Учебник по языку R для начинающих

Борис Демешев

2016-08-30

Оглавление

О книге

Сейчас живут три кита анализа данных и научных вычислений — Julia, Python и R. Эта книга про одного из китов!

```
library("knitr")
library("tikzDevice")

activateTikz <- function() {

  # tikz plots options
  options(tikzDefaultEngine = "xetex")

  # cash font metrics for speed:
  options(tikzMetricsDictionary = "./tikz_metrics")

  add_xelatex <- c("\\defaultfontfeatures{Ligatures=TeX,Scale=MatchLowercase}",
                   "\\setmainfont{Linux Libertine O}",
                   "\\setmonofont{Linux Libertine O}",
                   "\\setsansfont{Linux Libertine O}",
                   "\\newfontfamily{\\cyrillicfonttt}{Linux Libertine O}",
                   "\\newfontfamily{\\cyrillicfont}{Linux Libertine O}",
                   "\\newfontfamily{\\cyrillicfontsf}{Linux Libertine O}")

  options(tikzXelatexPackages = c(getOption("tikzXelatexPackages"),
                                  add_xelatex))

  # does remove warnings:
  # it is important to remove fontenc package wich is loaded by default
  options(tikzUnicodeMetricPackages = c("\\usetikzlibrary{calc}",
                                        "\\usepackage{fontspec, xunicode}", add_xelatex))


  opts_chunk$set(dev = "tikz", dev.args = list(pointsize = 11))
}


colFmt <- function(x, color) {
  outputFormat <- opts_knit$get("rmarkdown.pandoc.to")
```

```
  if (outputFormat == "latex") {
    result <- paste0("\\textcolor{", color, "}{", x, "}")
  } else if (outputFormat %in% c("html", "epub")) {
    result <- paste0("<font color='", color, "'>", x, "</font>")
  } else {
    result <- x
  }
  return(result)
}

outputFormat <- opts_knit$get("rmarkdown.pandoc.to")


if (outputFormat == "latex") {
  activateTikz()
}
```

Данная версия книги скомпилирована для latex.

```
library("dplyr")
library("ggplot2")
```

Глава 1

Первые шаги

> Поехали!
>
> Алексей Гагарин

1.1. Установка софта

Казалось бы, чего проще — поставить программу?! Однако не всегда всё идёт гладко.

Самая распространённая проблема, с которой мне доводилось бороться на разных компьютерах, — это русские буквы и пробелы в названиях файлов и папок под Windows.

> Если ты используешь Windows, то никогда при серьёзной работе не используй русские буквы и пробелы в названиях файлов и папок.

Папку с котиками можно оставить под названием мои котики :)

Заповедь о русских буквых легко нарушить даже не осознавая этого. Если имя пользователя Windows написано русскими буквами, например, Машенька, то все документы этого пользователя будут находиться в папке C:/Users/Машенька/Documents/.

> Поэтому для серьёзной работы под Windows нужно создать нового пользователя с английским именем, например, Mashenka, залогиниться и работать из-под него.

Переименование старого пользователя не помогает.

1.1.1. R

...

1.1.2. Rstudio

...

1.1.3. LaTeX

...

1.1.4. git-клиент

...

1.1.5. Текстовый редактор

Самое важное: Word — это не текстовый редактор! Текстовый редактор — это программа с помощью которой редактируют файлы с простым текстовым содержимым, а Word сохраняет файлы в специальном формате, где кроме текста сохраняется ещё куча дополнительной информации. Расширение у текстового файла может быть довольно произвольным, .txt, .md, .R, .tex и так далее.

Текстовых редакторов много. Я советую кросс-платформенный открытый и бесплатный Atom. Скачать его можн на atom.io.

1.1.6. STAN

...

1.1.7. jupyter

...

1.1.8. gretl

Тебе страшно? Тебя пугает даже список того, что нужно установить? Ты боишься R, а регрессию надо построить через 5 минут? Тогда разумное спасение — это gretl. Для gretl не обязательно учиться программировать: статистические тесты, графики и эконометрические модели доступны через меню. Кроме того, gretl даёт возможность пользователю взаимодействовать с R, что спасает в тех случаях, когда возможностей gretl не хватает.

Естественно, gretl кросс-платформенный открытый и бесплатный, gretl.

1.2. Весёлый калькулятор

R можно использовать как весёлый калькулятор:

```
5 + 9
```

```
## [1] 14
```

Что-нибудь более интересное:

```
a <- factorial(4)
b <- 2^3
a + b
```

```
## [1] 32
```

Признайся, ты всегда мечтал пошалить? Давай пока никто не видит поделим на ноль?

```
a <- 1 / 0
a
```

```
## [1] Inf
```

Что можно делать с бесконечностью, Inf?

```
1 / (Inf - 9)
```

```
## [1] 0
```

Возьмём арктангенс!

```
atan(Inf)
```

```
## [1] 1.570796
```

Ба! Да это же $\pi/2$:

```
pi / 2
```

```
## [1] 1.570796
```

Но с неопределенностью ничего не поделаешь:

```
0 / 0
```

```
## [1] NaN
```

Сокращение NaN означает «Not a Number», такой объект возникает в результате запрещённых арифметических операций.

Также в дебрях R живёт другой интересный зверь — NA, «Not Available», пропущенное наблюдение. Наблюдение может быть пропущенным по разным причинам: может быть его не было изначально, а может оно родилось в результате запрещённых арифметических операций. Поэтому всякое NaN является NA, но не всякое NA является NaN. Проверять, является ли что-либо NA или NaN, можно так:

```
is.na(0 / 0)
```

```
## [1] TRUE
```

```
is.nan(0 / 0)
```

```
## [1] TRUE
```

```
a <- NA
is.na(a)
```

```
## [1] TRUE
```

```
is.nan(a)
```

```
## [1] FALSE
```

1.2.0.1. Простые операции с векторами

Вектор из чисел по порядку:

```
a <- 3:10
a
```

```
## [1]  3  4  5  6  7  8  9 10
```

Вектор из одинаковых чисел:

```
b <- rep(777, times = 5)
b
```

```
## [1] 777 777 777 777 777
```

Вектор из конкретных чисел:

```
vect <- c(5, -4, 1)
vect
```

```
## [1]  5 -4  1
```

Что можно делать с вектором?

```
sum(vect)
```

```
## [1] 2
```

Хотите среднее арифметическое?

```
mean(vect)
```

```
## [1] 0.6666667
```

1.2.1. Отбор значений

Выберем из вектора s значения больше 0:

```
# случайная выборка из 40 равномерно распределённых на [-3;1] чисел:
s <- runif(40, min = -3, max = 1)
b <- s[s > 0] # отбираем те s, что больше нуля
b
```

```
## [1] 0.39776367 0.43293705 0.04295582 0.46592759 0.76293254 0.25051761
## [7] 0.99199148 0.92773618
```

Можно выбрать конкретные s, например с 6-го по 20-ое: {r} s[6:20]

Хочу 5-ый, 7-ой и 13-ый элементы вектора!

```
s[c(5, 7, 13)]
```

```
## [1] -2.109155 -1.080038 -2.989559
```

Можно узнать, сколько значений s больше нуля:

```
sum(s > 0)
```

```
## [1] 8
```

Еще полезная штучка — количество элементов в векторе:

```
length(b)
```

```
## [1] 8
```

Операции, похожие на те, что проделали с векторами, можно делать и с другими типами данных, например, с матрицами (*matrix*), таблицами (*table*) и др. Покажем на примере наборов данных (*data frame*).

