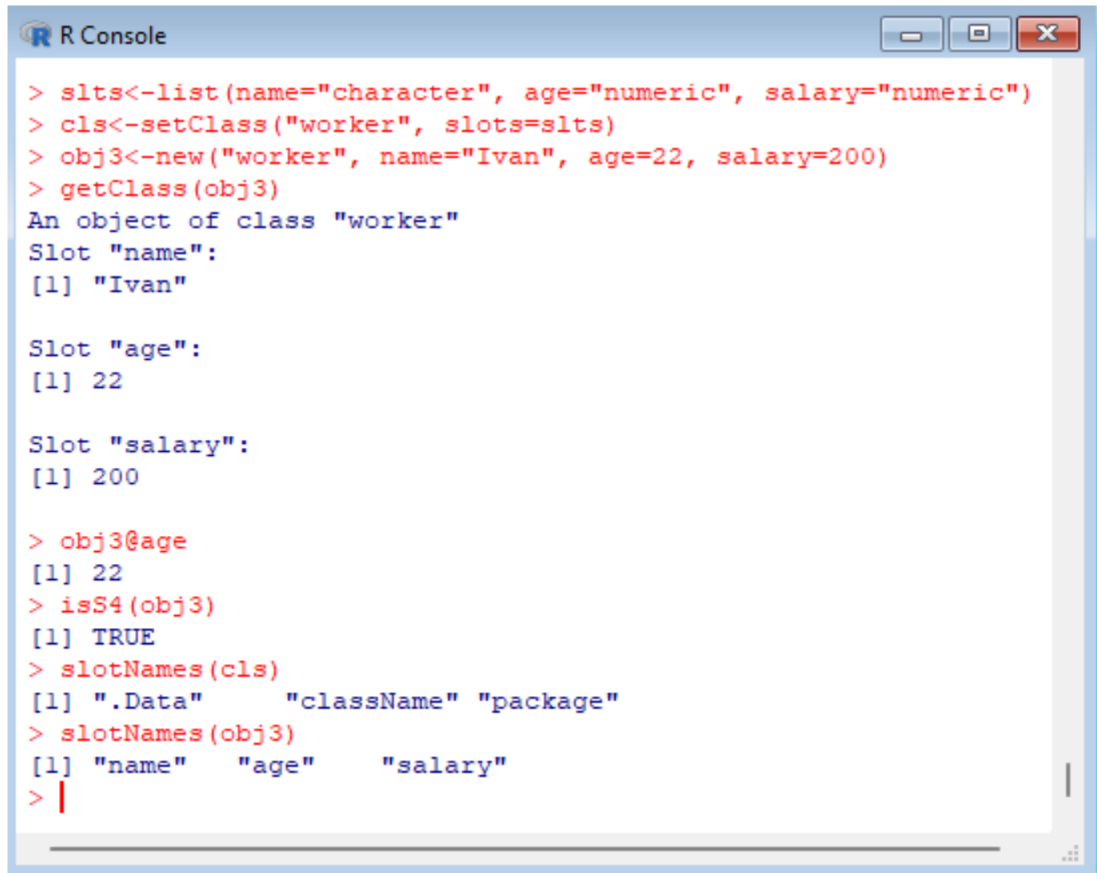
> class(lm.mtcars)
[1] "lm"
> is(lm.mtcars,"glm")
[1] FALSE
> is(lm.mtcars,"lm")
[1] TRUE
```


Особенности S4 классов по сравнению с S3

- Формальное определение классов
- Валидация соответствия объектов и классов
- Полиморфизм методов позволяет учитывать не только класс, но и аргументы
- Задание класса:
setClass(Class, representation, prototype, contains=character(),
validity, access, where, version, sealed, package, S3methods = FALSE, slots)
Class – имя
slots – слоты (а ля поля будущего объекта), список
contains – наследование, перечень
prototype – настройки слотов по умолчанию
validity – функция проверки соответствия объекта классу
- Виртуальные классы: `representation("VIRTUAL")`

Создание S4 класса

- Обычно через `new` с указанием класса и значений слотов, которые не берутся по умолчанию
- Доступ к слотам через `@` не через `$`, чтобы не было проблем с доступом к родительским слотам
- Получение имен слотов `slotNames`
- описание класса через `getClass`



```
R Console
> slts<-list(name="character", age="numeric", salary="numeric")
> cls<-setClass("worker", slots=slts)
> obj3<-new("worker", name="Ivan", age=22, salary=200)
> getClass(obj3)
An object of class "worker"
Slot "name":
[1] "Ivan"

Slot "age":
[1] 22

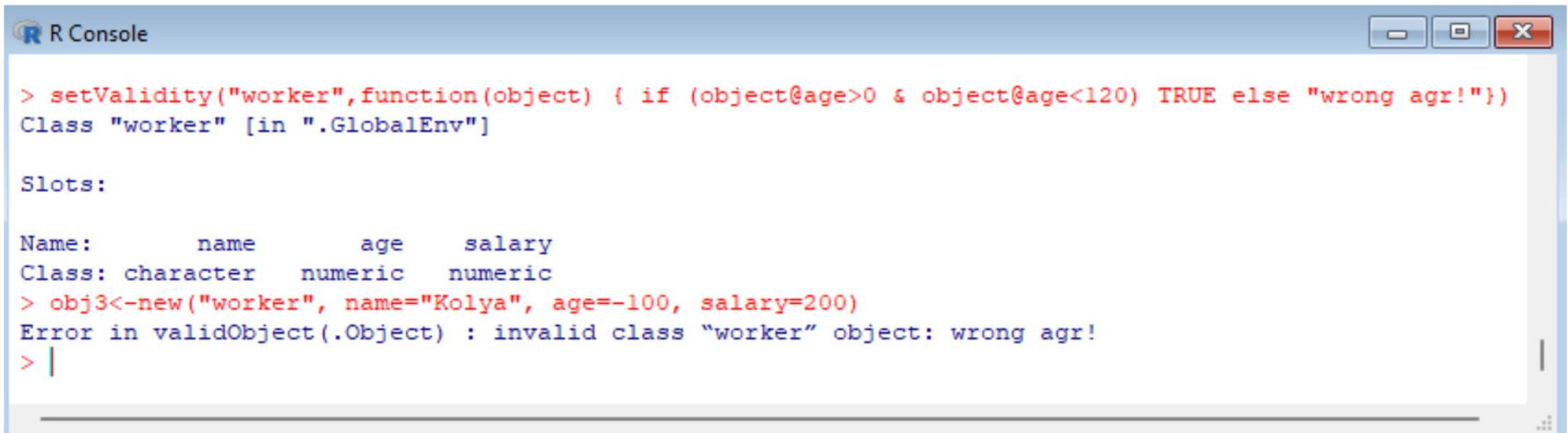
Slot "salary":
[1] 200

> obj3@age
[1] 22
> isS4(obj3)
[1] TRUE
> slotNames(cls)
[1] ".Data"      "className"  "package"
> slotNames(obj3)
[1] "name"      "age"      "salary"
> |
```

Валидация

- Можно задать функцию для проверки соответствия объекта классу:

setValidity(«имя класса», функция)



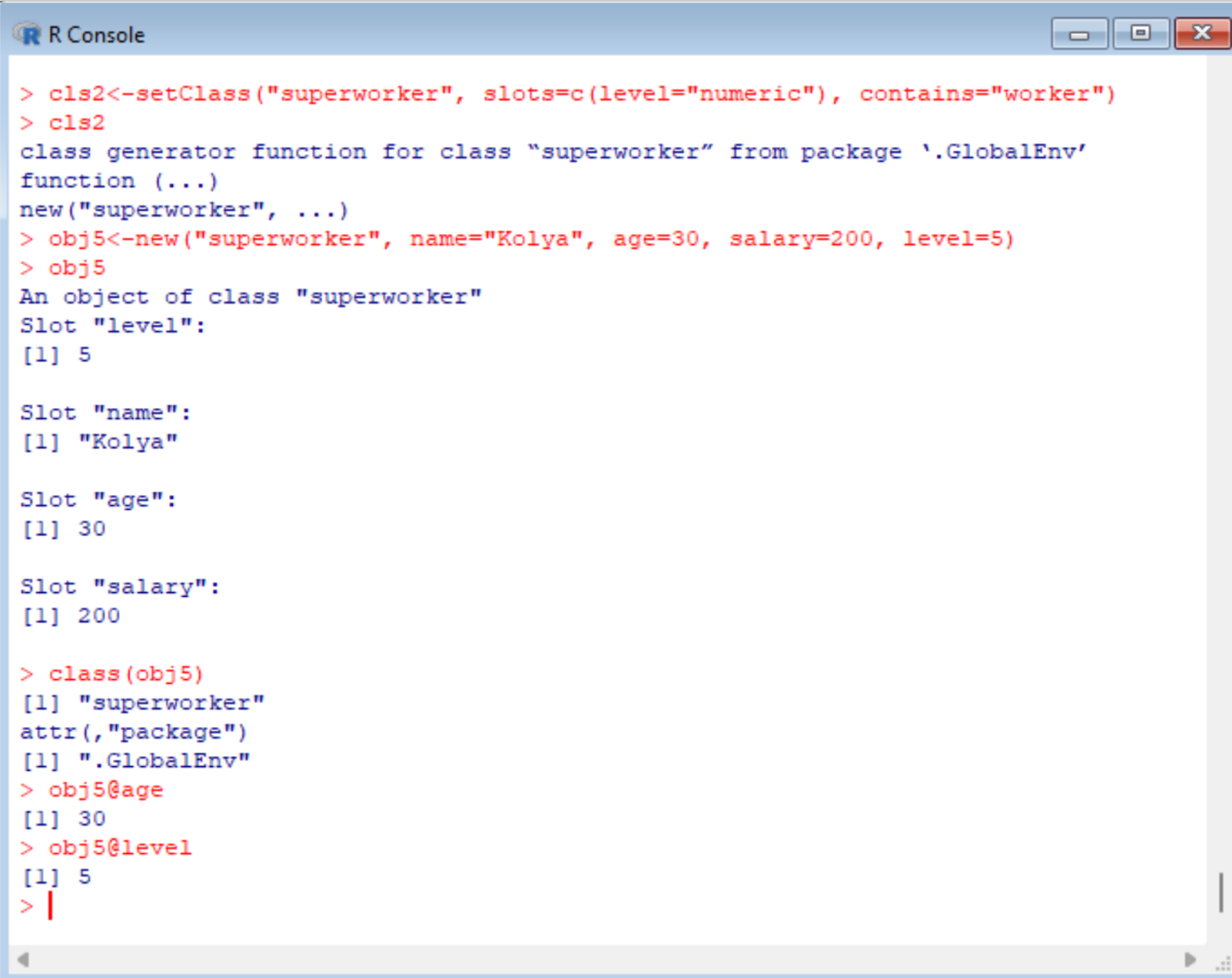
```
> setValidity("worker",function(object) { if (object@age>0 & object@age<120) TRUE else "wrong agr!"})
Class "worker" [in ".GlobalEnv"]

Slots:

Name:      name      age      salary
Class: character numeric numeric
> obj3<-new("worker", name="Kolya", age=-100, salary=200)
Error in validObject(.Object) : invalid class "worker" object: wrong agr!
> |
```

S4 наследование

- Делается через параметр `contains` в `setClass`
- В отличие от S3 поля (слоты) тоже наследуются



```
> cls2<-setClass("superworker", slots=c(level="numeric"), contains="worker")
> cls2
class generator function for class "superworker" from package '.GlobalEnv'
function (...)
new("superworker", ...)
> obj5<-new("superworker", name="Kolya", age=30, salary=200, level=5)
> obj5
An object of class "superworker"
Slot "level":
[1] 5

Slot "name":
[1] "Kolya"

Slot "age":
[1] 30

Slot "salary":
[1] 200

> class(obj5)
[1] "superworker"
attr(,"package")
[1] ".GlobalEnv"
> obj5@age
[1] 30
> obj5@level
[1] 5
> |
```

Определение методов в S4

- задаем базовый метод:

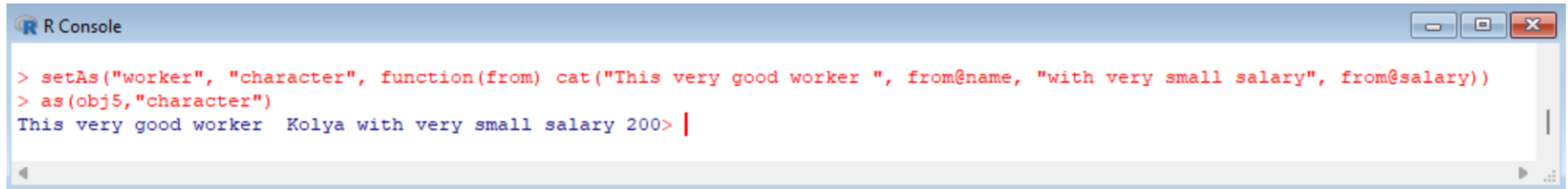
setGeneric(«имя метода», **function(x)** **standardGeneric**(«имя метода»))

- задаем реализацию (для разных классов в иерархии наследования с разным функционалом)

setMethod(«имя метода», «имя класса», **function(x)**)

- явное преобразование типов:

setAs(«имя класса from», «имя класса to», **function(x)**)



```
R Console
> setAs("worker", "character", function(from) cat("This very good worker ", from$name, "with very small salary", from$salary))
> as(obj5, "character")
This very good worker  Kolya with very small salary 200> |
```

Пример с переопределением метода в S4

```
R Console

> setGeneric("promote",function(x,...) standardGeneric("promote"))
[1] "promote"
> setMethod("promote", "worker", function(x,...) {if (x@age>40) TRUE else FALSE})
> setMethod("promote", "superworker", function(x,...) {if (x@age>30) TRUE else FALSE})
> obj3
An object of class "worker"
Slot "name":
[1] "Ivan"

Slot "age":
[1] 35

Slot "salary":
[1] 200

> promote(obj3)
[1] FALSE
> obj5
An object of class "superworker"
Slot "level":
[1] 5

Slot "name":
[1] "Kolya"

Slot "age":
[1] 35

Slot "salary":
[1] 200

> promote(obj5)
[1] TRUE
```

Наследование S3 классов в S4

- S3 классы должны быть объявлены через `setOldClass`
- Как всегда в S3 наследуются только методы

```
R Console

> objOld<-list(name="Sasha")
> class(objOld)<-"oldWorker"
> print.oldWorker<-function(x) {cat("Old worker here!\n"); NextMethod("print")}
> setOldClass("oldWorker")
> objOld
Old worker here!
$name
[1] "Sasha"

attr(,"class")
[1] "oldWorker"

> clsl<-setClass("newWorker", slot=c(salary="numeric"),contains="oldWorker")
> newObj<-new("newWorker",salary=500)
> newObj
Object of class "newWorker"
Old worker here!
<S4 Type Object>
attr(,"class")
[1] "oldWorker"
Slot "salary":
[1] 500

> isS4(objOld)
[1] FALSE
> isS4(objNew)
[1] TRUE
> objNew@.S3Class
[1] "oldWorker"
>
> |
```

Reference Class

- Отличие от обычного S4:
 - **setRefClass** вместо **setClass** создает генераторную функцию
 - доступ к методам и полям снова через **\$** вместо **@**
 - есть конструкторы (в том числе копирования) и деструкторы
 - можно напрямую создавать методы и поля
 - Можно наследовать через **contains**
 - Получить ссылку на текущий объект через **self**
 - Использовать **callSuper** для вызова родительских методов



```
> cls5<-setRefClass("progamer", fields=c(game="character", gamelvl="numeric", money="numeric"))
> cls5
Generator for class "progamer":

Class fields:

Name:      game   gamelvl   money
Class: character numeric  numeric

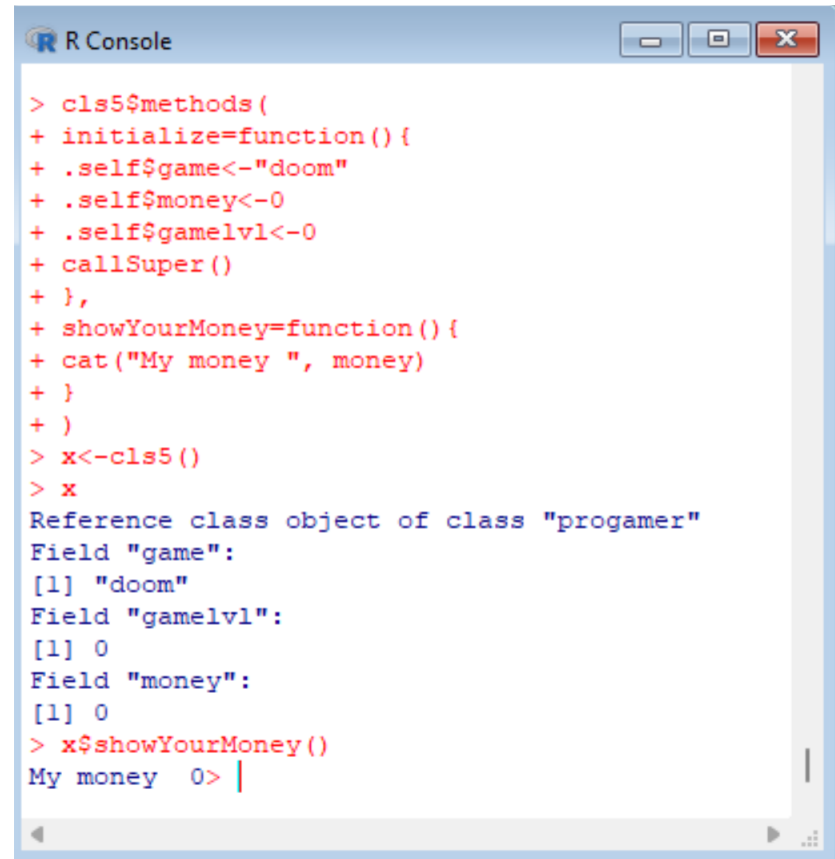
Class Methods:
  "field", "trace", "getRefClass", "initFields", "copy", "callSuper", ".objectPackage", "export", "untrace", "getClass", "show", "usingMethods",
  ".objectParent", "import"

Reference Superclasses:
  "envRefClass"

> |
```


Добавление методов

- Внутри **setRefClass** можно задать через параметр **methods** как именованный список
- Либо через генераторную функцию и **methods**



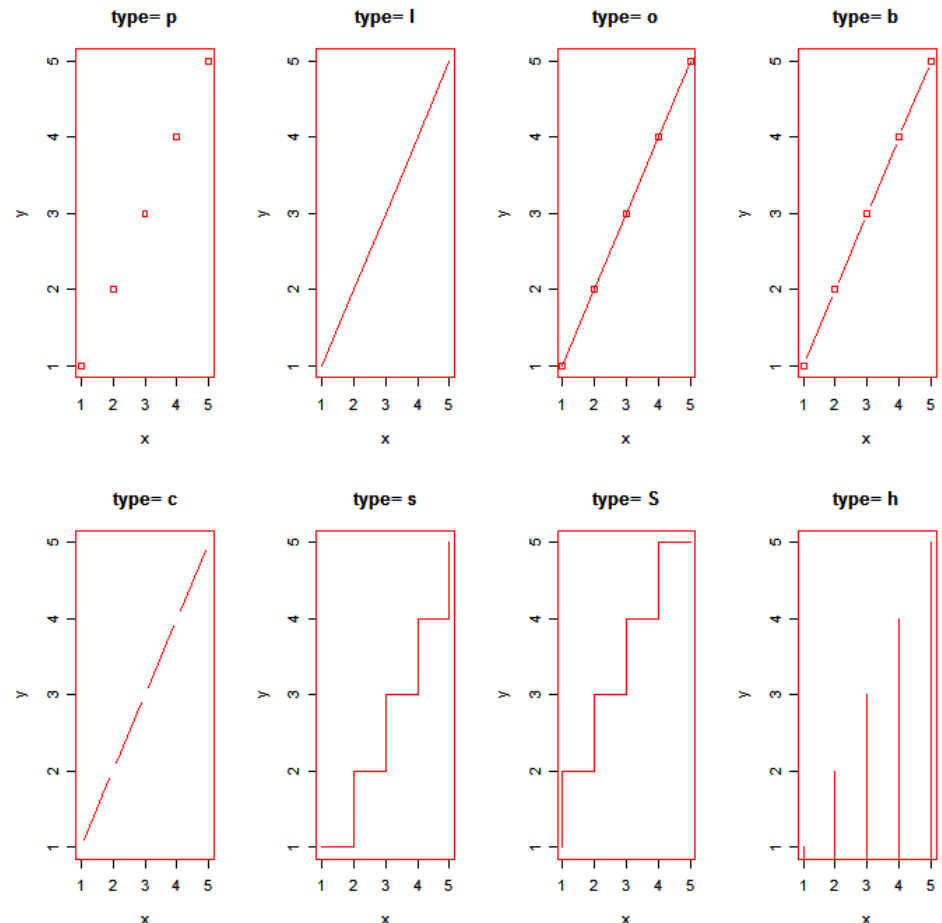
```
> cls5$methods(  
+ initialize=function() {  
+   .self$game<-"doom"  
+   .self$money<-0  
+   .self$gamelvl<-0  
+   callSuper()  
+ },  
+ showYourMoney=function() {  
+   cat("My money ", money)  
+ }  
+ )  
> x<-cls5()  
> x  
Reference class object of class "progamer"  
Field "game":  
[1] "doom"  
Field "gamelvl":  
[1] 0  
Field "money":  
[1] 0  
> x$showYourMoney()  
My money  0> |
```

Возможности базовой графики в R

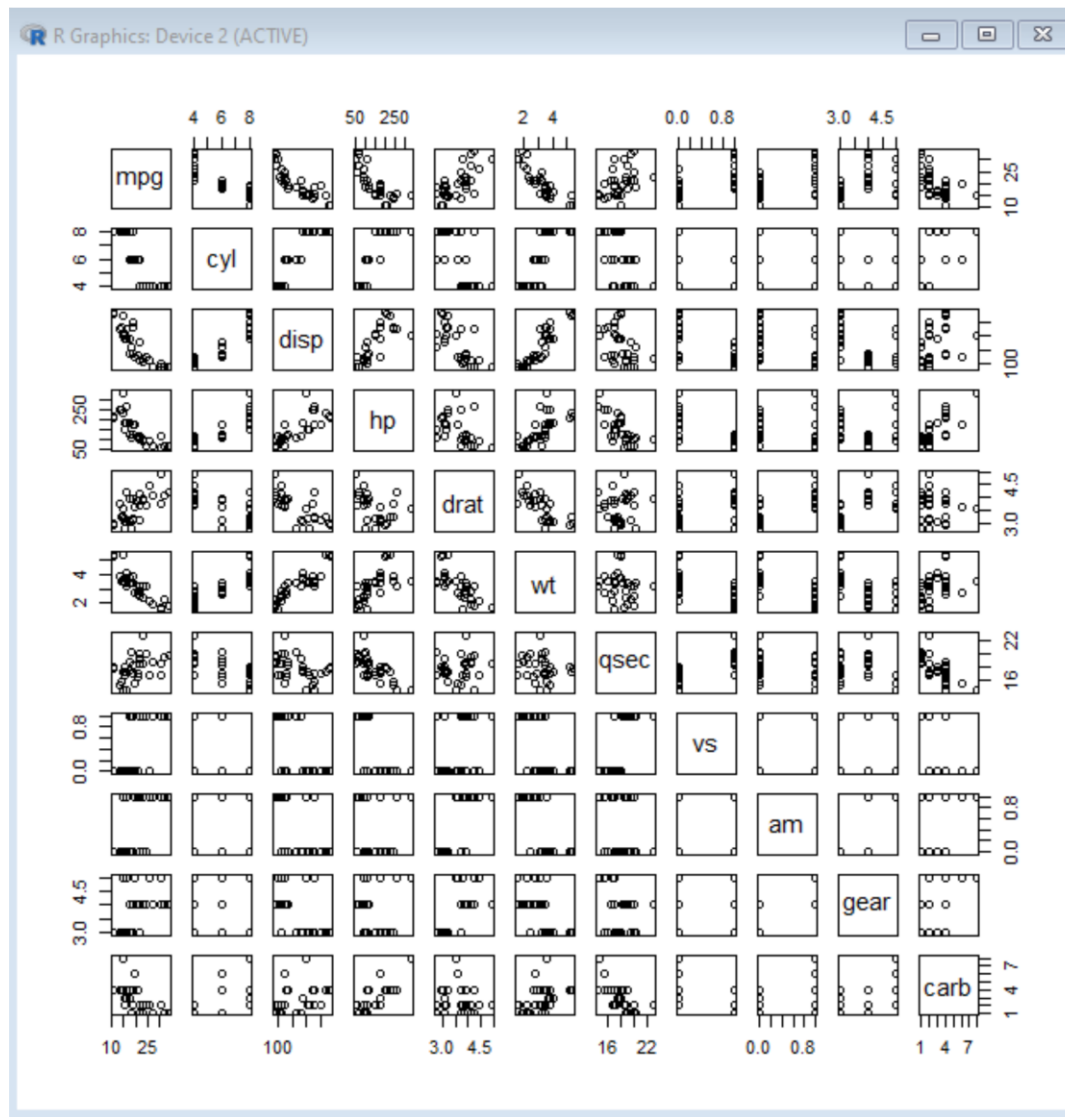
- В R много графических пакетов, мы рассмотрим базовый и ggplot
- Базовые графики
 - Scatter (разброс), series (временной ряд), step (ступенчатая диаграмма), band (с ограничениями), needle plots («игольчатый» график), bubble («пузырьковый»)
- Графики с моделями
 - Loess (локальная непараметрическая регрессия), regression (полиномиальная регрессия), spline curves (регуляризированные сплайны), ellipses (эллипсы рассеивания)
- Графики для оценки распределений
 - box plots («ящик с усами»), histograms (гистограммы), kernel density estimates (плотность распределения на основе ядерных функций)
- Графики с категориями (диаграммы)
 - Точечные, столбчатые, круговые и линейные диаграммы
- Рассмотрим базовые графические функции:
 - plot, hist, pie, boxplot, barplot, heatmap, abline и другие

Функция plot

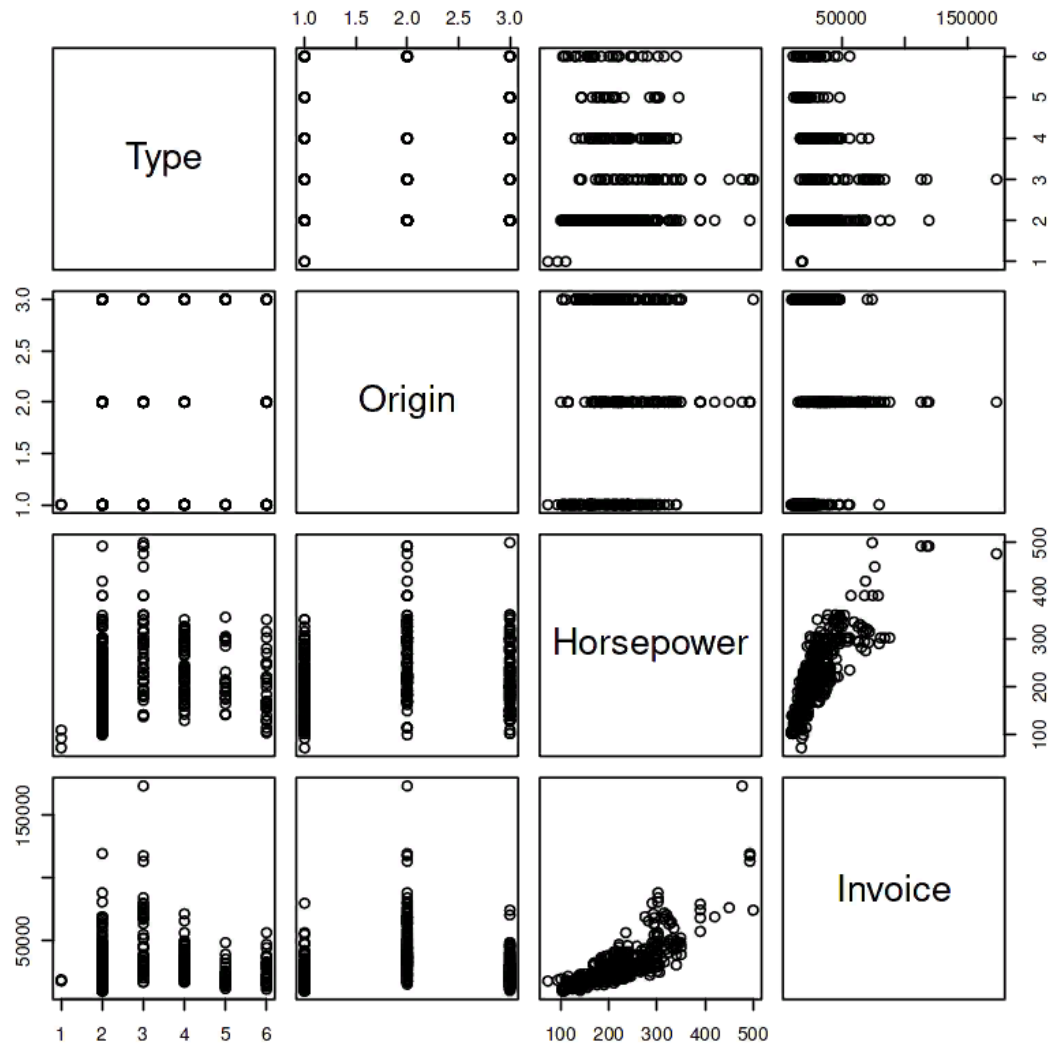
- Используется для визуализации многих типов графиков, общий синтаксис **plot(x,y,type)**
- Типы: p - точки, l - линии, b – точки и линии, c - ломаные, o – точки и ломаные, h – «гистограммы», s – ступеньки
- Можно как параметр задавать датасеты и матрицы (будут строиться попарные графики)
- Можно как параметр задавать формулы и модели



Plot датасета (тип scatter)




Построение матрицы графиков



```
> plot(cars[,c("Type", "Origin", "Horsepower", "Invoice")])
```

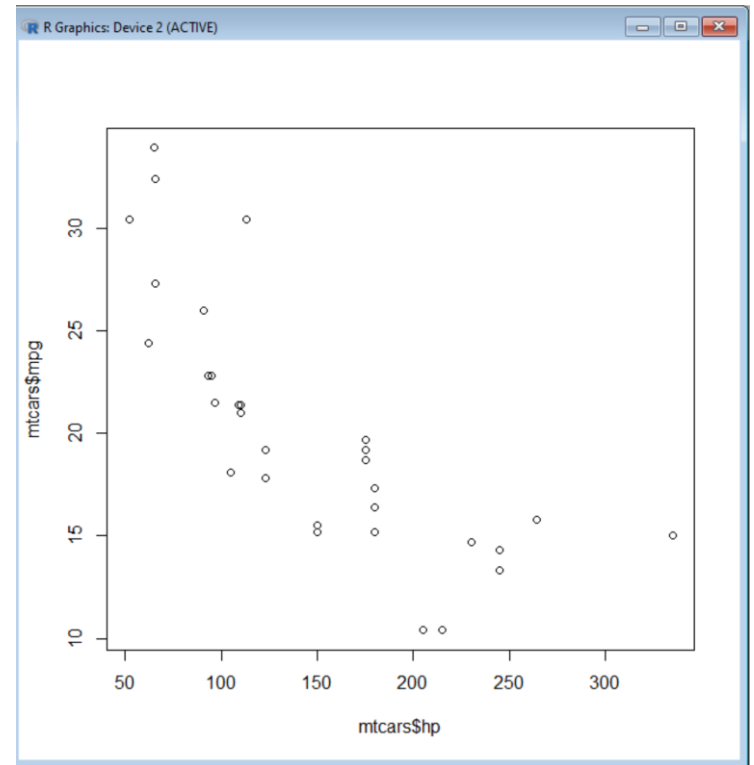
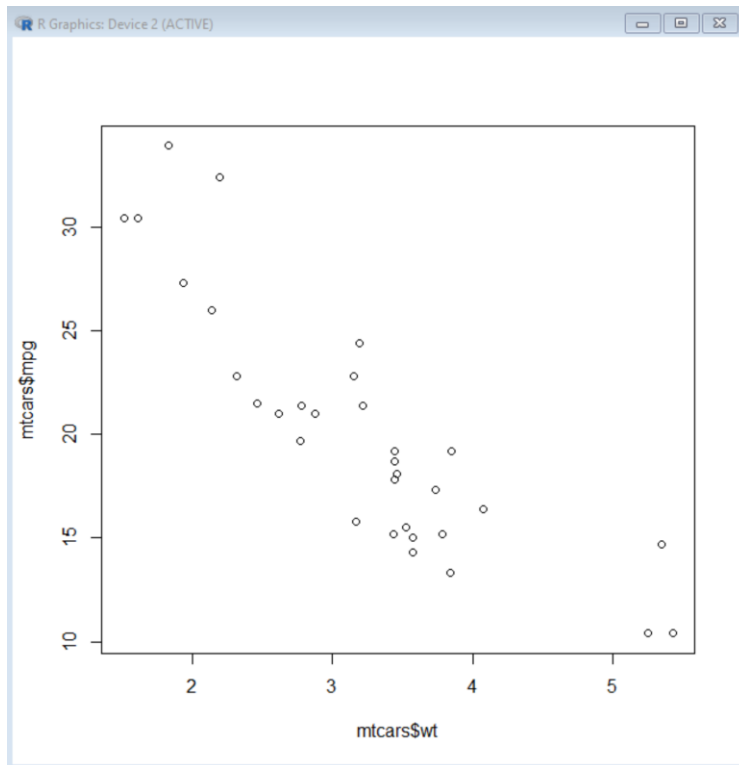
Plot формулы (тип scatter)



The screenshot shows an R Console window with the following text:

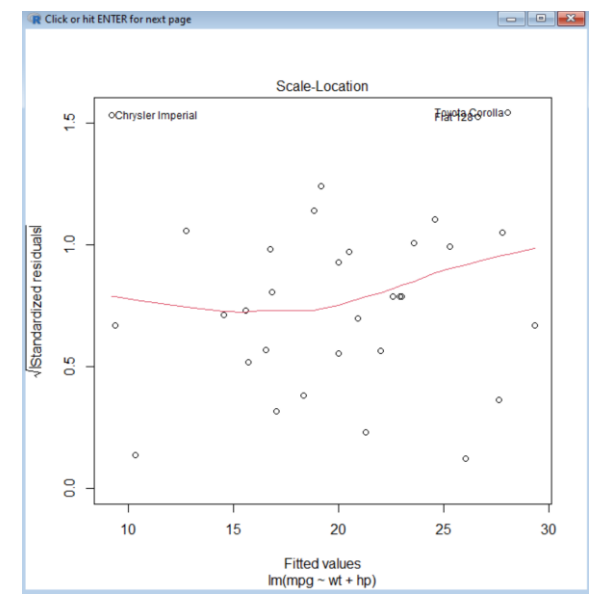
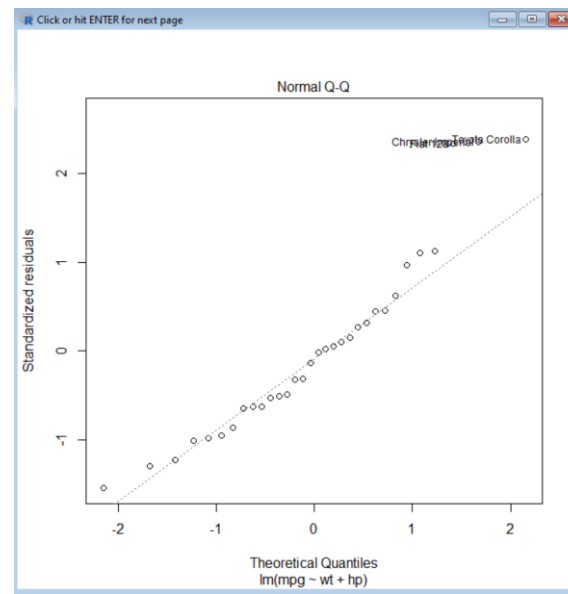
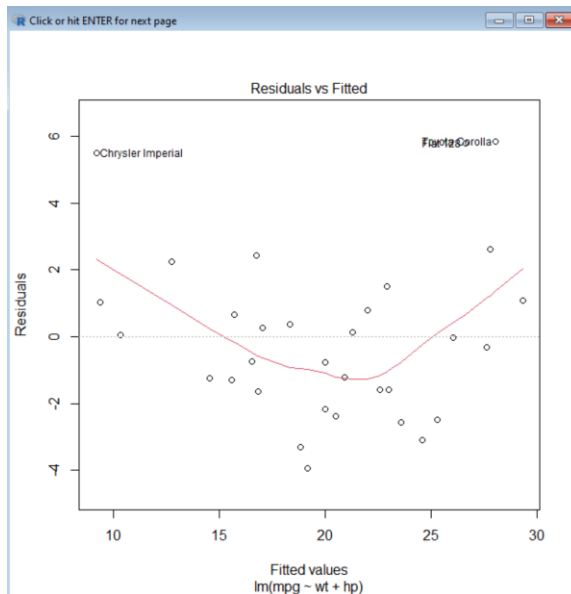
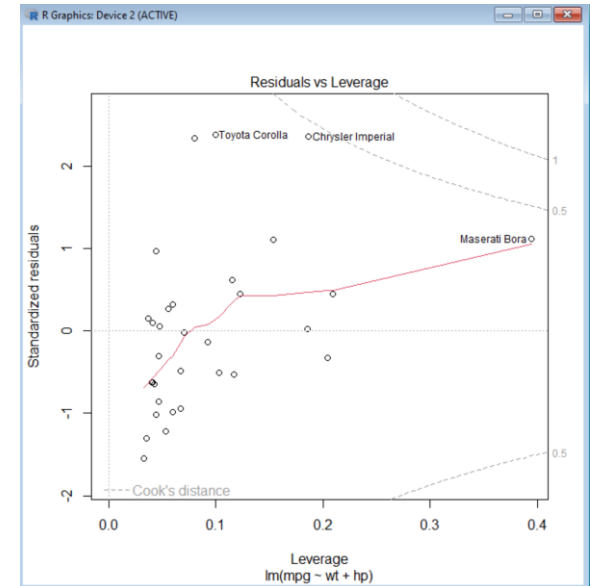
```
> plot(mtcars$mpg~mtcars$wt*mtcars$hp,data=mtcars)
Waiting to confirm page change...
Waiting to confirm page change...
> |
```

The window has a title bar that says "R Console" and standard window controls (minimize, maximize, close) on the right. The output shows that the plot command was executed, but the plot itself is not visible in this view.



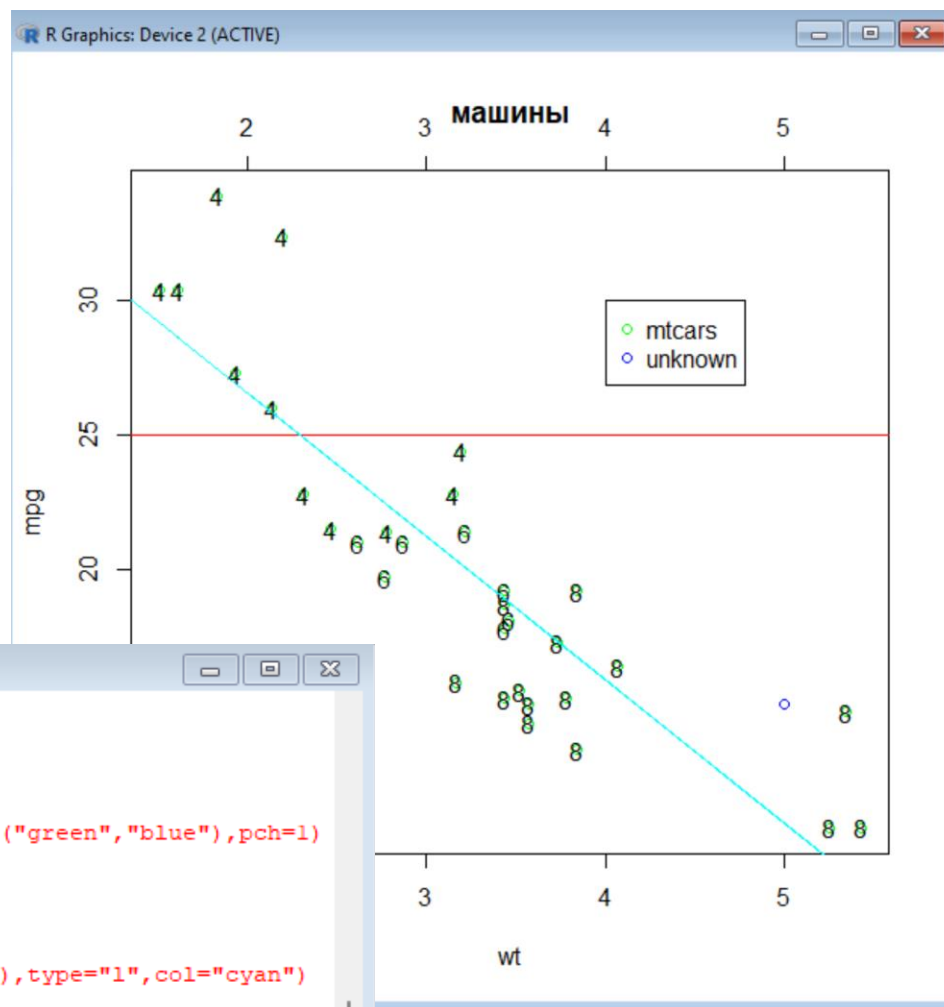
Plot модели