Будем использовать набор данных diamonds из пакета ggplot2. Множество других встроены во многие пакеты, а также в сам R, и список с кратким описанием последних можно вызвать командой library(help = "datasets").

Наборы данных состоят из строк (*rows*) и столбцов (*columns*); в каждой строке хорошо построенного набора — характеристики одного объекта, а по столбцам — значения одной характеристики у разных объектов. Так, например, в diamonds объектами являются бриллианты, а их характеристиками — цена, вес, цвет и т. д. Это можно увидеть с помощью всё той же команды str():

```
str(diamonds)
```

```
## Classes 'tbl_df', 'tbl' and 'data.frame':    53940 obs. of  10 variables:
##  $ carat  : num  0.23 0.21 0.23 0.29 0.31 0.24 0.24 0.26 0.22 0.23 ...
##  $ cut    : Ord.factor w/ 5 levels "Fair"<"Good"<..: 5 4 2 4 2 3 3 3 1 3 ...
##  $ color  : Ord.factor w/ 7 levels "D"<"E"<"F"<"G"<..: 2 2 2 6 7 7 6 5 2 5 ...
##  $ clarity: Ord.factor w/ 8 levels "I1"<"SI2"<"SI1"<..: 2 3 5 4 2 6 7 3 4 5 ...
##  $ depth  : num  61.5 59.8 56.9 62.4 63.3 62.8 62.3 61.9 65.1 59.4 ...
##  $ table  : num  55 61 65 58 58 57 57 55 61 61 ...
##  $ price  : int  326 326 327 334 335 336 336 337 337 338 ...
##  $ x      : num  3.95 3.89 4.05 4.2 4.34 3.94 3.95 4.07 3.87 4 ...
##  $ y      : num  3.98 3.84 4.07 4.23 4.35 3.96 3.98 4.11 3.78 4.05 ...
##  $ z      : num  2.43 2.31 2.31 2.63 2.75 2.48 2.47 2.53 2.49 2.39 ...
```

Отобразим наблюдения с 5-го по 7-е:

```
diamonds[c(5:7), ]
```

```
## # A tibble: 3 x 10
##   carat       cut color clarity depth table price     x     y     z
##   <dbl>     <ord> <ord>   <ord> <dbl> <dbl> <int> <dbl> <dbl> <dbl>
## 1  0.31      Good     J     SI2  63.3    58   335  4.34  4.35  2.75
## 2  0.24 Very Good     J    VVS2  62.8    57   336  3.94  3.96  2.48
## 3  0.24 Very Good     I    VVS1  62.3    57   336  3.95  3.98  2.47
```

Первые 4 значения параметров цены (*price*) и веса (*carat*):

```
head(diamonds[, c("price", "carat")], n = 4)
```

```
## # A tibble: 4 x 2
##   price carat
##   <int> <dbl>
```

```
## 1   326  0.23
## 2   326  0.21
## 3   327  0.23
## 4   334  0.29
```

Можно отобразить и все названия характеристик:

```
colnames(diamonds)
```

```
##  [1] "carat"   "cut"     "color"   "clarity" "depth"   "table"   "price"
##  [8] "x"       "y"       "z"
```

Найдём, сколько камней имеют чистоту не меньшую, чем VS2 (характеристика *clarity* — упорядоченный фактор).

```
nrow(diamonds[diamonds$clarity >= "VS2", ])
```

```
## [1] 30940
```

1.2.2. Сравнение чисел — штука тонкая

Правда ли, что 0.4 + 0.1 равно 0.5?

```
0.4 + 0.1 == 0.5
```

```
## [1] TRUE
```

А правда ли, что 0.4 - 0.1 равно 0.3?

```
0.4 - 0.1 == 0.3
```

```
## [1] FALSE
```

Хм, что-то Марь Иванна в школе другое говорила...

Почему так случилось? Компьютер хранит числа в памяти в двоичной системе счисления. В двоичной системе обычное число 0.1 будет записываться в виде бесконечной периодической дроби. Следовательно, без дополнительных ухищрений храниться в памяти абсолютно точно оно не может. Поэтому де-факто компьютер хранит в памяти округленную версию от 0.4, 0.1 и 0.3. В данном случае при вычитании ошибки округления не компенсируют друг друга.

Мораль из этого примера проста:

> При операциях с дробными числами помни: компьютер считает примерно!

Скорее всего, проверка точного равенства потенциально дробных чисел не нужна. Вместо неё бывает осмысленна проверка примерного равенства:

```
all.equal(0.4 - 0.1, 0.3)
```

```
## [1] TRUE
```

Единственный нюанс. Оператор == выдаёт результат типа TRUE/FALSE, а функция all.equal может выдать развёрнутый ответ текстом. Поэтому, если нужно использовать функцию all.equal после проверки условия if, то её нужно обрамить в isTRUE:

```
if (isTRUE(all.equal(a, b))) {
  ...
}
```

1.3. Первый скрипт

....

Если текст программы содержит русские или другие неанглийские буквы, например, в комментариях, то при сохранении файла Rstudio предложит выбрать кодировку.

картинка

Кодировка определяет какой конкретно числовой код будет сопоставлен в записанном файле каждой букве. Например, букве ё в кодировке UTF-8 сопоставлен десятичный[1] код 1105.

Для русского языка есть несколько распространённых кодировок: UTF-8 и CP1251. Linux и Macos используют по умолчанию кодировку UTF-8, а вот Windows[2] сохраняет простые текстовые файлы в кодировке CP1251.

Если русскоязычный файл записать в одной кодировке, а пытаться открыть с помощью другой, то мы увидим на экране “кракозябры”. Поэтому хорошо, когда все используют одну кодировку. Кодировка UTF-8 более универсальна, чем CP1251. Например, с помощью кодировки UTF-8 в одном тексте можно использовать и русские буквы и французские акценты и китайские иероглифы.

> Мы всегда будем использовать кодировку UTF-8.

[1] Если шестнадцатиричный, то 0451. Ради интереса можно посмотреть сопоставление букв и их кодов в UTF-8, например, на unicode-table.com

[2] На самом деле всё немного хитрее и сама Windows технически использует UTF-16, а вот многие приложения под ней — CP1251.

1.4. Установка и подключение пакетов

Одна из сильных сторон R — это открытость: каждая домохозяйка может написать свой пакет для R и выложить его в публичное пользование. Пакеты расширяют возможности R. Для R написано более 10 тысяч пакетов. Среди них есть и откровенный мусор, и бриллианты, например, ggplot2, настолько ценные, что их копируют в другие языки программирования.

Скорее всего нужный тебе пакет можно найти:

1. В официальном хранилище пакетов R, CRAN.

Здесь пакеты прошли минимальное тестирование. Это отнюдь не гарантия качества пакета, но всё же серьезный давно функционирующий пакет наверняка будет выложен на CRAN.

2. В системе репозиториев github.com.

Здесь, как правило, разработчики публикуют более свежие версии пакетов, ещё не выложенные на CRAN, или молодые пакеты в процессе разработки.

3. В хранилище пакетов для биологов bioconductor.

Это своя отдельная экосистема пакетов R со специальным инсталлятором, блэкджеком и поэтэссами.

Есть и другие хранилища пакетов, например, R-forge и ..., но они гораздо менее популярны.

Сначала надо определиться, какой пакет тебе нужен. Можно погуглить, можно воспользоваться официальным классификатором пакетов R ....