```
R Console
>
>
> mdl<-lm(mpg~wt+hp,data=mtcars)
> plot(mdl)
Waiting to confirm page change...
Waiting to confirm page change...
```



Дополнительные элементы на графиках

Операция	Результат
points	Точки на графике
lines, abline	Референсные линии
legend	Легенда
title	Название
axes	Оси
text	Аннотации



```
R Console

> plot(wt,mpg,type="p",col="green")
> abline(h=25, col="red")
> points(x=5,y=15,col="blue")
> legend(x=4, y=30, legend=c("mtcars", "unknown"),col=c("green","blue"),pch=1)
> title("машины")
> text(x=wt,y=mpg,cyl)
> axis(3)
> ft<-lm(mpg~wt)
> lines(order(wt),predict(ft,data.frame(wt = order(wt))),type="l",col="cyan")
> detach(mtcars)
> |
```


Графические параметры

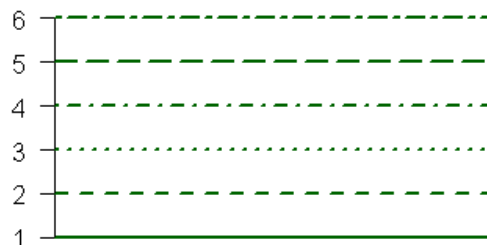
- Можно читать и задавать общие параметры для графического вывода в рамках сессии:

par(«параметр»=«значение», «параметр»=«значение» ...)

параметр	Результат
pch	Тип символа
cex, cex.axis, cex.main, ...	Размер символов элементов графика
lty	Тип линии
lwd	Толщина линии
col, col.axis, col.main, ..., fg, bg,	Цвет элементов графика
font, font.axis, font.main, ...	Шрифт элементов графика
mai, mar	Вектор для размера полей

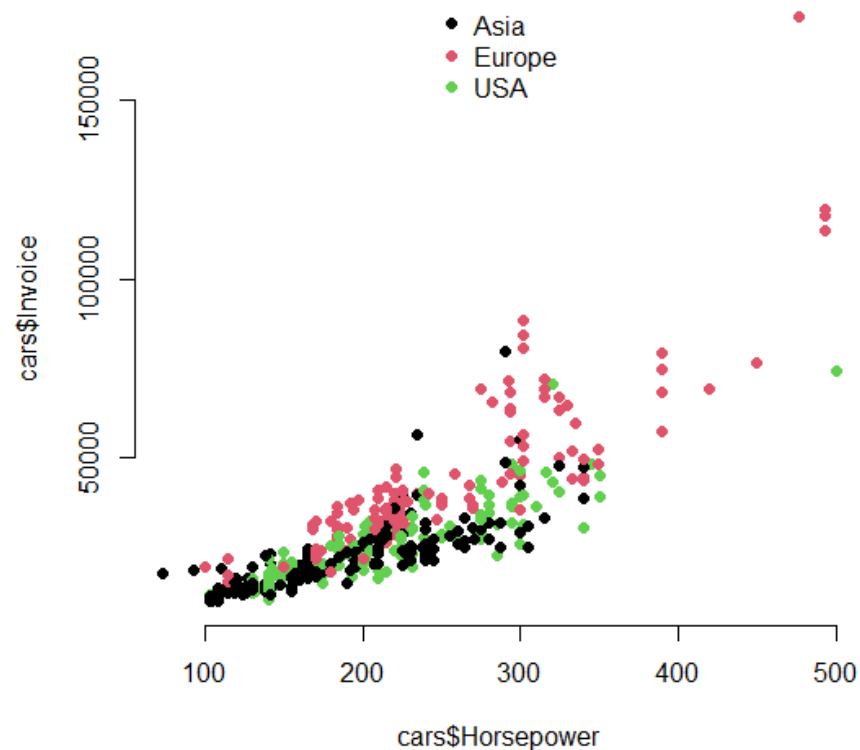
Значение
pch, также
может быть
«СВОЙ»
СИМВОЛ

Line Types: lty=



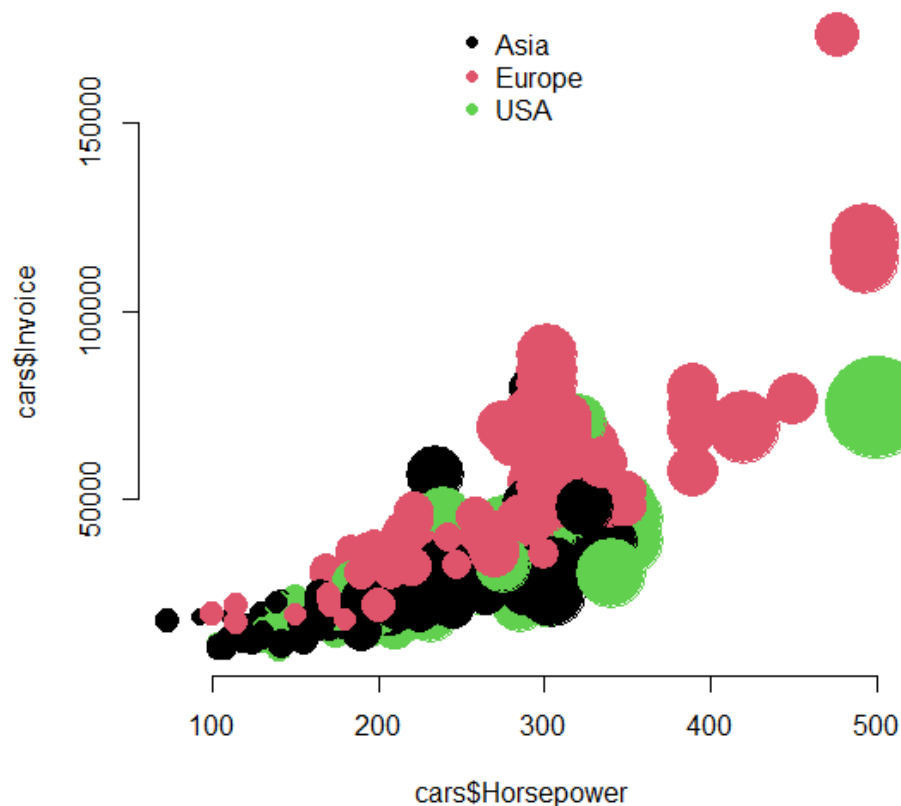
0: □	10: ⊕	20: ●	A: A
1: ○	11: ☆	21: ●	a: a
2: △	12: ⊞	22: ■	B: B
3: +	13: ⊗	23: ◆	b: b
4: ×	14: ⊠	24: ▲	S: S
5: ◇	15: ■	25: ▼	`: `
6: ▽	16: ●	@: @	.: .
7: ⊠	17: ▲	+ : +	,: ,
8: ✱	18: ◆	%: %	? : ?
9: ⬠	19: ●	#: #	*: *

SCATTER (разброс)



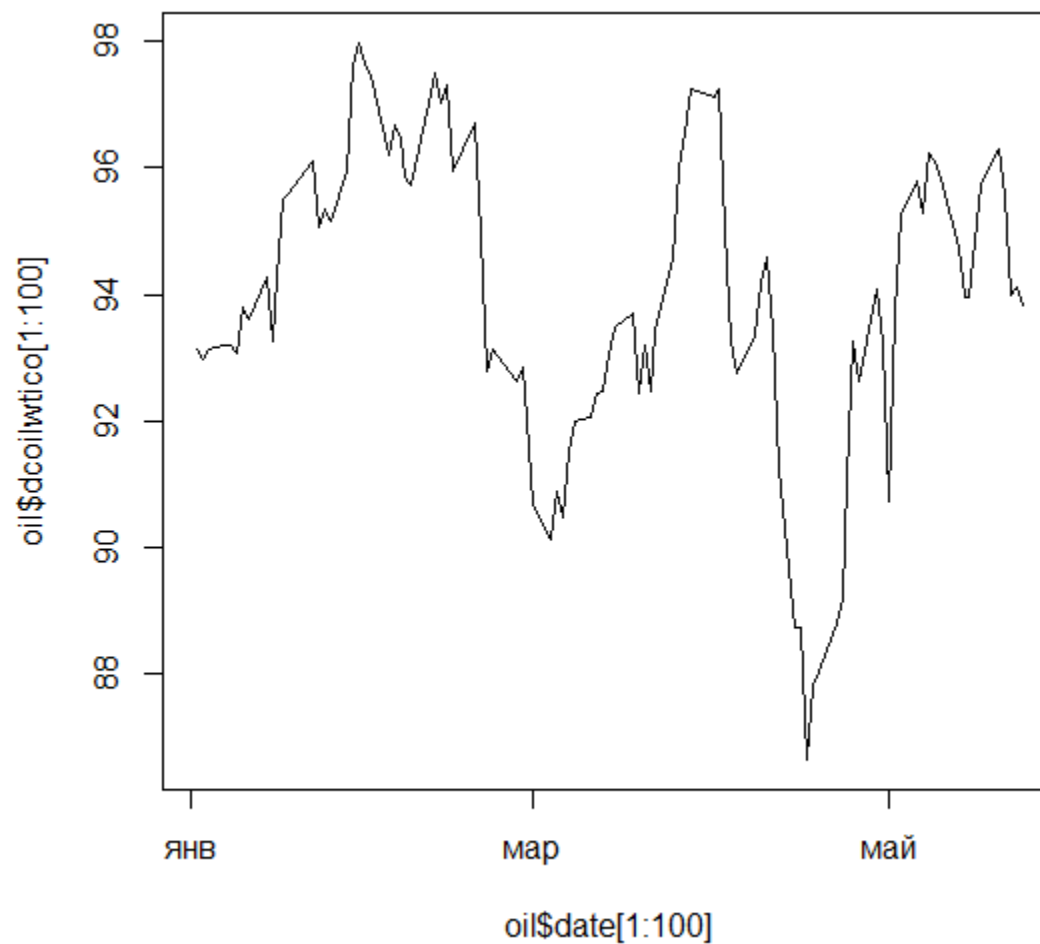
```
> plot(cars$Horsepower, cars$Invoice,  
+       pch = 19, frame = FALSE, col = factor(cars$Origin))  
>  
> legend("top", box.lty=0,  
+       legend = levels(factor(cars$Origin)),  
+       pch = 19,  
+       col = factor(levels(factor(cars$Origin))))
```

Bubble («пузырьковый» график)



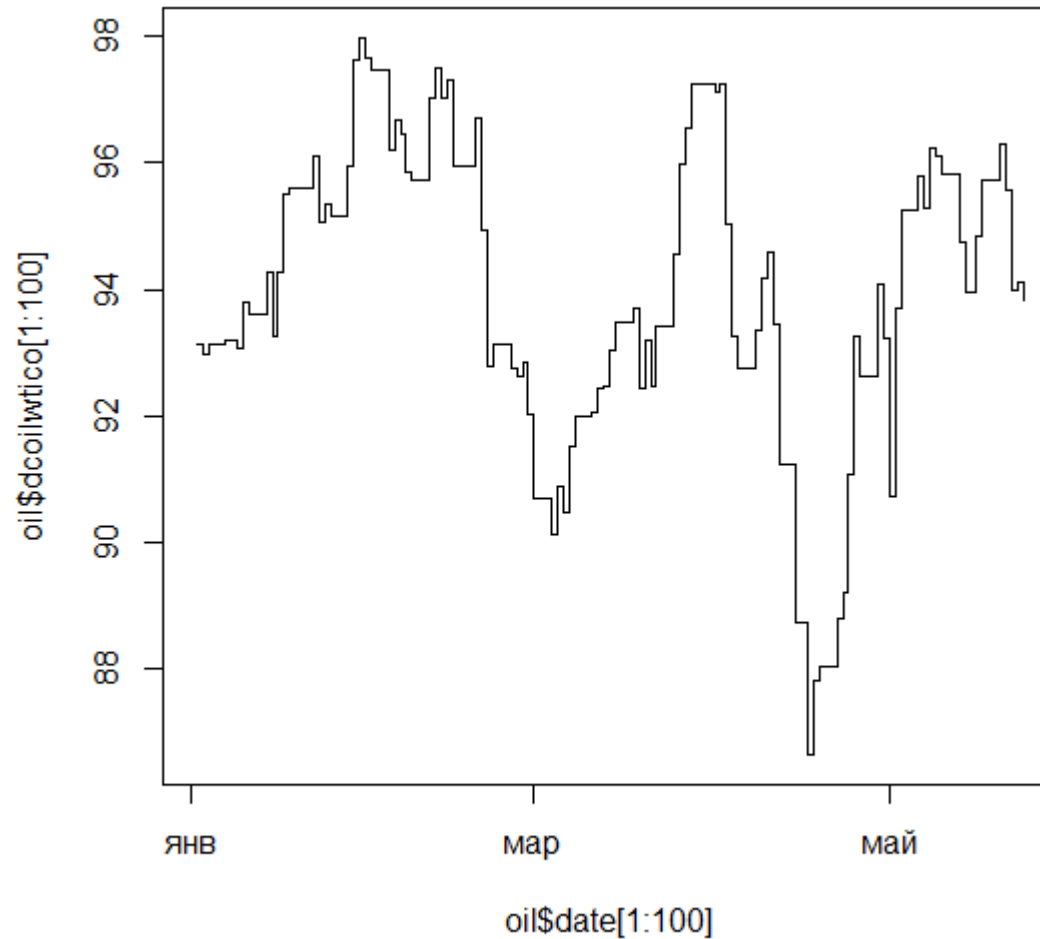
```
> plot(cars$Horsepower, cars$Invoice, cex = cars$EngineSize,  
+       pch = 19, frame = FALSE, col = factor(cars$Origin))  
>  
> legend("top", box.lty=0,  
+       legend = levels(factor(cars$Origin)),  
+       pch = 19,  
+       col = factor(levels(factor(cars$Origin))))
```

SERIES (временной ряд)



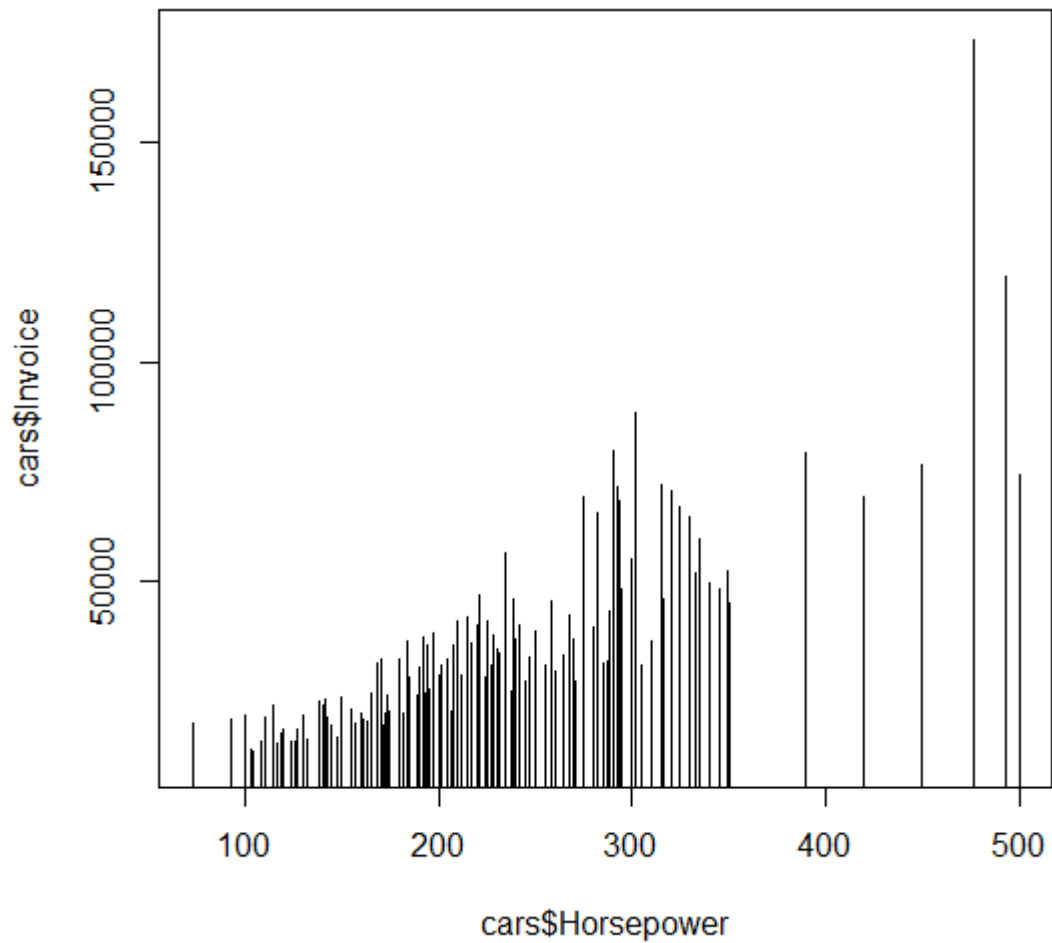
```
> plot(oil$date[1:100], oil$dcoilwtico[1:100], type = "l")
```

STEP (ступенчатый)



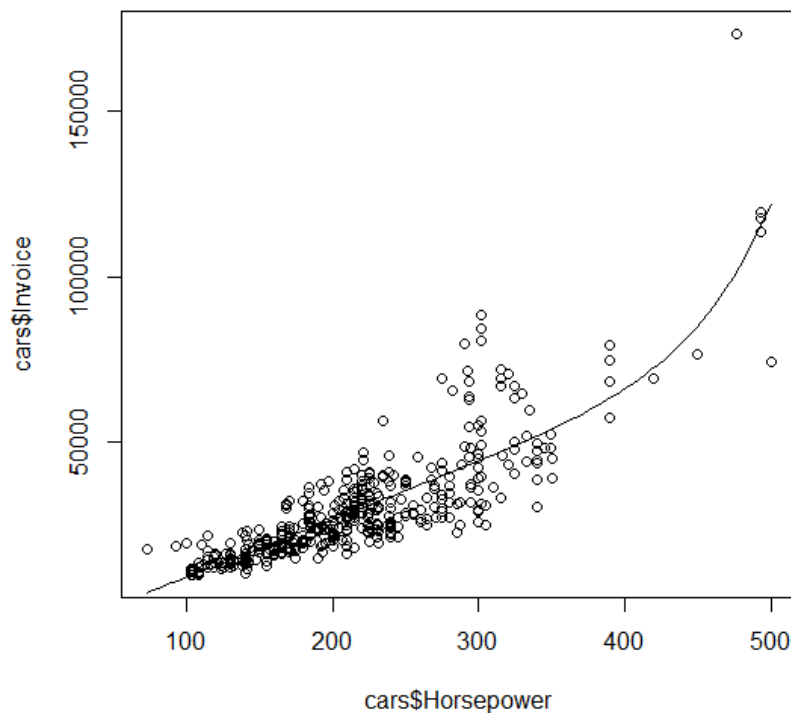
```
> plot(oil$date[1:100], oil$dcoilwtico[1:100], type = "s")
```

NEEDLE («игольчатый»)



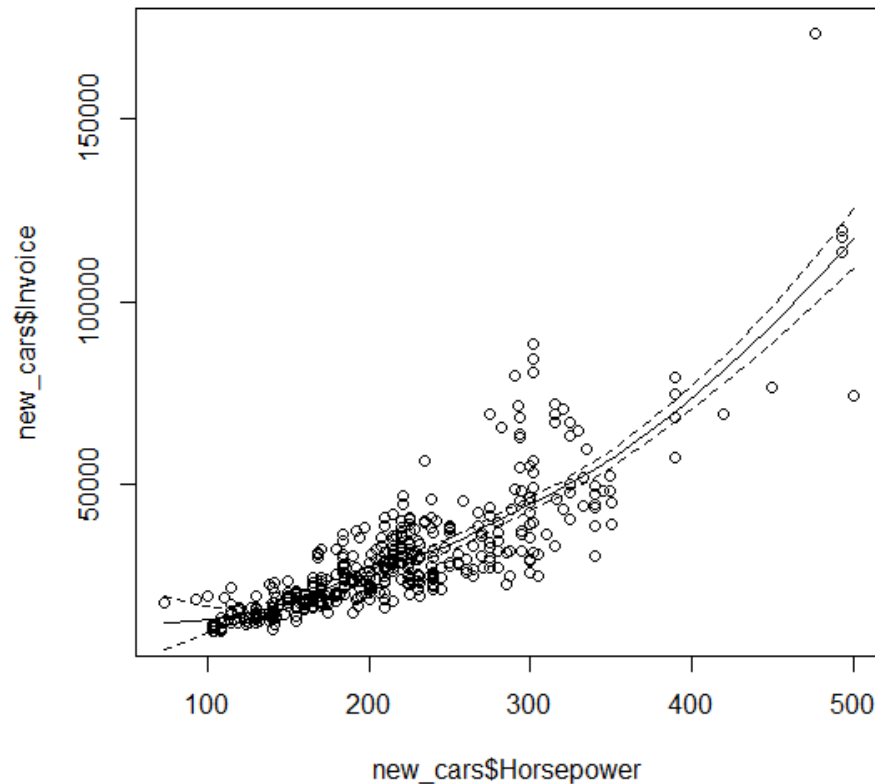
```
> plot(cars$Horsepower, cars$Invoice, type = "h")
```

REG (с полиномиальной регрессией)



```
> fit <- lm(Invoice ~ Horsepower + I(Horsepower^10), data = cars)
> new_cars <- data.frame(Horsepower = seq(min(cars$Horsepower),
+                                     max(cars$Horsepower),
+                                     length.out = 100))
> new_cars$pred <- predict(fit, newdata = new_cars)
>
> plot(cars$Invoice ~ cars$Horsepower)
> with(new_cars, lines(x = Horsepower, y = pred))
```

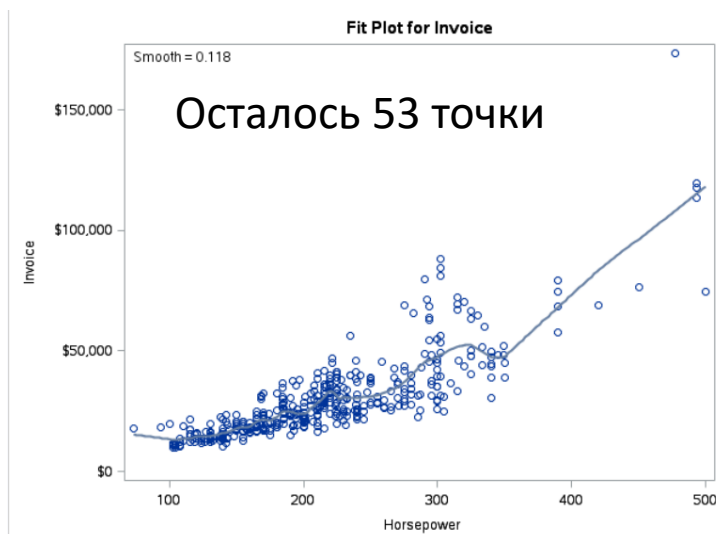
LOESS (локальная непараметрическая регрессия)



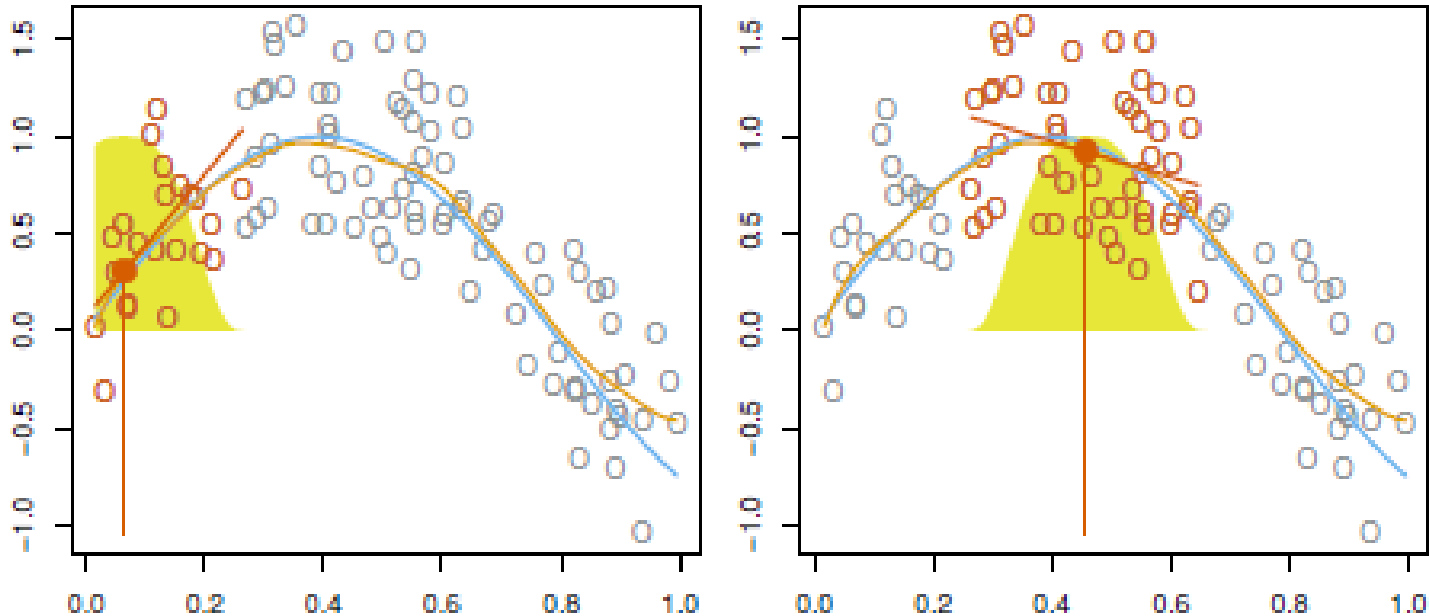
```
> new_cars <- cars[order(cars$Horsepower), ]  
> plot(new_cars$Invoice ~ new_cars$Horsepower)  
> plx<-predict(loess(new_cars$Invoice ~ new_cars$Horsepower), se=T)  
>  
> lines(new_cars$Horsepower,plx$fit)  
> lines(new_cars$Horsepower,plx$fit - qt(0.975,plx$df)*plx$se, lty=2)  
> lines(new_cars$Horsepower,plx$fit + qt(0.975,plx$df)*plx$se, lty=2)
```


Локальная непараметрическая регрессия

- Для каждого t -го наблюдения подгоняется локальная регрессионная модель $y_t = g(x_t) + \varepsilon_t, t = 1, \dots, m$
 - с заданным уравнением $g(\cdot)$ (обычно линейная или кубическая регрессия)
 - по «окружению» этого наблюдения, где размер окружения задается как параметр - пропорция от всей выборки
 - Используется взвешенный МНК, где при подгонке вес наблюдения из «окружения» определяются расстоянием до «центрального» наблюдения, например:
 - Можно задать $w(x) = (1 - |x|^3)^3 I[|x| \leq 1]$ а параметра сглаживания (AIC, CV, DF, ...)



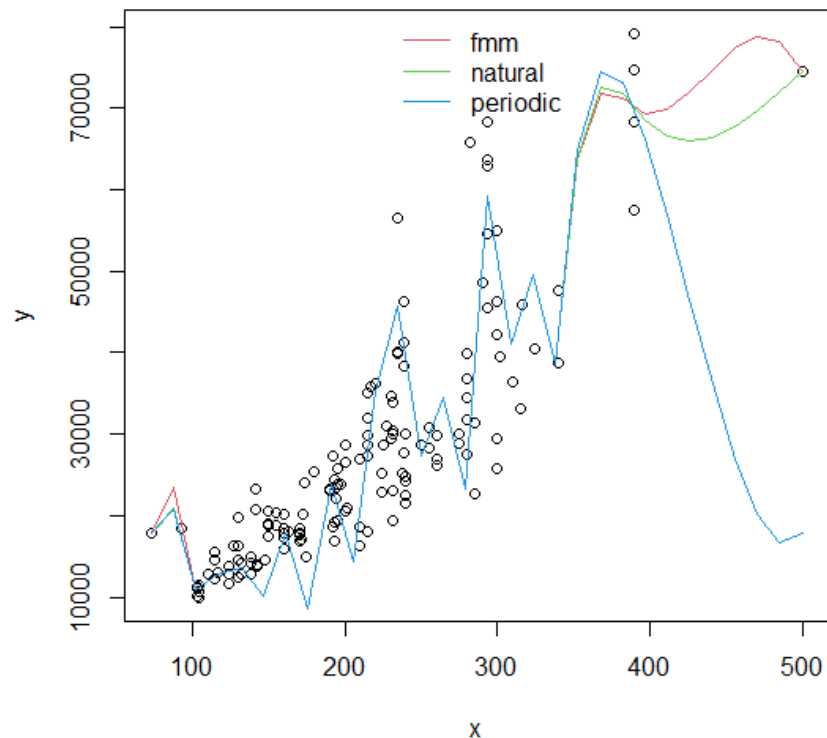
Локальная взвешенная регрессия



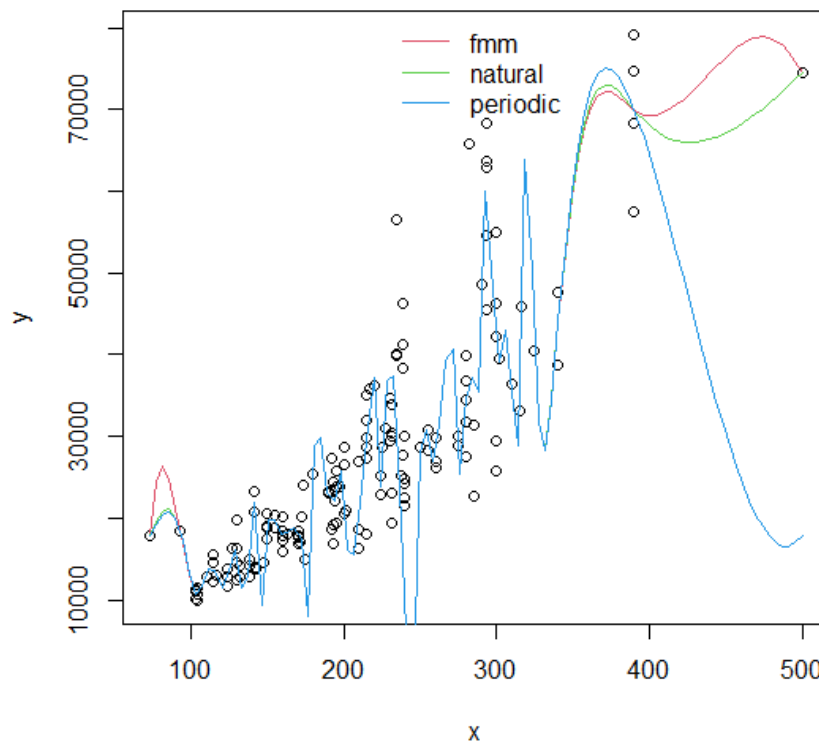
- С помощью скользящей весовой функции отдельно подгоняем линейные участки по диапазону X с помощью взвешенных наименьших квадратов
- Параметр регуляризации подбирается кросс-валидацией или через информационные критерии

SPLINE (интерполяция сплайнами)

n = 30



n = 100