Чтобы начать использовать какой-нибудь пакет R нужно сделать две вещи. Во-первых, установить его. Установка означает, что пакет будет скачан из Интернета и сохранён в специальной папке на жёстком диске. Установка пакета выполняется один раз. Каждый раз при использовании пакета устанавливать его не нужно. Переустанавливать пакет имеет смысл только если вышла новая его версия. Во-вторых, нужно подключить пакет. Подключение пакета выполняется каждый раз перед его использованием.

С репозитория CRAN пакет ставится командой R:

```
install.packages("имя пакета")
```

В Rstudio установить пакет с репозитория CRAN можно через меню: Tools - Install packages. Далее нужно набрать название пакета, можно указать сразу несколько названий через пробел, и нажать Install.

> Главное при установке пакета — не бояться сообщений красным цветом!

Любые **сообщения** (messages) R выводит красным цветом и по неопытности их можно принять за ошибку, что скорее всего не так. Ошибка всегда сопровождается словом Error. Если слова Error нет, то всё идёт по плану!

Почему R использует красный цвет? Потому, что установка пакета — это потенциально опасное действие, как и установка любой программы. Пакеты на официальном репозитории CRAN проходят определённую проверку, но если ты используешь R для многомиллионных сделок каждый день, то неплохо бы точно знать, что ты ставишь :)

Установить пакет с github.com немногим сложнее. Здесь надо знать не только название пакета, но и название репозитория, где он хранится. Часто название репозитория — это фамилия автора пакета. Официальной классификации всех пакетов R на github нет, поэтому чтобы понять, какой нужен, остаётся только гуглить.

Кроме того, для установки пакетов с github.com потребуется установить с официального репозитория

Джентельменский набор пакетов R зависит от сферы деятельности, но практически всем сталкивающимся с анализом данных пригодятся:

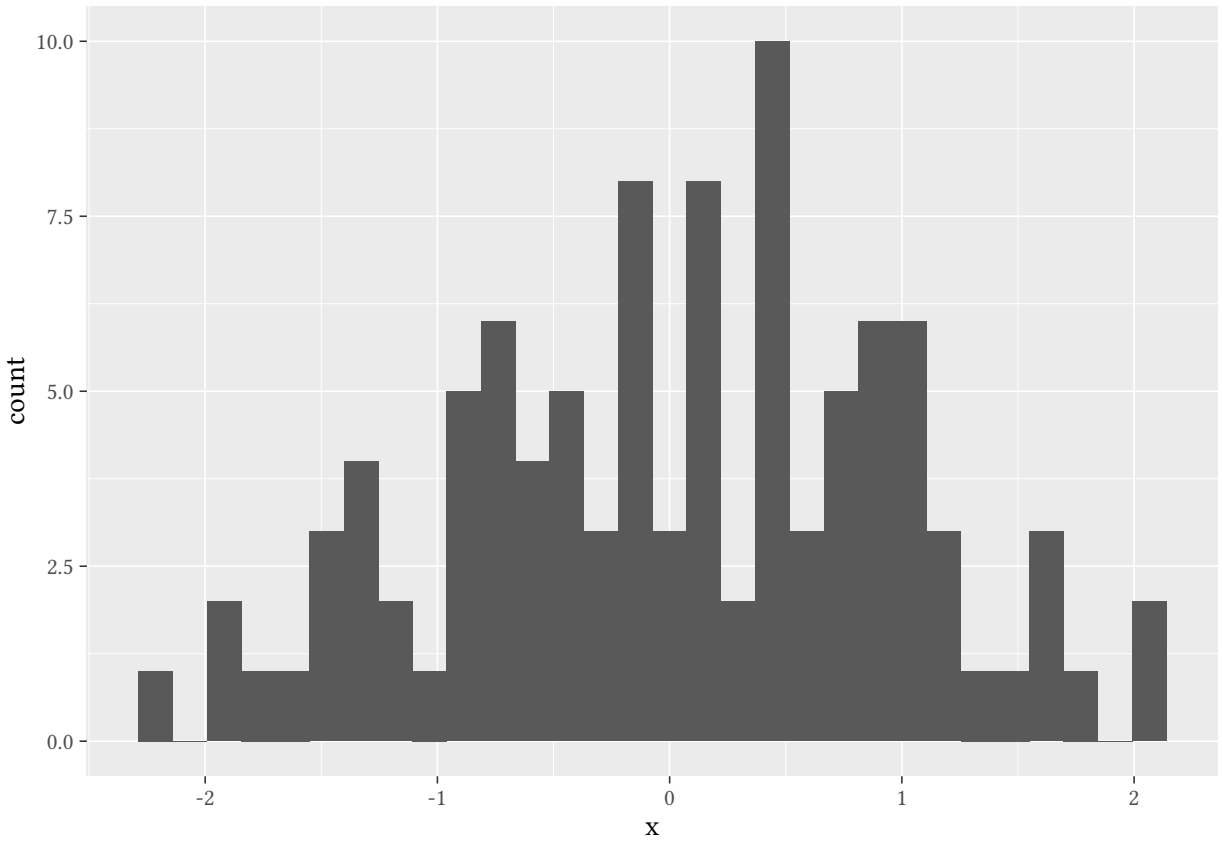
...

Очень часто пакеты R ошибочно называют библиотеками. Библиотека — это папка на жёстком диске компьютера, где хранятся пакеты.

Если пакет установлен, то можно воспользоваться его командами. Если из пакета нужна всего одна команда и один раз, то быстрее указать и нужный пакет, и нужную команду. Например, вызовем команду qplot из пакета ggplot2 и построим гистограмму для случайной выборки из 100 нормальных стандартных случайных величин:

```
x <- rnorm(100) # генерируем случайную выборку из 100 нормальных N(0;1) случайных величин
ggplot2::qplot(x)
```