```
> x <- cars$Horsepower[100:250]
> y <- cars$Invoice[100:250]
> plot(x, y)
> lines(spline(x, y, n = 30), col = 2)
> lines(spline(x, y, n = 30, method = "natural"), col = 3)
> lines(spline(x, y, n = 30, method = "periodic"), col = 4)
> legend("top", c("fmm", "natural", "periodic"),
+       col = 2:4, lty = 1, box.lty=0)
```

Сплаины

Кубические сплайны с узлами ξ_k , $k = 1, \dots, K$ представляют собой кусочно-кубический многочлен с непрерывными производными до второго порядка в каждом узле.

Мы можем представить эту модель со степенными базисными функциями

$$y_i = \beta_0 + \beta_1 b_1(x_i) + \beta_2 b_2(x_i) + \dots + \beta_{K+3} b_{K+3}(x_i) + \epsilon_i,$$

$$b_1(x_i) = x_i$$

$$b_2(x_i) = x_i^2$$

$$b_3(x_i) = x_i^3$$

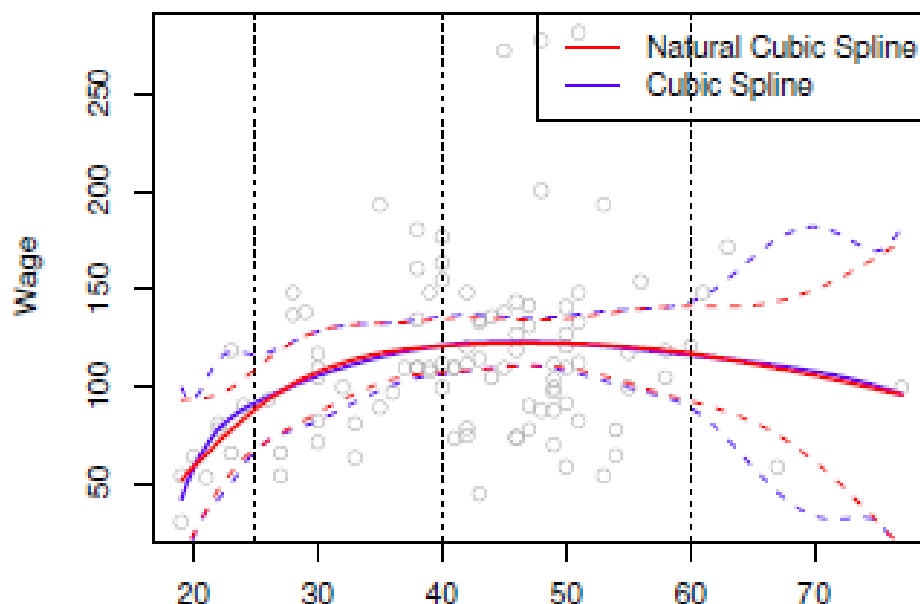
$$b_{k+3}(x_i) = (x_i - \xi_k)_+^3, \quad k = 1, \dots, K$$

где

$$(x_i - \xi_k)_+^3 = \begin{cases} (x_i - \xi_k)^3 & \text{if } x_i > \xi_k \\ 0 & \text{otherwise} \end{cases}$$

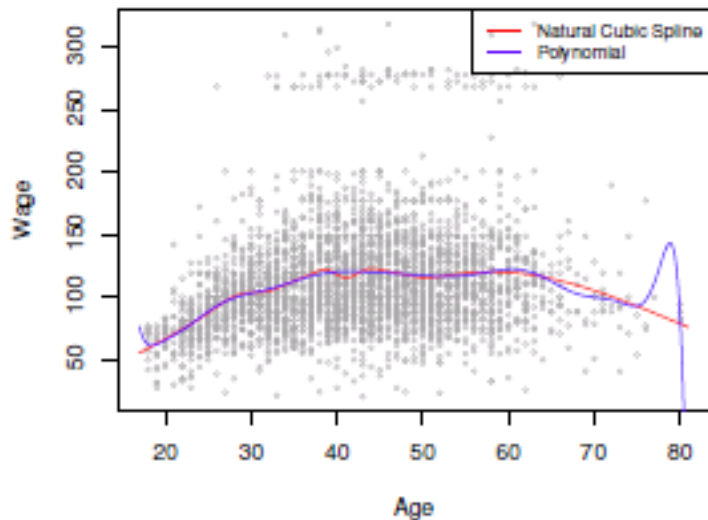
Естественные кубические сплайны

Естественный кубический сплайн осуществляет линейную экстраполяцию за граничные узлы. Это добавляет $4 = 2 * 2$ дополнительных ограничения и позволяет нам делать больше внутренних узлов для тех же степеней свободы, по сравнению с обычным кубическим сплайном.



Размещение узлов

- Одна из стратегий состоит в том, чтобы определить значение K (количество узлов), а затем поместить их в соответствующие квантили наблюдаемого X .
- Кубический сплайн с K узлами имеет $K + 4$ параметров или степеней свободы.
- Естественный сплайн с K узлами имеет K степеней свободы.



Сравнение полинома степени 14 и естественного кубического сплайна, каждый с 15.

Сглаживание сплайнами

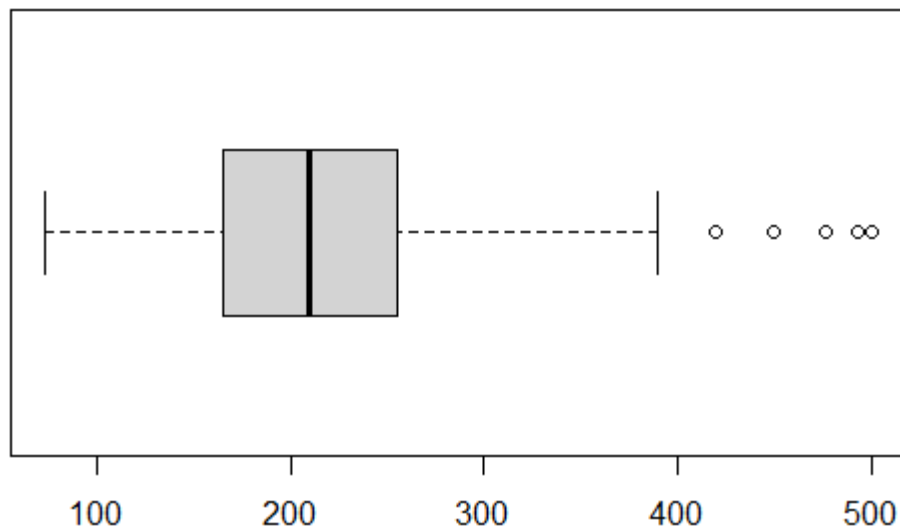
Подгонка гладкой функцией $g(x)$:

$$\underset{g \in \mathcal{S}}{\text{minimize}} \sum_{i=1}^n (y_i - g(x_i))^2 + \lambda \int g''(t)^2 dt$$

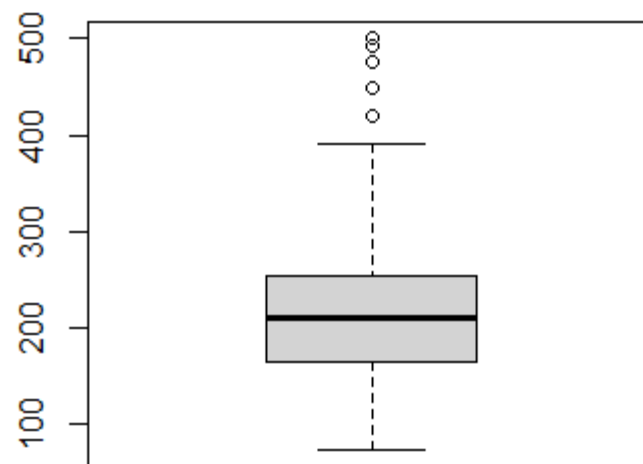
- Первое слагаемое - RSS и он нацелен на то, чтобы $g(x)$ соответствовала данным в каждом x_i .
- Второе слагаемое - это штраф за *грубое приближение* и он управляет тем, насколько $g(x)$ «извилистая». Он варьируется *параметром настройки* $\lambda \geq 0$.
 - Чем меньше, тем более извилистая функция, в конечном счете интерполирующая y_i когда $\lambda = 0$.
 - Когда $\lambda \rightarrow \infty$, функция $g(x)$ становится линейной.

НВОХ, VBOX («ящик с усами»)

boxplot horizontal

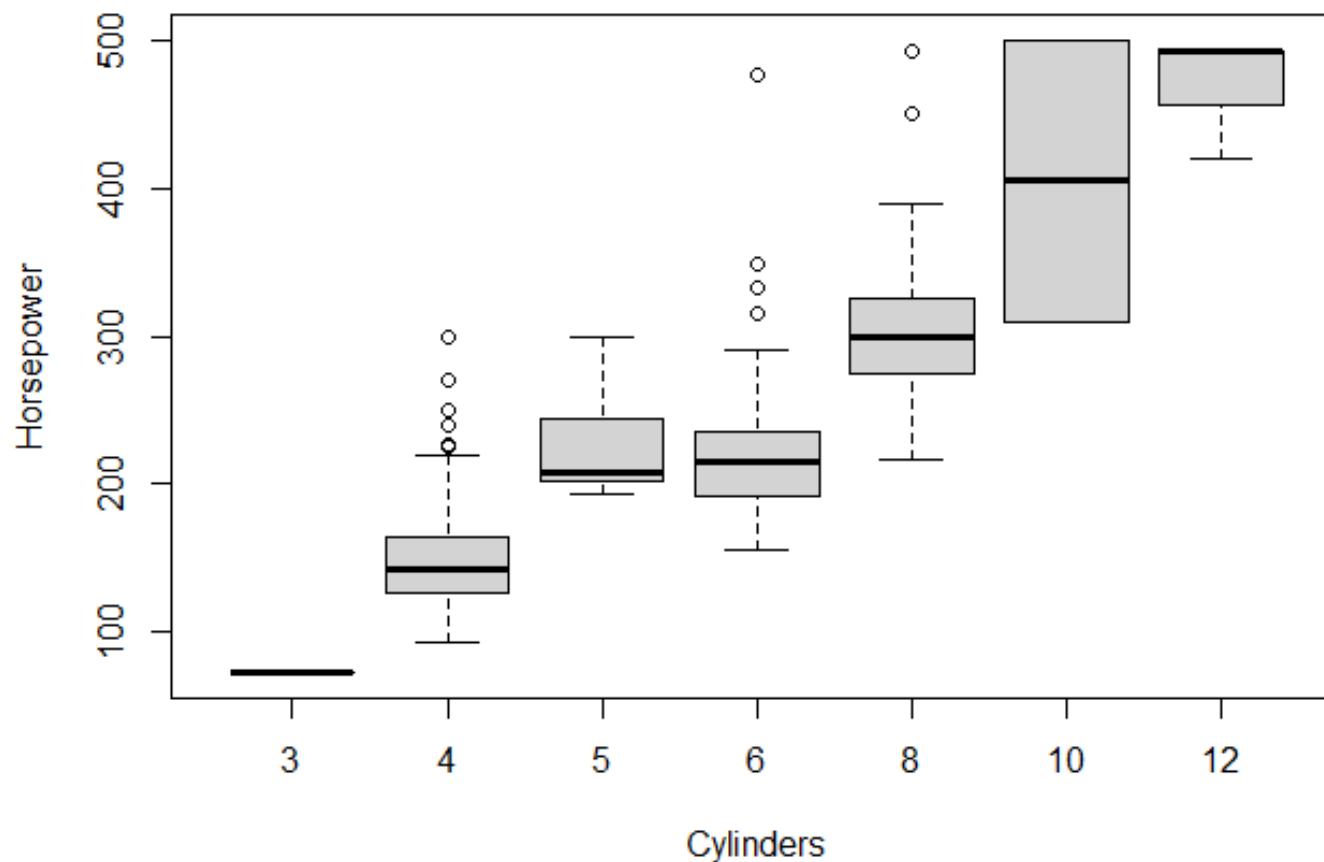


boxplot vertical



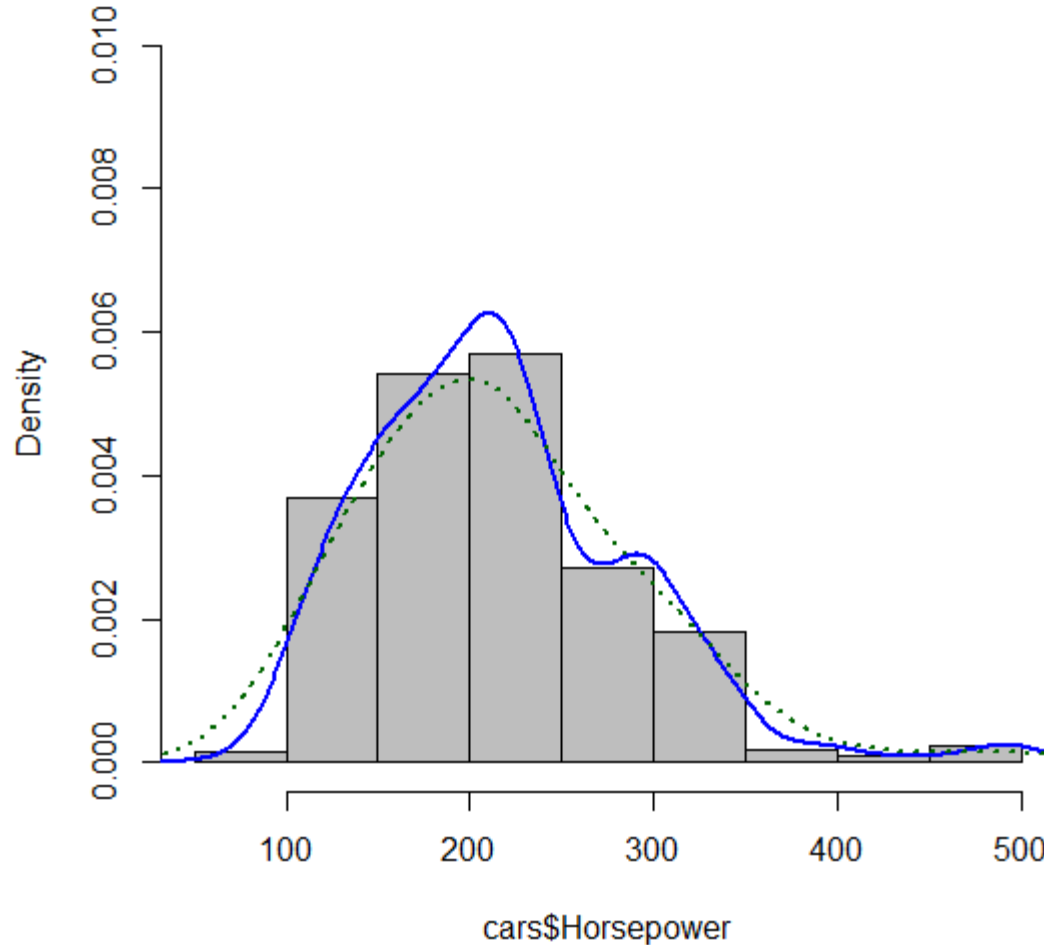
```
> boxplot(new_cars$Horsepower, main = "boxplot horizontal", horizontal = TRUE)
> boxplot(new_cars$Horsepower, main = "boxplot vertical", horizontal = FALSE)
```


НВОХ, VBOX («ящик с усами»)



```
> boxplot(Horsepower ~ Cylinders, data = cars)
```

Гистограмма и плотность



```
> hist(cars$Horsepower, prob=TRUE, ylim = c(0, 0.01), col="grey")  
> lines(density(cars$Horsepower), col="blue", lwd=2)  
> lines(density(cars$Horsepower, adjust=2), lty="dotted", col="darkgreen", lwd=2)
```

Оценка плотности распределения с помощью ядерных функций

- Оценка плотности распределения:

$$\hat{f}_h(x) = \frac{1}{n} \sum_{i=1}^n K_h(x - x_i) = \frac{1}{nh} \sum_{i=1}^n K\left(\frac{x - x_i}{h}\right),$$

- где K - ядерная функция (положительно-определенная функция), такая что $K(x)=K(-x)$ и:

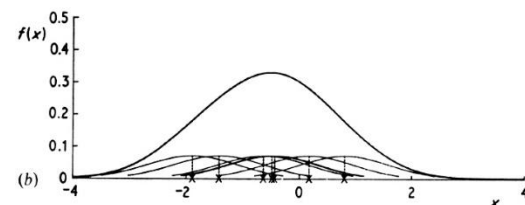
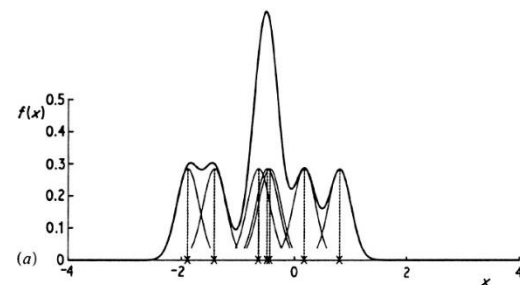
$$\int_{-\infty}^{\infty} K(x)dx = 1.$$

- например

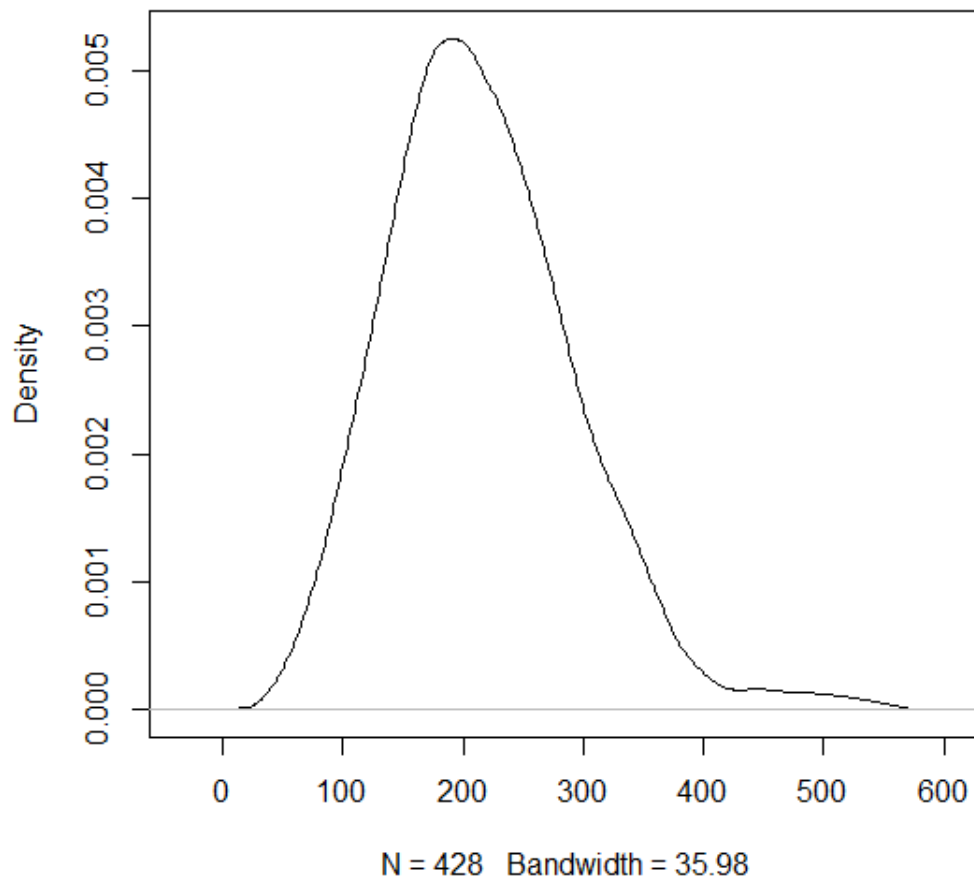
$$K(u) = \frac{1}{\sqrt{2\pi}} e^{-\frac{1}{2}u^2}$$

- x_i – значения из выборки,
- h – параметр сглаживания,
автоматический выбор h по MISE

$$\text{MISE}(\lambda) = \int_x \{E(\hat{f}_\lambda(x)) - f(x)\}^2 dx + \int_x \text{var}(\hat{f}_\lambda(x)) dx$$

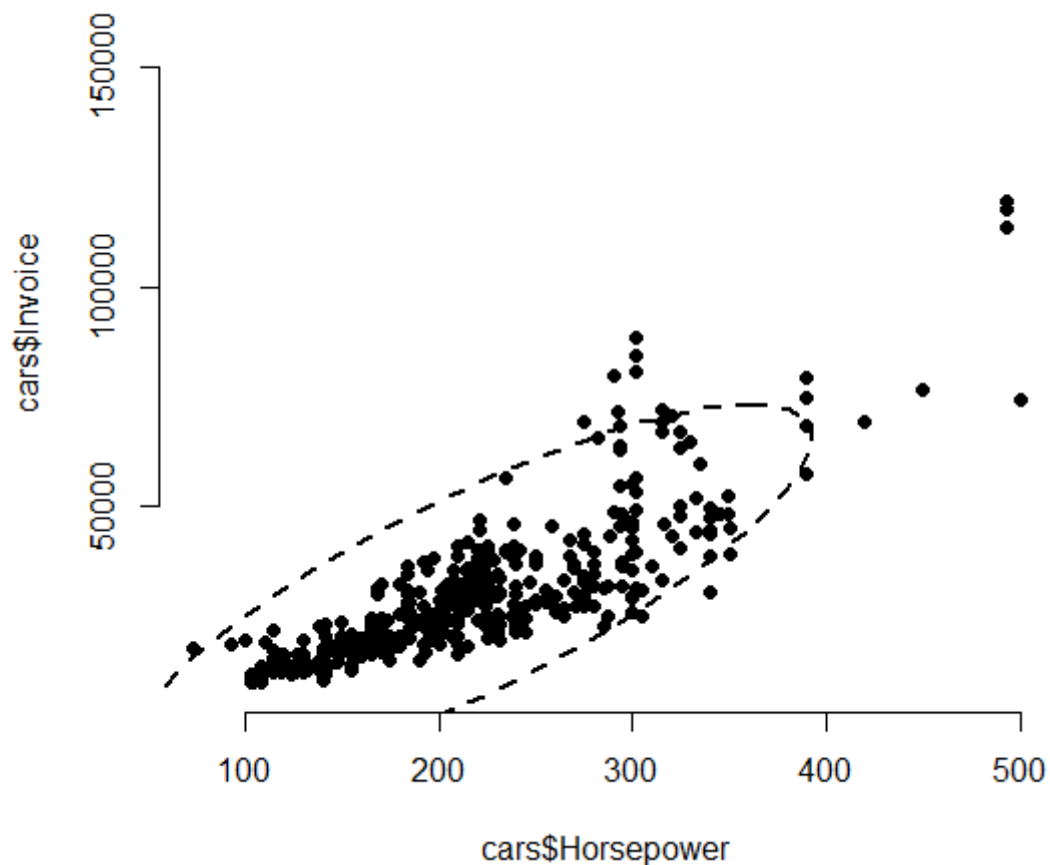


Оценка плотности распределения с помощью ядерных функций



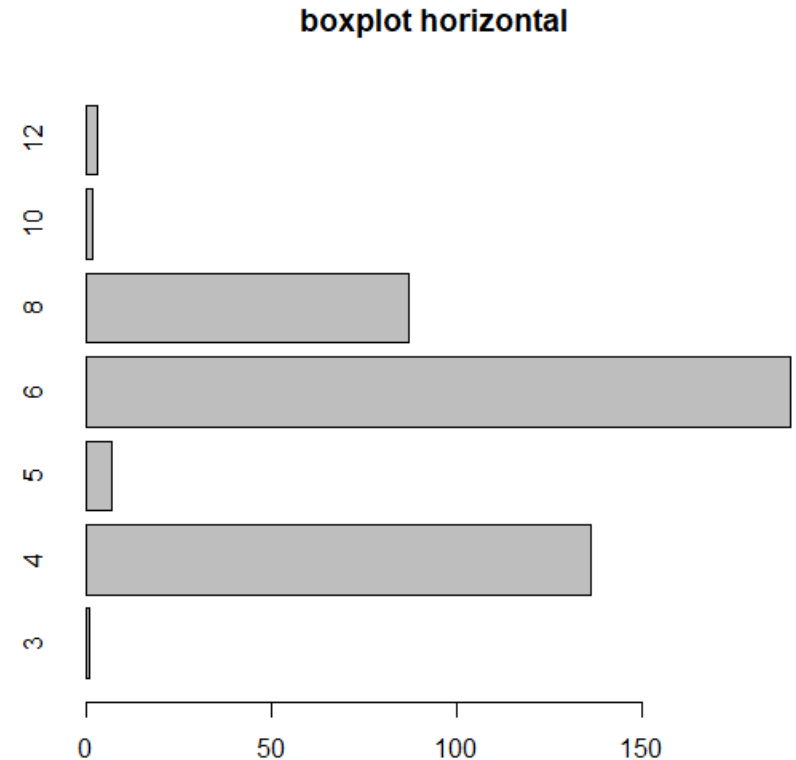
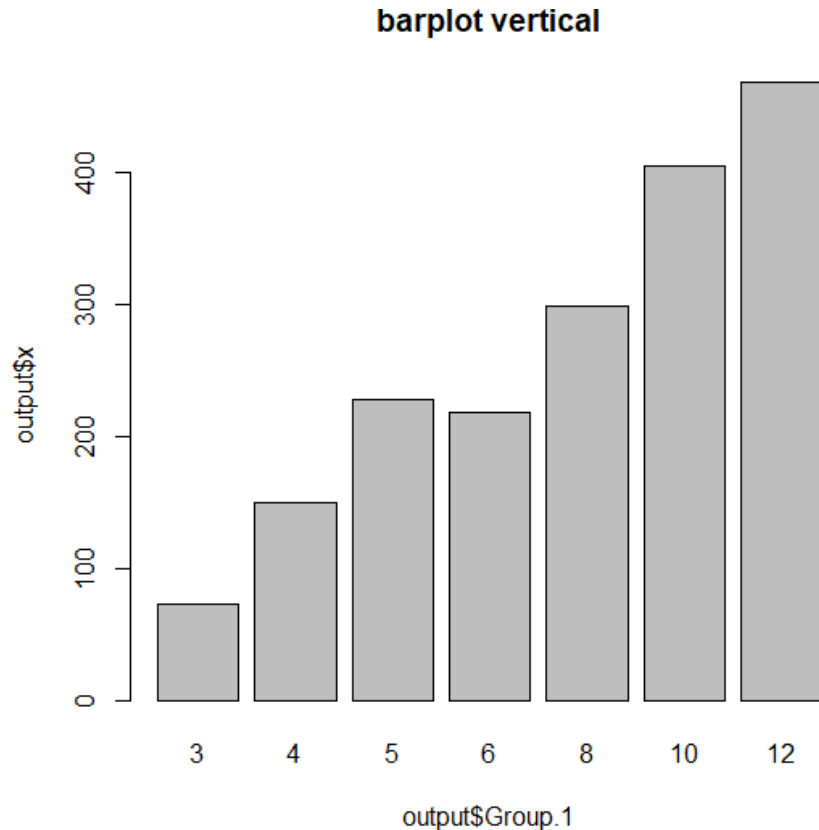
```
> plot(density(cars$Horsepower, adjust=2, kernel = "epanechnikov"))
```

Эллипс разброса



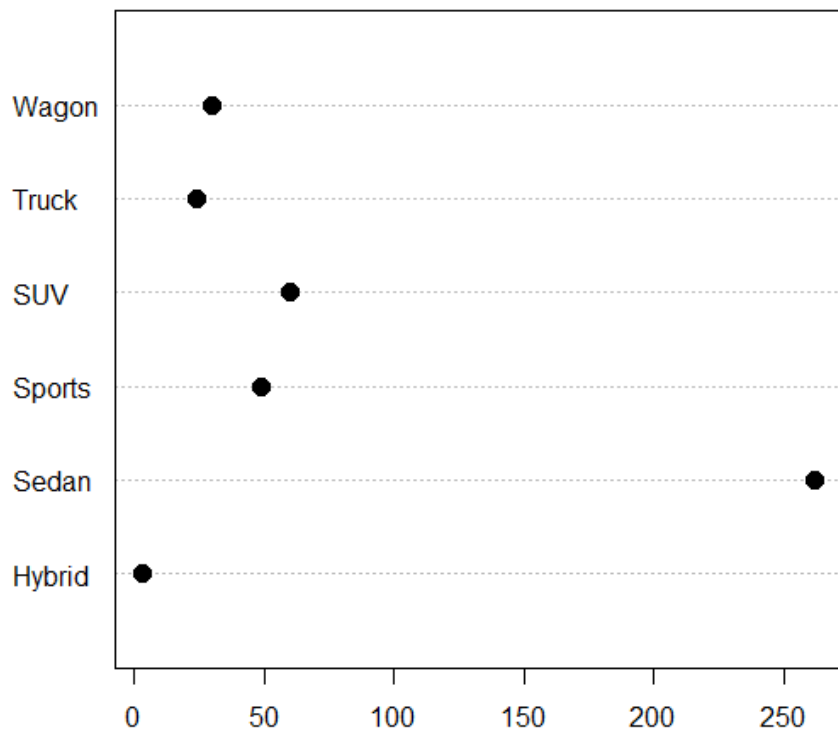
```
> plot(cars$Horsepower, cars$Invoice,  
+       pch = 19, frame = FALSE)  
> dataEllipse(cars$Horsepower, cars$Invoice, levels=0.95,  
+             lty=2, col = "black", add=TRUE)
```

HBAR, VBAR, HLINE, VLINE (столбчатые и линейные диаграммы)

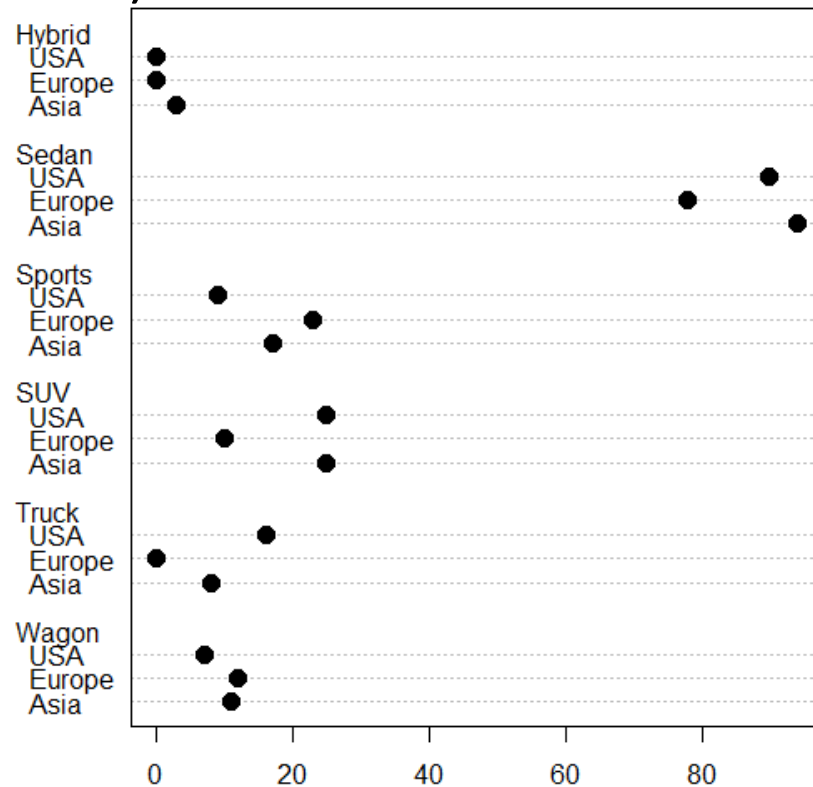


```
> output <- aggregate(cars$Horsepower,  
+                     by=list(cars$Cylinders),  
+                     FUN=mean)  
>  
> barplot(output$x ~ output$Group.1, main = "barplot vertical", horiz = FALSE)  
> barplot(table(cars$Cylinders), main = "boxplot horizontal", horiz = TRUE)
```

DOT (точечная диаграмма)



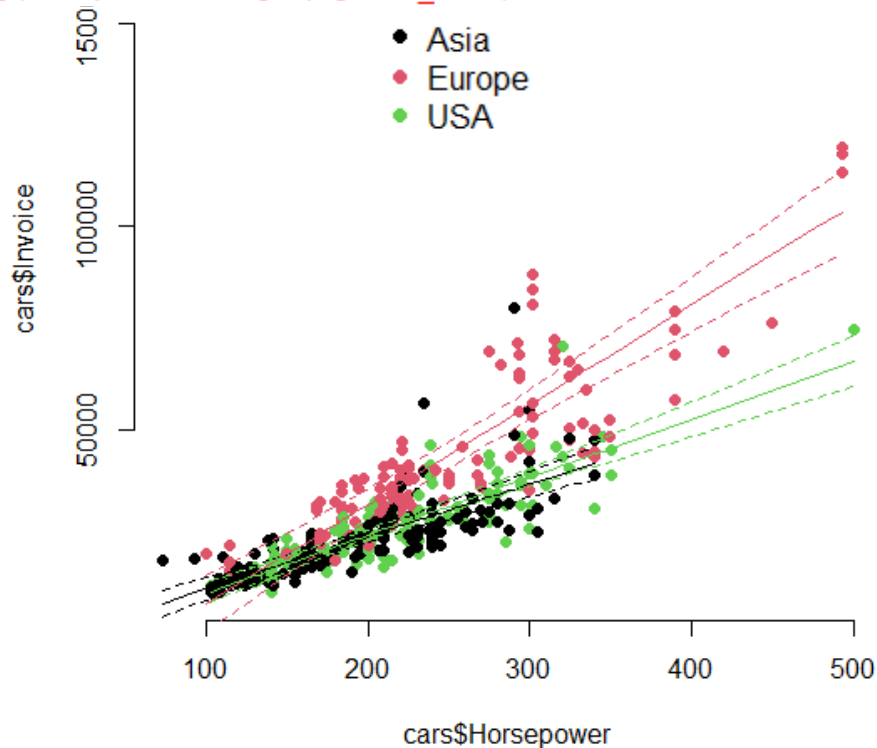
```
> dotchart(table(cars$Type),
+           pch = 19,
+           pt.cex = 1.5,
+           frame.plot = TRUE)
```



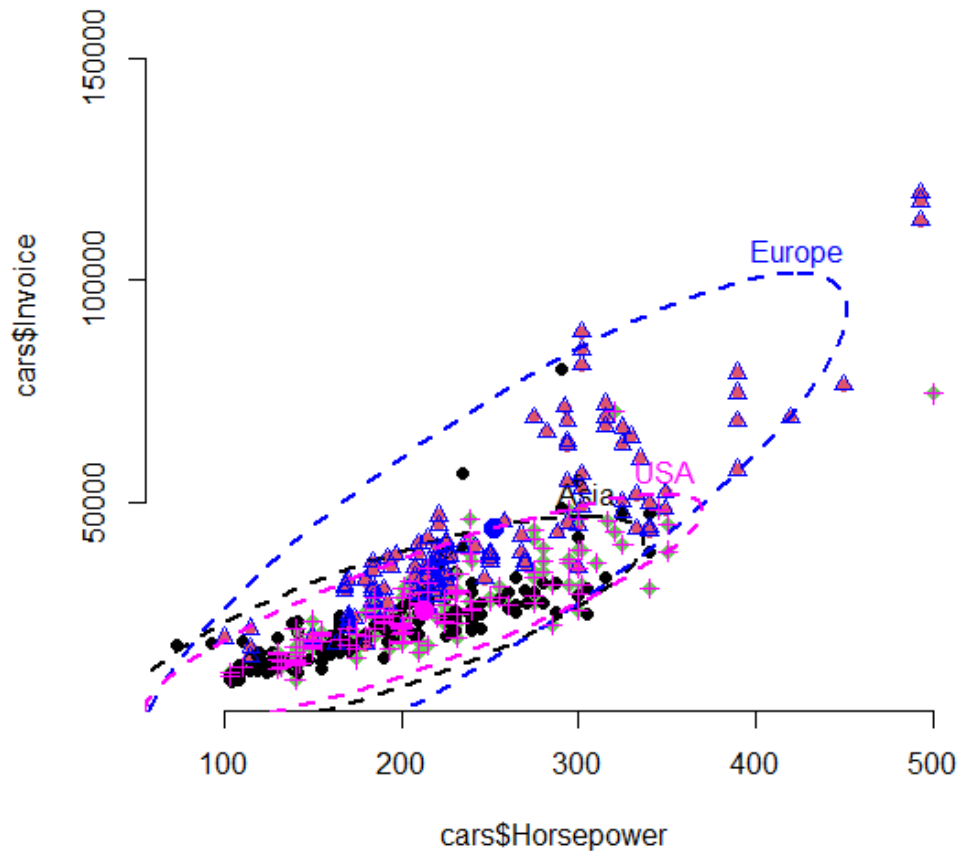
```
> count <- table(cars$Origin, cars$Cylinders)
>
> dotchart(count,
+           pch = 19,
+           pt.cex = 1.5,
+           frame.plot = TRUE)
```

Разброс с моделью