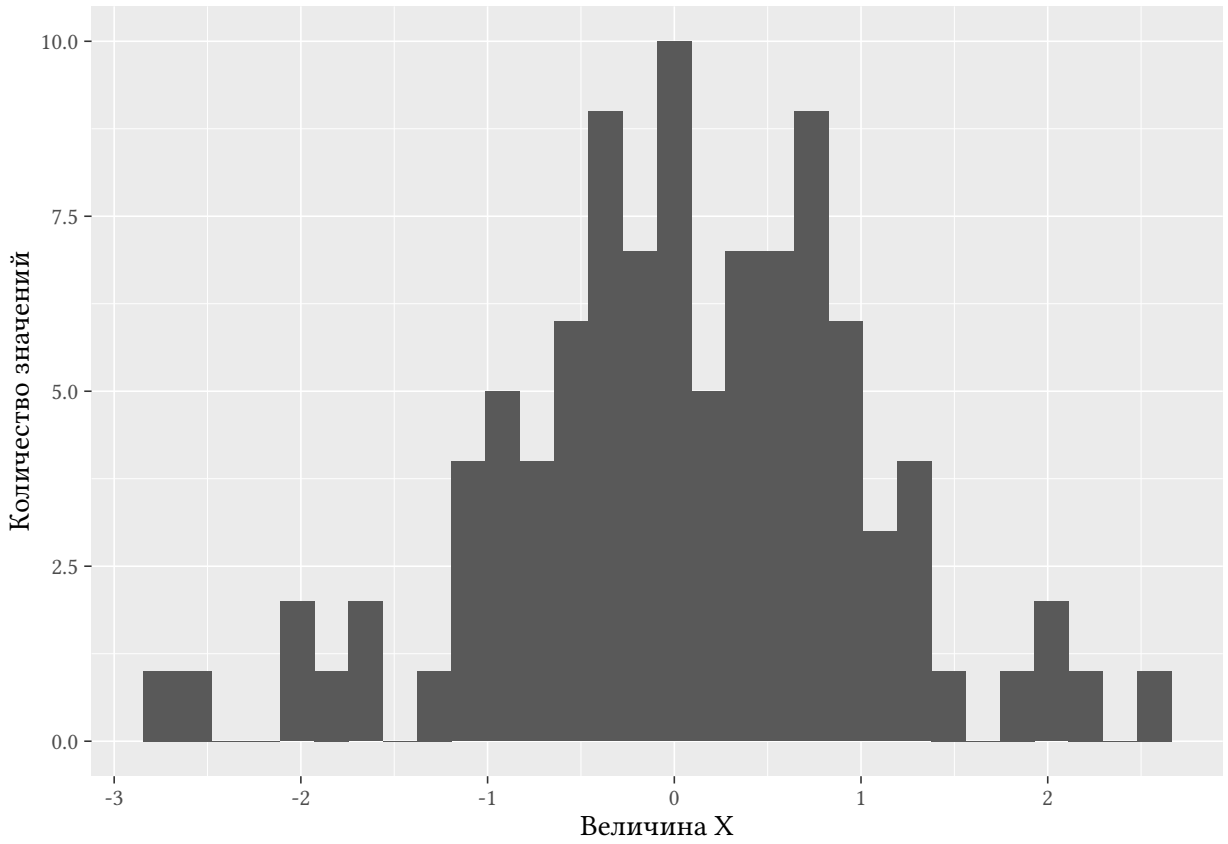
```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



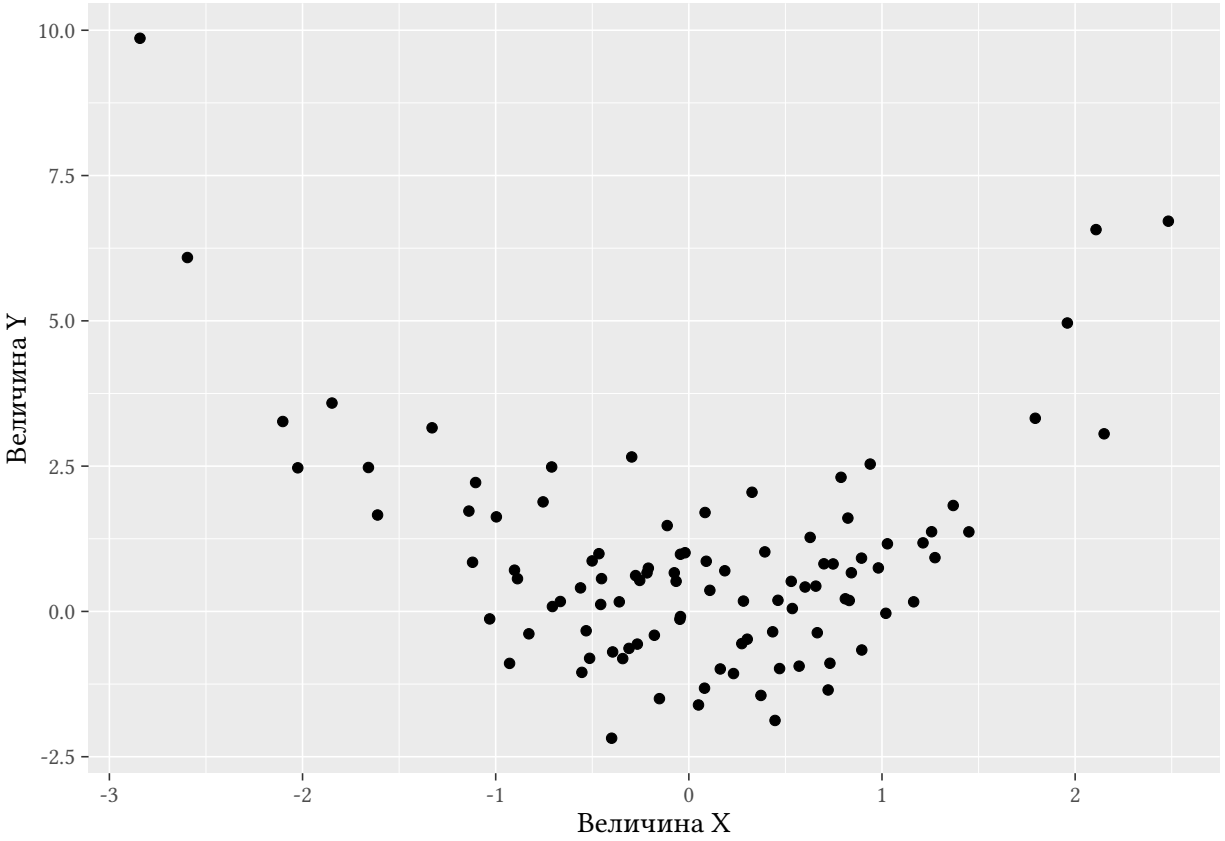
Если же ты хочешь использовать команды некоторого пакета много раз, то проще **подключить** пакет командой library(). При этом не надо будет каждый раз набирать название пакета и двойное двоеточие. Можно просто использовать команды из этого пакеты:

```
library(”ggplot2”) # подключаем пакет ‘ggplot2‘

x <- rnorm(100) # генерируем случайную выборку из 100 нормальных N(0;1) случайных величин
y <- x^2 + rnorm(100)

# строим гистограмму величины x:
qplot(x) + xlab(”Величина X”) + ylab(”Количество значений”)
```

```
## ‘stat_bin()‘ using ‘bins = 30‘. Pick better value with ‘binwidth‘.
```




```
# строим диагрраму рассеяния величин x и y:
qplot(x, y) + xlab("Величина X") + ylab("Величина Y")
```



При подключении пакета, как и при его установке, не стоит пугаться сообщений красным шрифтом. Только явное слово Error говорит об ошибке. Кроме того, часто можно столкнуться с **предупреждением** (warning) о том, что пакет был создан для более новой версии R.

Warning message: package 'xxx' was built under R version 3.3.1

Это не страшно. Это означает лишь, что у разработчика пакета xxx установлена более свежая версия R, чем у тебя. Обновлять R на своём компьютере при каждом его мелком обновлении, пожалуй, неразумно, но раз в полгода стоит.

> Правила хорошего тона советуют подключать все нужные пакеты в начале скрипта.

Чтобы узнать в каком пакете живет заданная функция, например lm, можно прямо спросить у R:

```
help("lm")
```

На самом верху справа документации будет написано lm {stats}. Это означает, что функция lm живёт в пакете stats. Пакет stats входит в ядро R, поэтому подключать его явно командой library("stats") не требуется.

1.5. Чтение и запись данных

Прежде всего неплохо бы знать, где лежит на жёстком диске файл с нужными данными. Напомню, что названия файлов и папок не должны содержать русским букв и пробелов!

У R есть понятие **рабочей папки** (working folder). В рабочей папке R ищет все требуемые файлы. Одно из простых решений — указать в качестве рабочей папки ту папку, где лежит нужный файл и далее прочитать его.

Допустим, нужный нам файл лежит в папке C:/project_A/data/. Тогда для установки рабочей папки достаточно выполнить команду:

```
setwd("C:/project_A/data/")
```

Вместо этой команды можно воспользоваться меню Rstudio: Session - Set working directory - Choose directory. Далее выбрать нужную папку и нажать Open.

После этого можно прочитать нужный нам файл. Начнём с пакета rio позволяющего импортировать данные практически любого типа. На самом деле авторы пакета rio просто объединили усилия многих разработчиков в единую команду. И получилось хорошо :)

Хочешь загрузить данные в формате .csv? Пожалуйста!