```
> print_line <- function(x) {  
+   x <- x[order(x$Horsepower), ]  
+   plx <- predict(lm(x$Invoice ~ x$Horsepower), se=T)  
+   lines(x$Horsepower,plx$fit, col = x$col)  
+   lines(x$Horsepower,plx$fit + 3*plx$se, lty=2, col = x$col)  
+   lines(x$Horsepower,plx$fit - 3*plx$se, lty=2, col = x$col)  
+ }  
>  
> cars$col <- factor(cars$Origin)  
> plot(cars$Horsepower, cars$Invoice,  
+     pch = 19, frame = FALSE, col = cars$col)  
>  
> by(cars, cars$Origin, print_line)
```

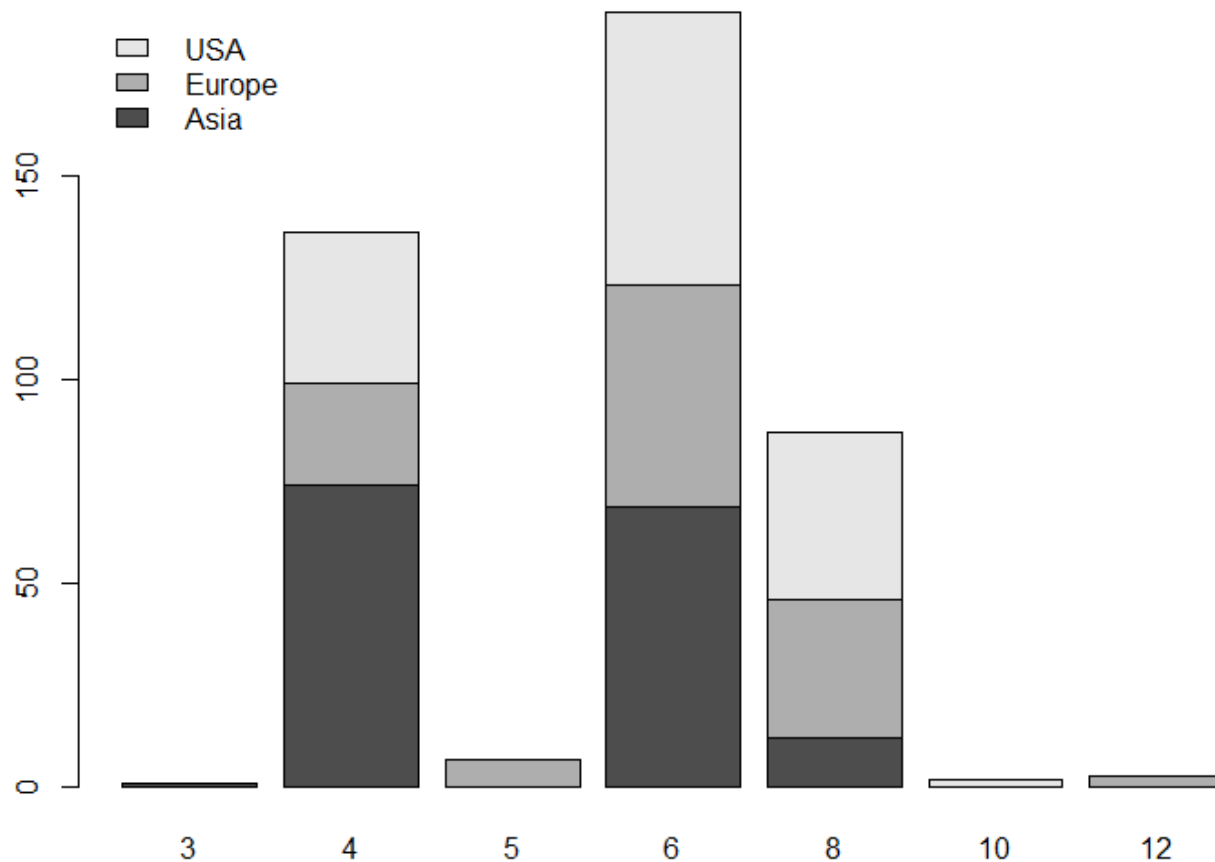


Разброс с эллипсом



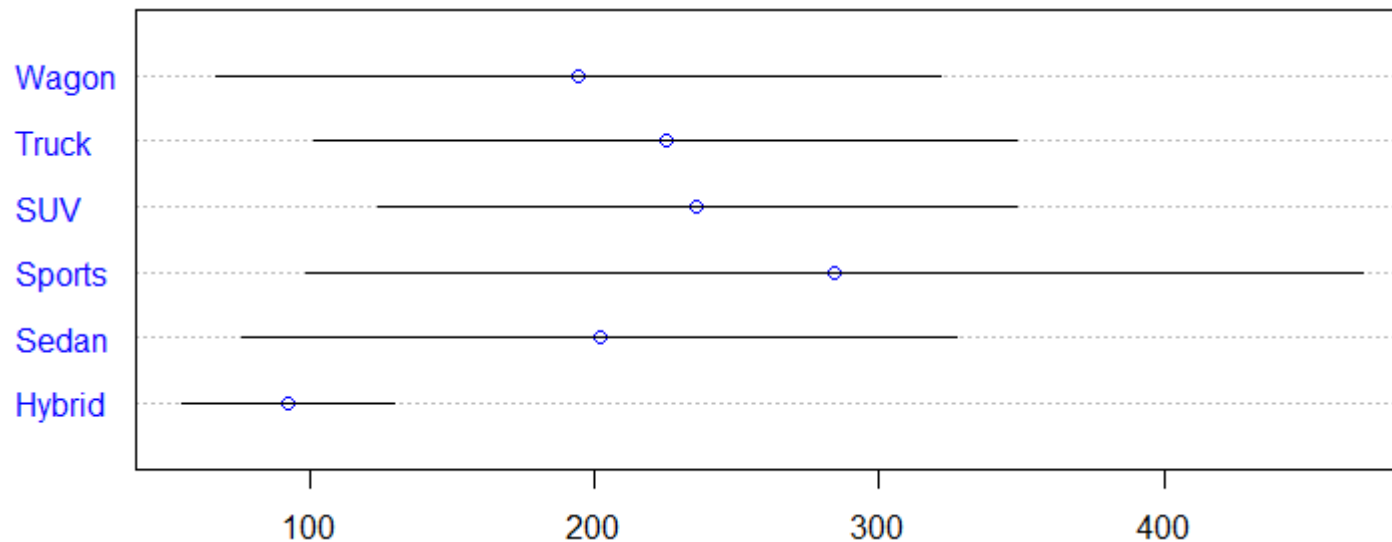
```
> cars$col <- factor(cars$Origin)
> plot(cars$Horsepower, cars$Invoice,
+       pch = 19, frame = FALSE, col = cars$col)
>
> dataEllipse(cars$Horsepower, cars$Invoice, levels=0.95,
+             lty=2, add=TRUE, groups = factor(cars$Origin))
```

Столбчатая диаграмма с группировкой



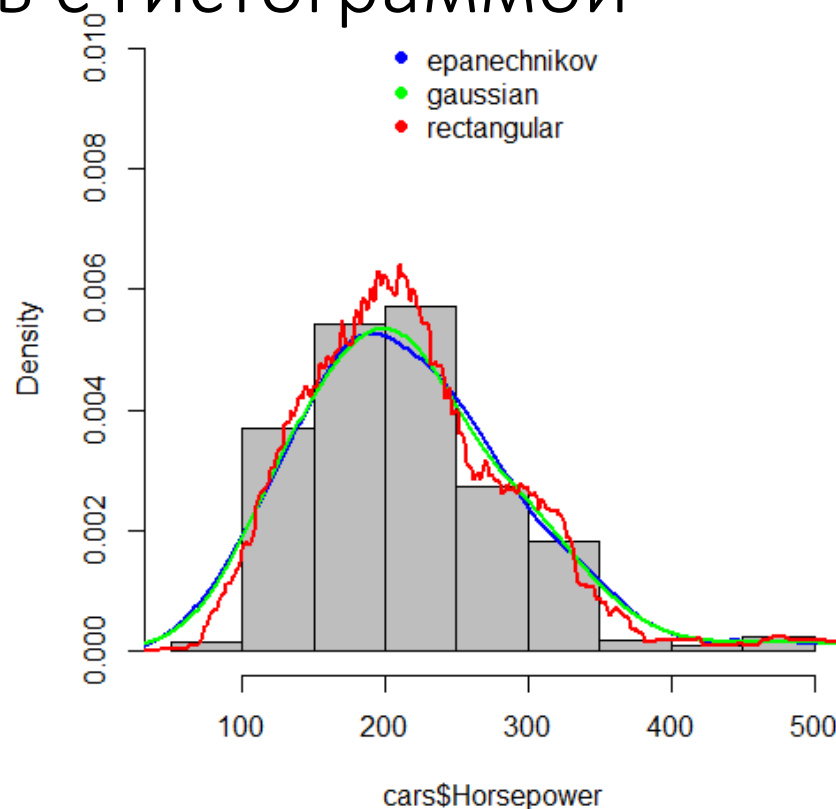
```
> count <- table(cars$Origin, cars$Cylinders)
>
> barplot(count, legend.text = rownames(count),
+         args.legend = list(x = "topleft", box.lty=0))
```

Точечная диаграмма с интервалами



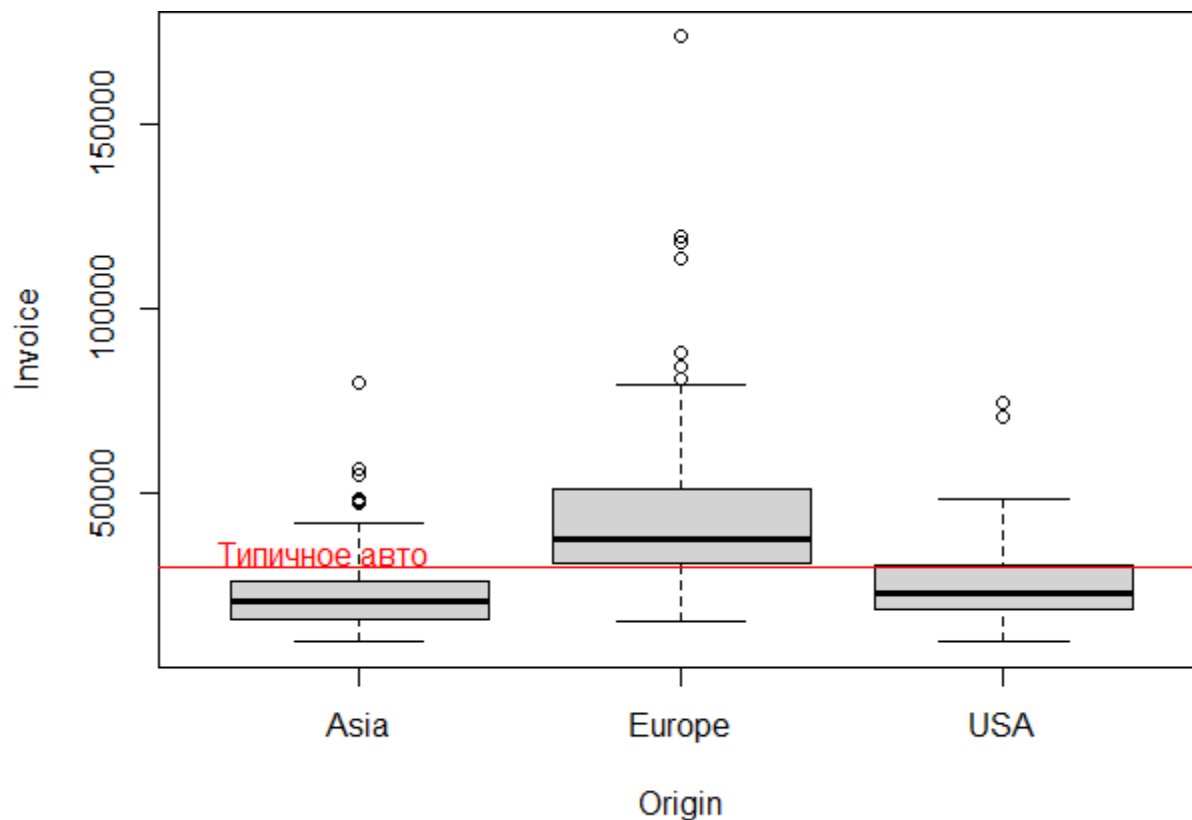
```
> cars_ <- cars[order(cars$Type),]
> x <- data.frame(
+   mean = tapply(cars_$Horsepower, list(cars_$Type), mean),
+   sd = tapply(cars_$Horsepower, list(cars_$Type), sd))
>
> x$LL <- x$mean-2*x$sd
> x$UL <- x$mean+2*x$sd
>
> dotchart(x$mean, col="blue", xlim=c(min(x$LL), max(x$UL)),
+   labels = unique(cars_$Type))
>
> for (i in 1:nrow(x)){
+   lines(x=c(x$LL[i],x$UL[i]), y=c(i,i))
+ }
```

Плотность с гистограммой



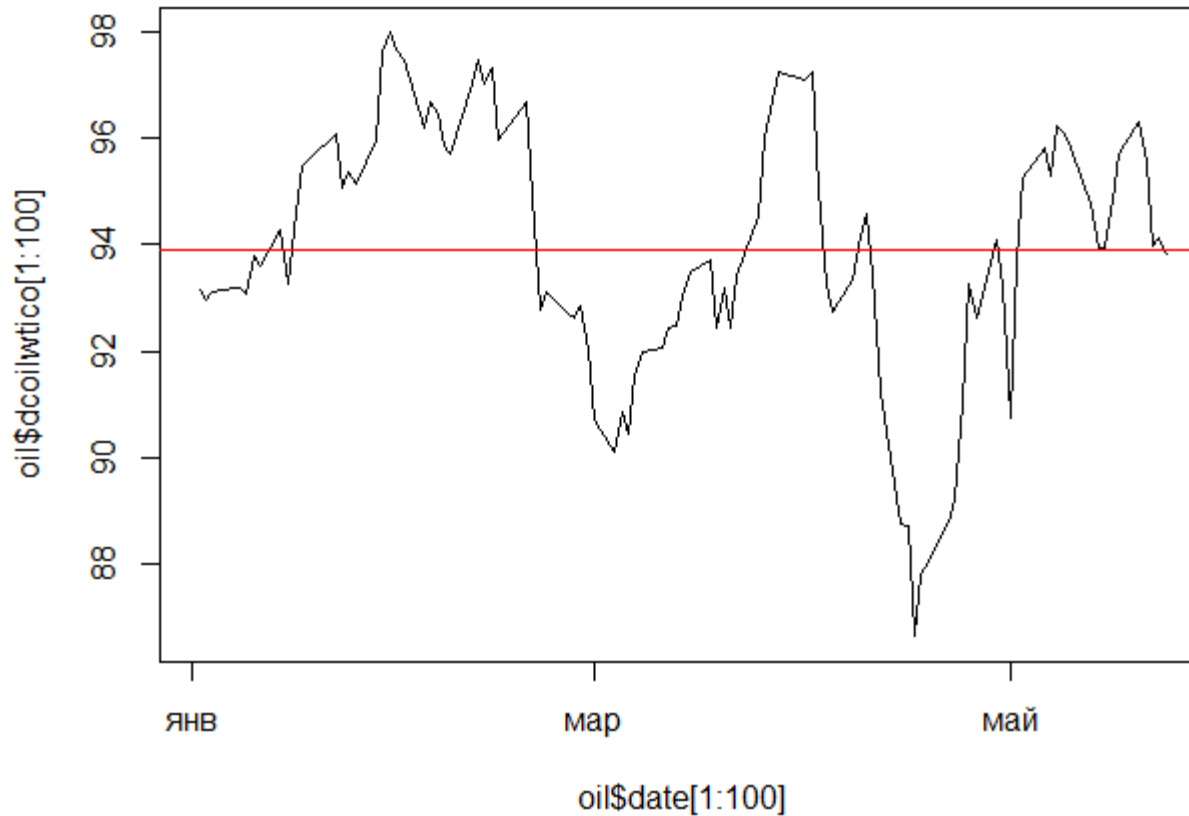
```
> hist(cars$Horsepower, prob=TRUE, ylim = c(0, 0.01), col="grey")
> lines(density(cars$Horsepower, adjust=2, kernel = "epanechnikov"), col="blue", lwd=2)
> lines(density(cars$Horsepower, adjust=2, kernel = "gaussian"), col="green", lwd=2)
> lines(density(cars$Horsepower, kernel = "rectangular"), col="red", lwd=2)
> legend("top", box.lty=0,
+ legend = c("epanechnikov", "gaussian", "rectangular"),
+ pch = 19,
+ col = c("blue", "green", "red"))
```

«Ящики с усами» с линией уровня



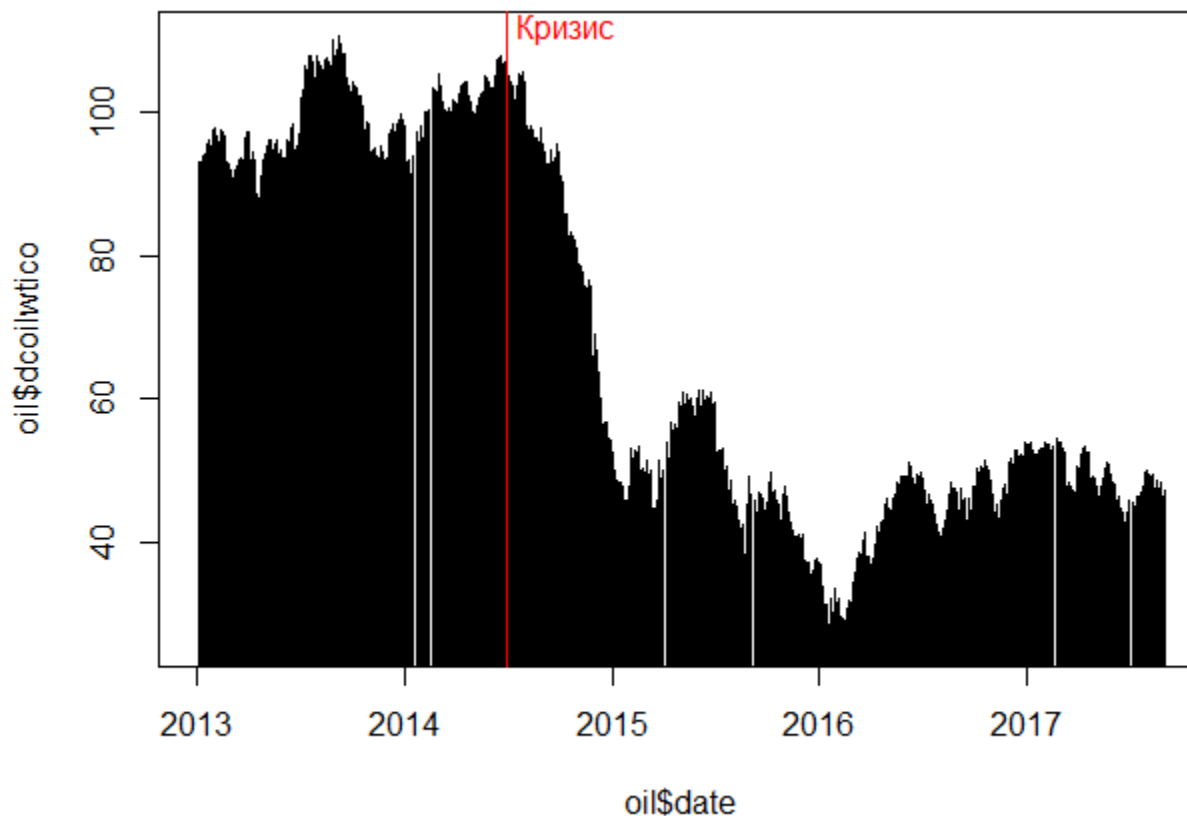
```
> boxplot(Invoice ~ Origin, data = cars, main = "boxplot", horizontal = FALSE)
> abline(h = 30000, col = "red")
> text(0.5, 30000, "Типичное авто", col = "red", adj = c(-.1, -.1))
```

Временные ряды с допуском



```
> plot(oil$date[1:100], oil$dcoiltwtico[1:100], type = "l")  
> abline(h = mean(oil$dcoiltwtico[1:100]), col = "red")
```

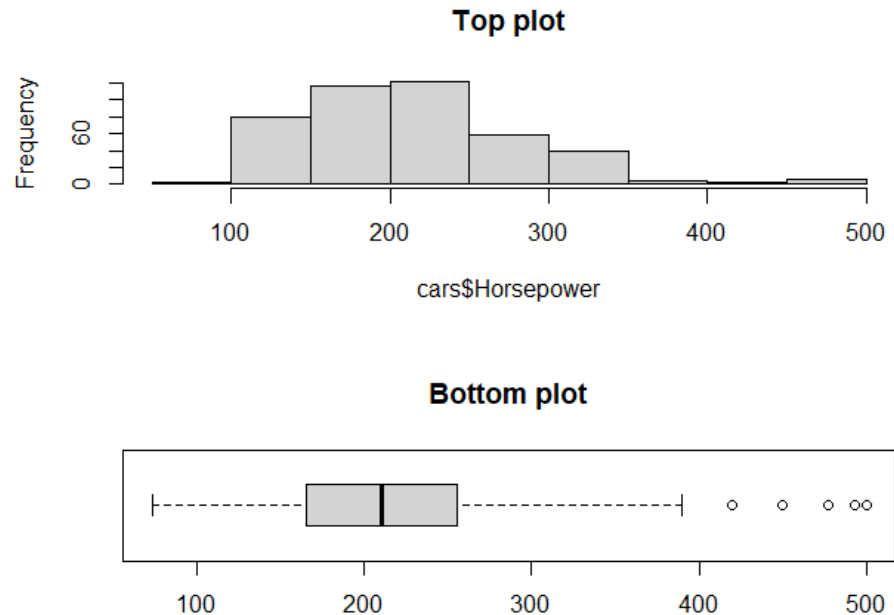
Игольчатая диаграмма с линией уровня



```
> crysis <- as.Date("2014-07-01", "%Y-%m-%d")
> plot(oil$date, oil$dcoillwtico, type = "h")
> abline(v = crysis, col = "red")
> text(crysis, 110, "Кризис", col = "red", adj = c(-.1, -.1))
```

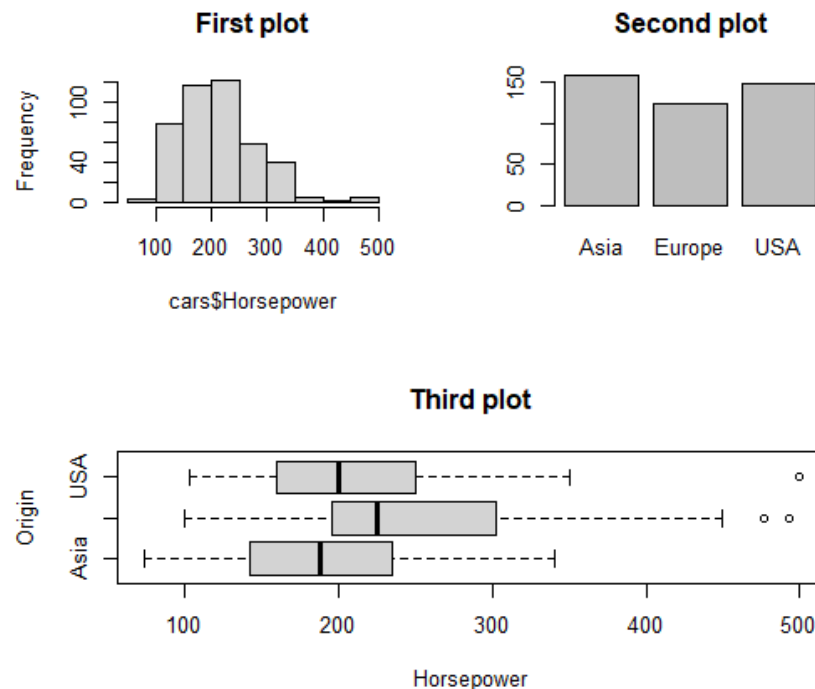
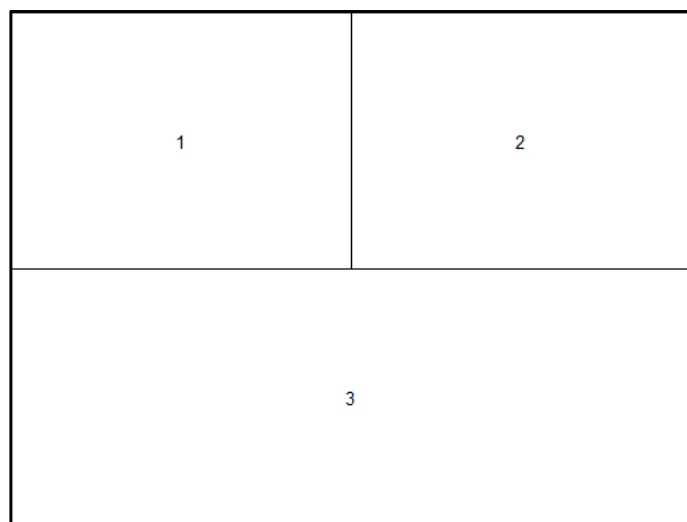
Объединение графиков в базовом графическом пакете

- `par` с параметрами `mfcol`, `mfrow`, которые в виде вектора вида `c(nr,nc)` задает формат матрицы
- `layout` сразу получает на вход матрицу, метод `layout.show` отрисовывает



```
> par(mfrow = c(2, 1))  
>  
> hist(cars$Horsepower, main = "Top plot")  
> boxplot(cars$Horsepower, main = "Bottom plot", horizontal = TRUE)
```


Объединение графиков

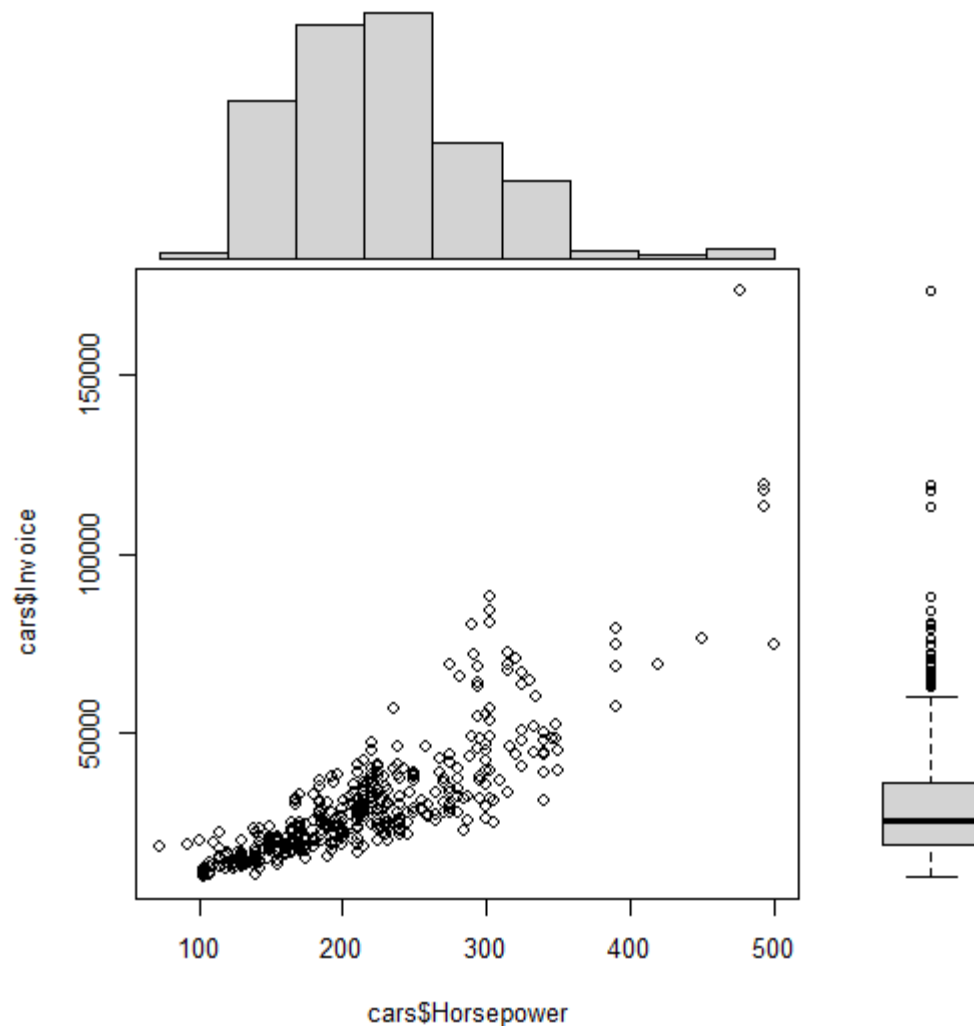


```
> l <- layout(matrix(c(1, 2, 3, 3),  
+                     nrow = 2, ncol = 2,  
+                     byrow = TRUE))  
>  
> layout.show(l)
```

```
> layout(mat = matrix(c(1, 2, 3, 3),  
+                     nrow = 2, ncol = 2,  
+                     byrow = TRUE)  
+ )  
>  
> hist(cars$Horsepower, main = "First plot")  
> barplot(table(cars$Origin), main = "Second plot")  
>  
> boxplot(Horsepower ~ Origin, data = cars,  
+         main = "Third plot", horizontal = TRUE)
```

Объединение графиков

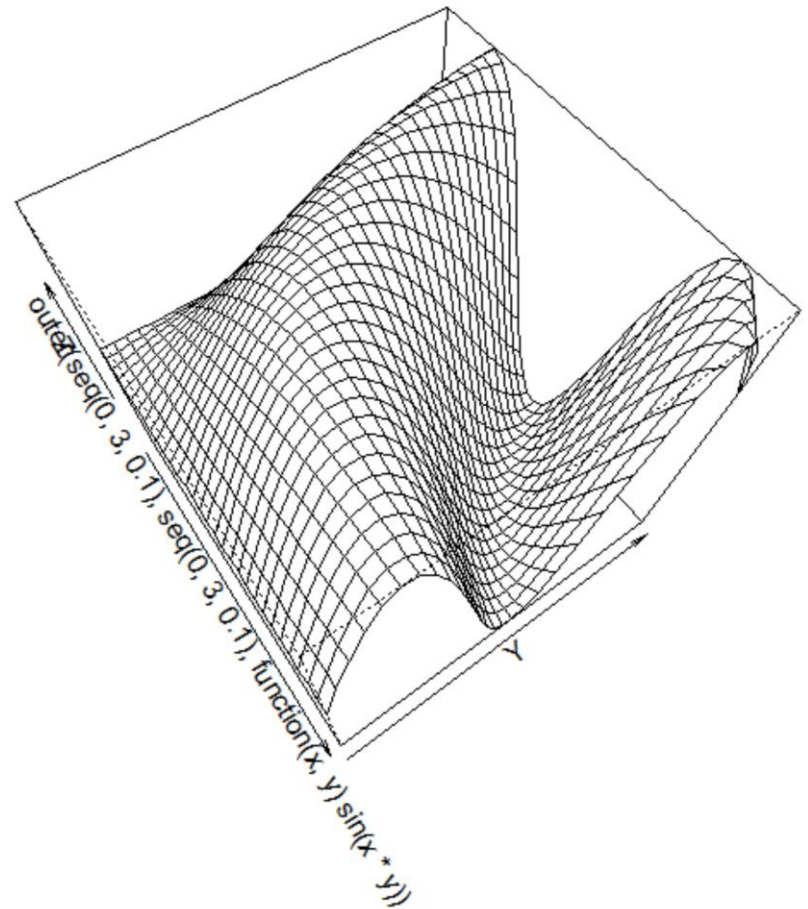
```
> layout(matrix(c(2, 0, 1, 3),
+               nrow = 2, ncol = 2,
+               byrow = TRUE),
+         widths = c(3, 1),
+         heights = c(1, 3), respect = TRUE)
>
> # Top and right margin of the main plot
> par(mar = c(5.1, 4.1, 0, 0))
> plot(cars$Horsepower, cars$Invoice)
>
> # Left margin of the histogram
> par(mar = c(0, 4.1, 0, 0))
> hist(cars$Horsepower, main = "", bty = "n",
+       axes = FALSE, ylab = "")
>
> # Bottom margin of the boxplot
> par(mar = c(5.1, 0, 0, 0))
>
> # Boxplot without plot region box
> par(bty = "n")
>
> # Boxplot without axes
> boxplot(cars$Invoice, axes = FALSE)
```



3D графики

persp(

```
x = seq(0, 1, length.out = nrow(z)),  
y = seq(0, 1, length.out = ncol(z)),  
z, xlim = range(x), ylim = range(y),  
zlim = range(z, na.rm = TRUE),  
xlab = NULL, ylab = NULL, zlab = NULL, main = NULL, sub = NULL,  
theta = 0, phi = 15, r = sqrt(3), d = 1,  
scale = TRUE, expand = 1,  
col = "white", border = NULL,  
ltheta = -135, lphi = 0, shade = NA, box = TRUE, axes = TRUE, nticks = 5, ticktype = "simple", ...)
```



```
persp(outer(seq(0, 3, 0.1), seq(0, 3, 0.1), function(x, y) sin(x * y)), phi=60, theta=55)
```



Пакет ggplot2

- Свой «синтаксис» для построения сложных графиков
- Основной вызов `ggplot()` с привязкой к данным
- Графики многослойные, включающие слои, шкалы, координаты и решетки, которые задаются последовательно, например с помощью `+`
- Сохранить график `ggsave()`
- Функции `aes_` привязывают переменные из данных к элементам графика (задают роли в зависимости от типа графика)
- Позволяет делать пользовательские геометрические элементы
- Основные слои:
 - `geom_` - для построения геометрических объектов
 - `stats_` - для статистических расчетов (например эллипсы рассеивания)
 - `position_` - для выравнивания и размещения элементов графика и графиков в целом
 - `annotations` и `labels` – текстовые слои и элементы графиков
 - `coord_` - разные системы координат
 - `guides` – оси и легенды
 - `themes` – позволяют задать и изменить весь дизайн графиков