```
data <- rio::import("имя_файла.csv")
```

Хочешь загрузить данные в формате .xlsx? Пожалуйста!

```
data <- rio::import("имя_файла.xlsx")
```

Однако не всегда всё идёт гладко, поэтому остановимся подробнее на разных форматах данных.

1.6. Интернет-источники данных

Зачастую данные не обязательно даже сохранять. В R есть пакеты, дающие доступ к некоторым источникам данных в Интернете:

1. quandl
2. quantmod
3. WDI

   СССР — родина слонов!

Пакеты, дающие доступ к данным по России:

1. sophisthse sophist.hse.ru
2. cbr Центральный Банк России
3. datamos datamos.ru
4. finam.ru.

А эти источники ещё ждут желающих написать пакет для R:

1. gks.ru
2. open data gov ???

1.7. Стиль кода

R одинаково выполнит и команды

```
x<-6-7
y<--6+9
x -    y
```

и команды

```
x <- 6 - 7
y <- -6 + 9
x - y
```

Однако второй вариант гораздо приятнее для чтения. С тем, кто пишет код как в первом примере, Английская королева рядом не сядет! Чтобы иметь возможность войти в палату Лордов и Общин, тебе следует писать стильный код!

Если ты работаешь в команде, то руководствуйся тем стилем кода, который в ней принят. А для новичков я советую использовать стиль кода, которого придерживается Hadley Wickham, автор очень популярных пакетов R ggplot2 и dplyr:

1. После запятой всегда пиши пробел. Перед запятой — никогда:

```
paste0("Hi ", "guys!")
```

2. Знак присваивания <-, знаки арифметических действий (+, -, *), логические проверки (>, <, == и прочие) с двух сторон окружай одинарными пробелами.

```
x <- (3.5 + 7) * (2.8 - 9)
```

3. Открывающую фигурную скобку оставляй на старой строке, а закрывающую — ставь на новую:

```
if (x == y) {
  message("Variables x and y are equal.")
}
```

В Rstudio можно включить автоматическую проверку стиля кода в Tools - Global options - Code - Diagnostics. Настоящие сэры и истинные леди в разделе Diagnostics могут проставить все галочки.

1.8. Две записи функций

Мы все привыкли к тому, что домохозяйки пишут рецепт в естественном порядке, а математики функции — в обратном. Сравни:

> Возьмите пепел перьев чёрного петуха
>
> Добавьте печень дракона
>
> Варите на медленном огне 2 дня

и

$$\cos(\sin(|x|))$$

У домохозяек порядок изложения совпадает с порядком действий. У математиков сначала написано про косинус, но считается он в самом конце.

Похоже Лёнька Эйлер и Алёшка Клеро

фото

введя обозначение $f(x)$ отделили математиков от домохозяек и, вероятно, пустили математику по ложному пути. Было бы гораздо удобнее, если бы в математике функции также записывали в естественном порядке! Но обозначение $f(x)$ мы впитали с молоком матери, уже вряд ли что исправишь.

R позволяет использовать обе традиции обозначени.

Традиция Эйлера-Клеро:

```
cos(sin(abs(10)))
```

```
## [1] 0.8556344
```

Для того, чтобы иметь возможность писать операции в естественном порядке, подключаем пакет dplyr:

```
library("dplyr")
```

И теперь в традиции лучших кулинарных рецептов можно написать

```
10 %>% abs() %>% sin() %>% cos()
```

```
## [1] 0.8556344
```

Оператор %>% называется трубочкой (pipe). (? канал) По первому впечатлению кажется, что эти трубочки долго писать. Но стоит к ним привыкнуть и ощущаешь, что они безумно удобны для сложных операций!

1.9. Манипуляции с данными

(здесь про типы данных)

1.10. Графики

График можно строить либо чтобы: - самому по-быстрому взглянуть на некий результат и сразу забыть график - показать график кому-нибудь

В первом случае нет никаких требований к графику — лишь бы самому было понятно, что там изображено. Если же график показывать кому-то ещё, то:

> Идеальный график должен быстро и верно восприниматься смотрящим.

Из этого простого принципа следует ряд выводов:

1. Идеальный график должен быть самодостаточным.

Если для понимания графика смотрящему требуется прочитать кучу текста вокруг или прослушать получасовое объяснение, то это нехорошо :) Вырежи свой график из статьи/книги/курсовой и подумай, понятен ли он?

2. Подписывай оси.

[ссылка на xkcd]

3. Выбирай единицы измерения так, чтобы читатель не мучился, считая нули у каждой цифры.
4. Указывай единицы измерения.
5. Хорошо бы указать источник данных.
6. Лучше расшифровать сокращения, хотя иногда это бывает нелегко.
7. Никаких круговых диаграмм.

Круговые диаграммы — это табу. Да, R умеет их строить. Процитирую документацию к функции pie:

```
help(pie)
```

> Pie charts are a very bad way of displaying information. The eye is good at judging linear measures and bad at judging relative areas. A bar chart or dot chart is a preferable way of displaying this type of data.
>
> Cleveland (1985), page 264: “Data that can be shown by pie charts always can be shown by a dot chart. This means that judgements of position along a common scale can be made instead of the less accurate angle judgements.” This statement is based on the empirical investigations of Cleveland and McGill as well as investigations by perceptual psychologists.

8. Никаких псевдо-3D эффектов и прочих «рюшечек».

Посмотрите пару анимаций от DarkHorse.

9. Ось должна начинаться от нуля.
10. Полезно немного ознакомиться с твореними мэтров :)

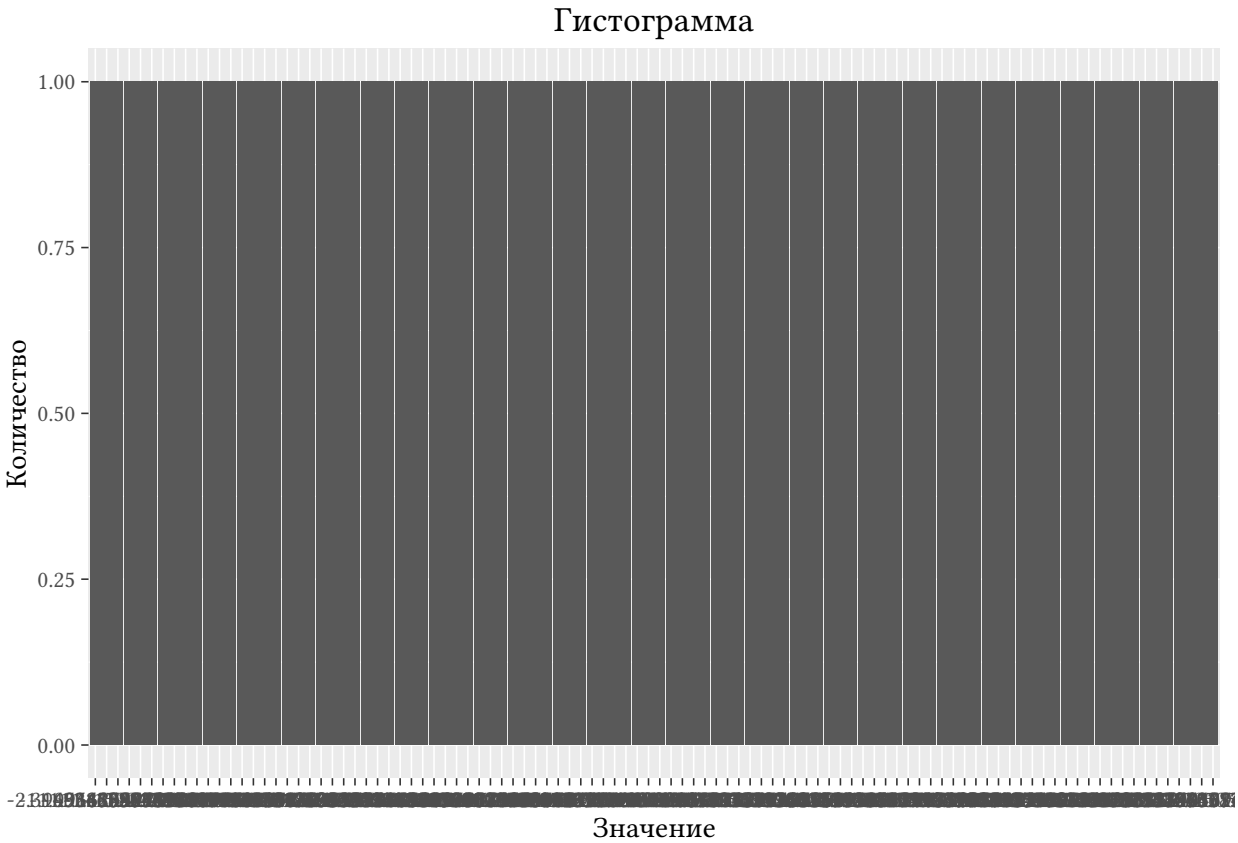
Tufte, link

Очень часто к графикам относятся как к чему-то вспомогательному. На самом деле график очень часто является основным результатом, гораздо более важным, чем несколько страниц текста. Именно поэтому полезно отказаться от мысли «я тут за пять минут график вставлю» и потратить на график часок-другой!

1.10.1. Два простеньких графика

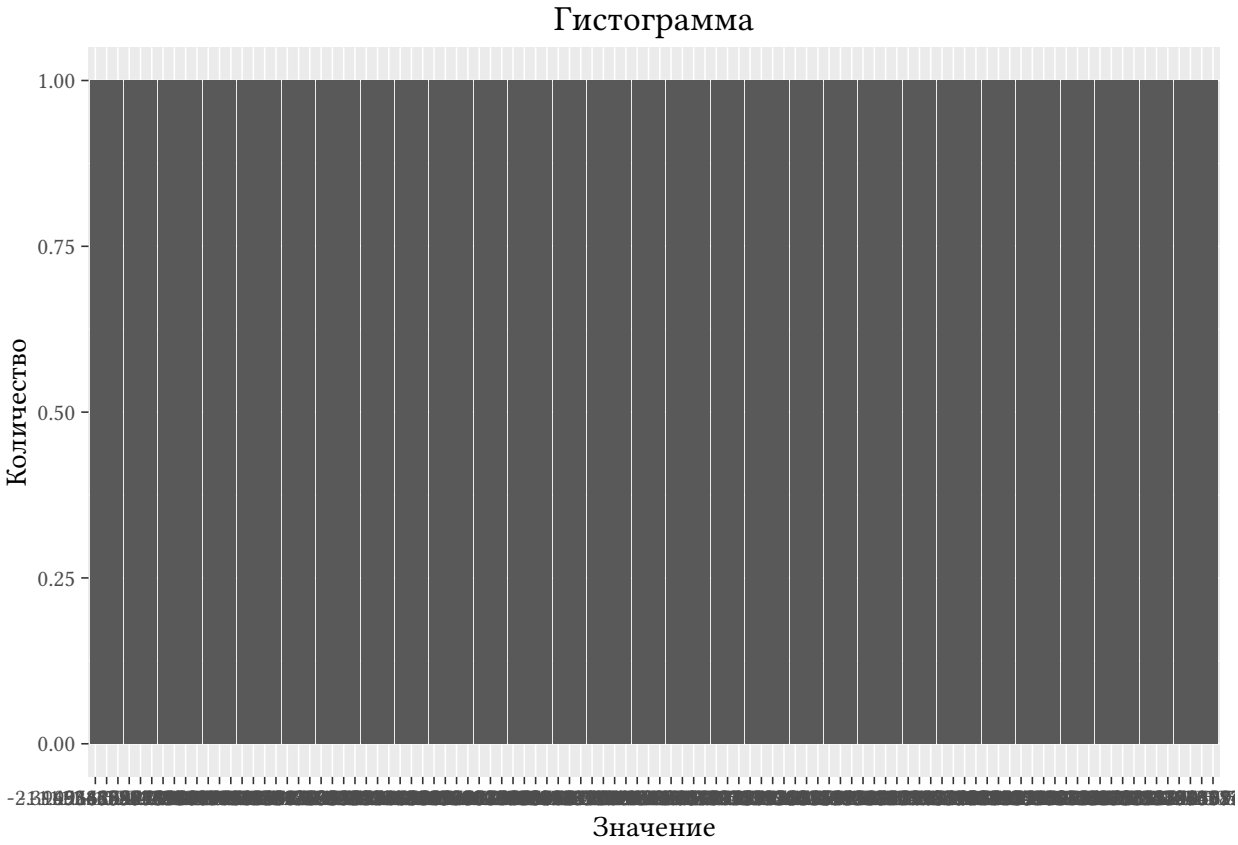
Простенькую гистограмму можно построить, например, с помощью функции qplot() из пакета ggplot2:

```
s <- rnorm(100)
qplot(factor(s), xlab = "Значение",
ylab = "Количество", main = "Гистограмма")
```



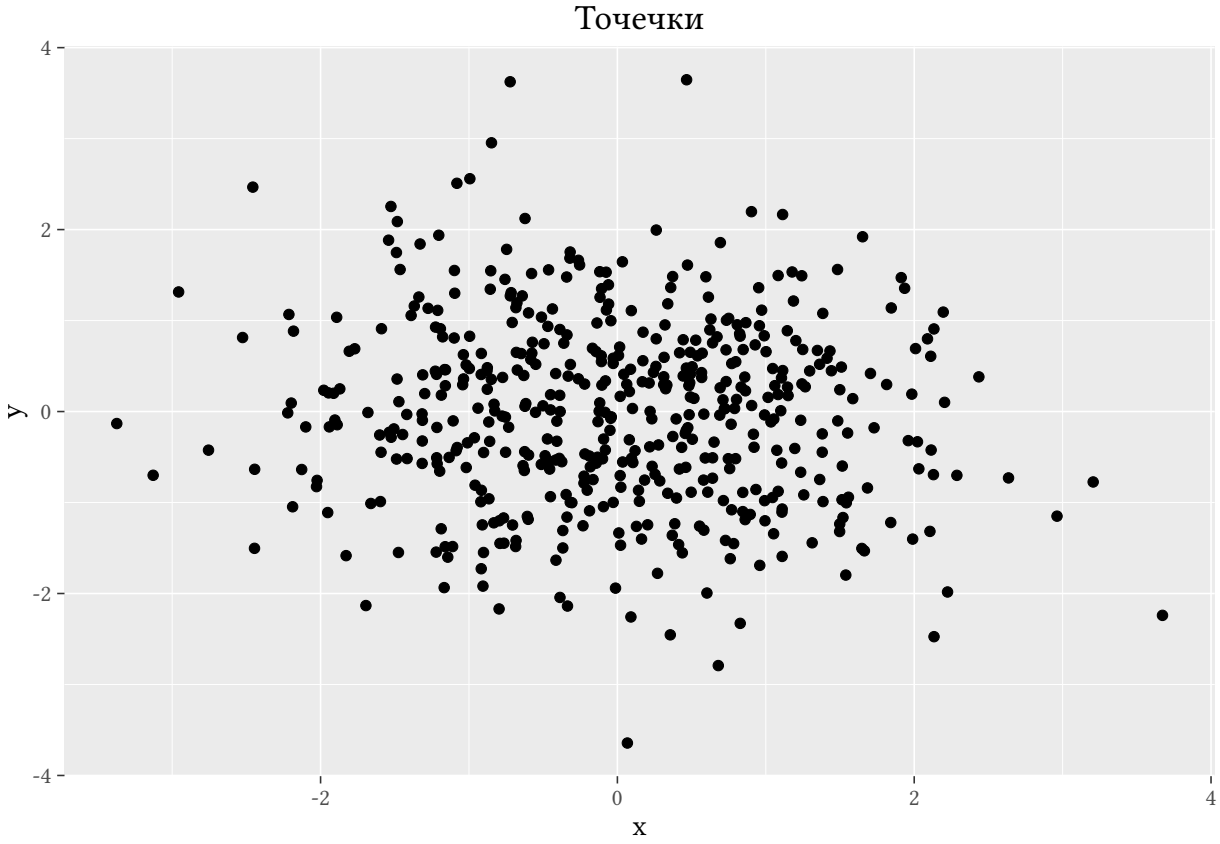
Ту же гистограммку можно построить с помощью функции ggplot() из того же пакета, предварительно преобразовав s в необходимый для ggplot() формат data frame:

```
s_df <- data.frame(s)
ggplot(s_df) + geom_bar(aes(factor(s))) +
labs(x = "Значение", y = "Количество", title = "Гистограмма")
```



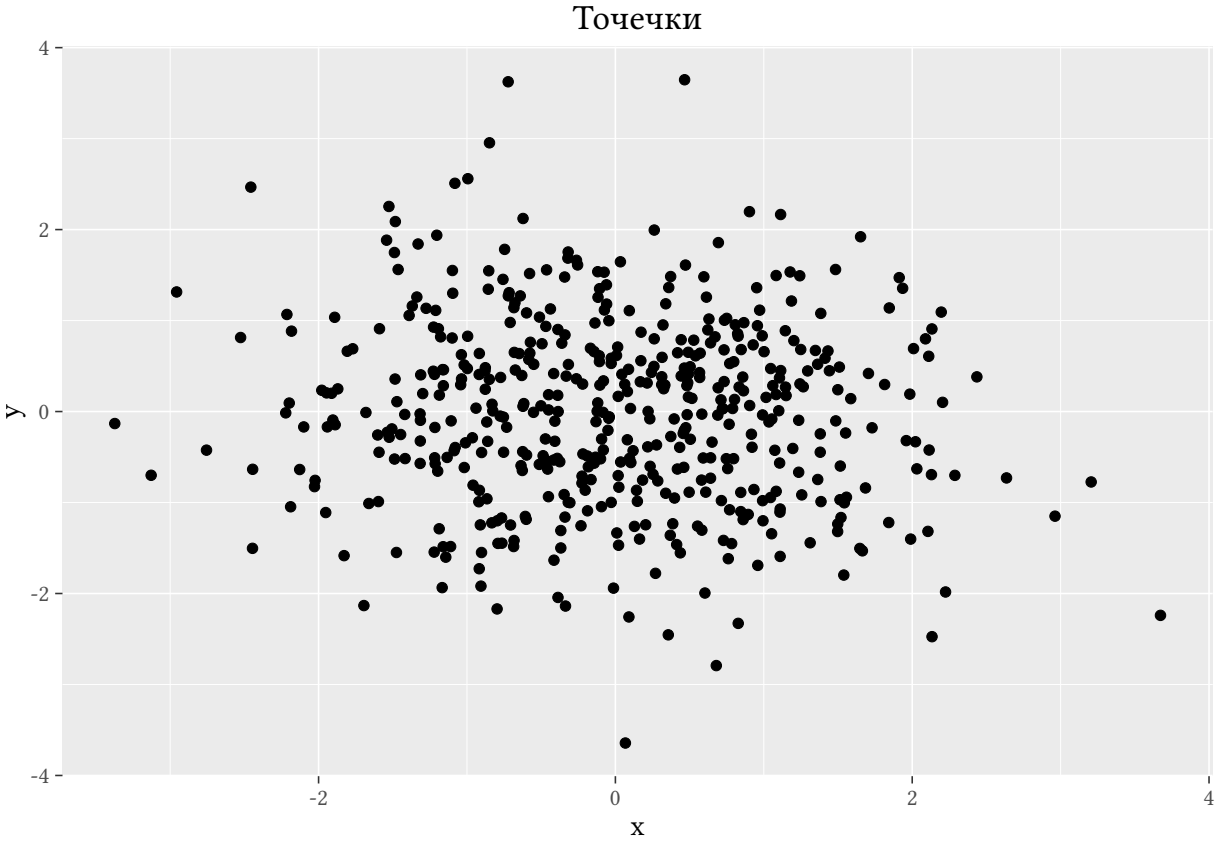
И еще простенький график:

```
x <- rnorm(500)
# 500 нормальных величин со средним 0 и дисперсией 1
y <- rnorm(500) # 500 нормальных величин со средним 0 и дисперсией 1
qplot(x, y, main = "Точечки")
```



Или:

```
xy_df <- data.frame(x = x, y = y)
ggplot(xy_df) + geom_point(aes(x, y)) + labs(title = "Точечки")
```



В зависимости от формата данных часто бывает удобно использовать и qplot(), и ggplot().

1.11. У меня ошибка!

> Шеф! Всё пропало! Гипс снимают, клиент уезжает!

поговори с уточкой, посиди у озера

1.12. Ресурсы по R

Начнём с русскоязычных крупных книжек:

- Шипунов А.Б., Балдин Е.М. и др. Наглядная статистика. Используем R! * Мастицкий С.Э., Шитиков В.К., Статистический анализ и визуализация с помощью R
- Василенко Є.С, Обробка геологічної інформації з використанням програмного середовища R * Кабаков Р.И. R в действии. Анализ и визуализация данных на языке R

На русском также могут быть полезны:

- Стилевое руководство гугла: english, русский

- Блог r-analytics. На нём есть несколько других подборок русскоязычных ресурсов по R: раз и два. * Группа вконтакте rstatistics * Группа вконтакте spbrug * Группа вконтакте introstats * Венэбльз, Смит Введение в R
- Зарядов И.С. Введение в R: часть 1
- Зарядов И.С. Введение в R: часть 2
- Борис Демешев Цикл маленьких заметок по R

Вопросы и ответы:

- stackoverflow Если возникли проблемы с программированием в R (и не только в R), а документация уже прочитана... *stats.stackexchange Можно спросить по статистическим методам и их реализации в R.* tex.stackexchange Вопросы и ответы про LaTeX

Он-лайн курсы и видеолекции с использованием R:

- Try R Путь к сокровищам R для самых начинающих пиRатов!
- datacamp.com Несколько платных и бесплатных интерактивных курсов по R
- Видео-лекции курса Computing for Data Analysis: неделя 1, неделя 2, неделя 3, неделя 4. Сам курс доступен на coursera.org

Открытые крупные источники на английском:

- Trevor Hastie, Robert Tibshirani, An Introduction to Statistical Learning. По книжке есть курс на платформе class.standford.edu
- Rob Hyndman, Forecasting: principles and practice Книжка по временным рядам от автора пакета forecast
- Garrett Grolemund, Hadley Wickham, R for Data Science
- Colin Gillespie, Robin Lovelace, Efficient R programming, он-лайн книга для более продвинутых пользователей R. Написана на R и markdown :)
- Rob Kabacoff, Quick R
- Farnsworth, Econometrics in R
- Cookbook for R
- R-inferno Адский учебник по R с картинками Ботичелли, “Even if it doesn’t help you with your problem, it might amuse you”
- Roger Peng, R programming for data science и на bookdown
- Introduction to R, Введение с официального сайта
- документация по графикам ggplot2
- документация к R от stats.stackexchange

- **Шикарные шпаргалки от Rstudio**
- 100 курсов и книг по R от разных университетов
- Агрегатор блогов r-bloggers
- Поисковик rseek.org

Великолепные виньетки к разным пакетам:

- vars, векторные авторегрессии, есть вольный пересказ по-русски *dplyr манипуляции с данными* sandwich борьба с гетероскедастичностью и заклинания HC0, HC1, HC2, ... *plm панельные модели в R* ggmap рисуем карты *vcd полезные графики для качественных переменных* Beanplot: A Boxplot Alternative for Visual Comparison of Distributions.

Ты ещё не заработал свой первый миллион и хочешь скачать научную статью или книжку бесплатно?

- Научные книги можно найти на gen.lib.rus.ec.
- Научные статьи можно скачать на sci-hub.cc. Есть даже бот @scihubot для Telegram, которые вышлет в ответ на запрос полный текст статьи.

Если какая-нибудь ссылка не работает или хочешь предложить свою, смело открывай issue на github!

Глава 2

Друзья R

ывава ыва ыва test

2.1. Рабочий процесс

...

2.2. Контроль версий

...

2.3. Латех

...

2.4. Маркдаун

...

2.5. Воспроизводимые исследования

...

2.6. Написание своего пакета

...

2.7. Вычисления в облаке

...

2.8. Презентации

...

2.9. Про эту книжку

Разбить на большее количество глав!!!

Что-то убрать? чтобы была ещё одна книжка? :)

Общие принципы:

1. Неформальный стиль, на “ты”
2. Больше картинок. Лицензия?
3. Больше гипер-ссылок.
4. Буква ё обязательна.

Пакет bookdown с помощью которого написана эта книжка ещё немного сыроват. В процессе работы я обнаружил, что:

1. Порой помогает удаление промежуточных файлов и компиляция заново.
2. После неанглоязычного названия главы обязательно должна идти метка {#label}.
3. Каждый .Rmd файл содержит только одну главу. Глава обозначается заголовком первого уровня #.
4. Сослаться на главу или подраздел можно с помощью \@ref(label).
5. Сослаться на источник литературы можно с помощью [@reference]
6. Автодобавление пакетов

глючит на ”M”uller

6. Для создания MOBI книг:

```
brew update
brew cask install calibre
brew cask install kindlegen
```

В предисловии

```
bookdown::kindlegen:
  epub: _book/r_manual.epub
```

ругается

или

```
bookdown::calibre:
  input: _book/r_manual.epub
  output: _book/r_manual.mobi
```

ругается

Если надо изобразить yaml в чанке кода, пока пишу, что он bash

7. заставляем травис работать

Создаём новый токен на github: кликнуть по иконке пользователя, далее settings - token - generate new token.

```
sudo gem install travis
travis encrypt GITHUB_TOKEN=[token from githum]
```

Именно переменная GITHUB_TOKEN должна использоваться для того, чтобы обращаться к гитхабу, т.е.

Добавляем получившуюся закодированную строку в .travis.yml. Забавно, что похоже используется рандомный ключ для шифрования и поэтому зашифрованная строка каждый раз выходит разной.

Можно не шифровать её, а добавить в travis-ci.org/user/repo/settings

Для быстрой компиляции добавляем в .travis.yml указание, что мы будем использовать готовые бинарные пакеты R:

```
r_binary_packages:
  - ggplot2
  - dplyr
  - rio
```

Черт его знает? всё равно компилирует?

8. Красный шрифт

по мотивам http://stackoverflow.com/questions/29067541

```
colFmt <- function(x, color) {
  outputFormat <- opts_knit$get("rmarkdown.pandoc.to")
  if (outputFormat == "latex") {
    result <- paste0("\\textcolor{", color, "}{", x, "}")
  } else if (outputFormat %in% c("html", "epub")) {
    result <- paste0("<font color='", color, "'>", x, "</font>")
  } else {
    result <- x
  }
  return(result)
}
```

Then you can use it inline like this: MY RED TEXT

9. Файл _output.yaml это просто output: кусок из yaml-части файла index.Rmd. Поэтому можно внести _output.yaml обратно в index.Rmd, чтобы все настройки были в одном месте.

Формально _output.yaml действует на все .Rmd документы в папке, но что там будет кроме учебника в целом. Разве что отдельные главы компилировать :)

Глава 3

Статистика и не только

ываываыв ываываыв

3.1. Генерирование случайных величин

Для решения задач по теории вероятностей или исследования свойств статистических алгоритмов может потребоваться сгененировать случайную выборку из заданного закона распределения.

Генерируем 10 равномерных на отрезке $[4; 10.5]$ случайных величин:

```
runif(10, min = 4, max = 10.5)
```

```
##  [1]  8.595328  5.186400  7.788169  9.413227  4.469055  5.694882  6.429750
##  [8]  9.191000  6.866668 10.372320
```

Генерируем 10 нормальных $N(2; 9)$ случайных величин с математическим ожиданием 2 и дисперсией $9 = 3^2$:

```
rnorm(10, mean = 2, sd = 3)
```

```
##  [1]  0.3646733 -0.6747956  2.5236002  1.8172200  6.8635468 -2.2709063
##  [7] 10.1709338  1.4740829 -1.0072876  0.2422856
```

Например, с помощью симуляций легко оценить математическое ожидание $E(1/X)$, где $X \sim N(2; 9)$. Для этого мы вспомним Закон Больших Чисел. Он говорит, что арифметическое среднее по большой выборке стремится по вероятности и почти наверное к математическому ожиданию. Поэтому мы просто сгенерируем большую выборку в миллион наблюдений:

```
n_obs <- 10^6
x <- rnorm(n_obs, mean = 2, sd = 3)
mean(1/x)
```

```
## [1] 0.2315722
```

Также легко оценить многие вероятности. Например, оценим вероятность $P(X_1 + X_2 + X_3^2 > 5)$, где величины X_i независимы и одинаково распределены $X_i \sim U[0; 2]$:

```
n_obs <- 10^6
x_1 <- runif(n_obs, min = 0, max = 2)
x_2 <- runif(n_obs, min = 0, max = 2)
x_3 <- runif(n_obs, min = 0, max = 2)
success <- x_1 + x_2 + x_3^2 > 5
sum(success) / n_obs
```

```
## [1] 0.147746
```

Здесь вектор success будет содержать значение TRUE там, где условие x_1 + x_2 + x_3^2 > 5 выполнено, и FALSE там, где условие не выполнено. При сложении командой sum() каждое TRUE будет посчитано как единица, а каждое FALSE как ноль. Поэтому sum(success) даст количество раз, когда условие x_1 + x_2 + x_3^2 > 5 выполнено.

С любым распределением [xxx] в R связано четыре функции: r[xxx], d[xxx], p[xxx] и q[xxx]. Для примера возьмём нормальное распределение $N(2; 9)$:

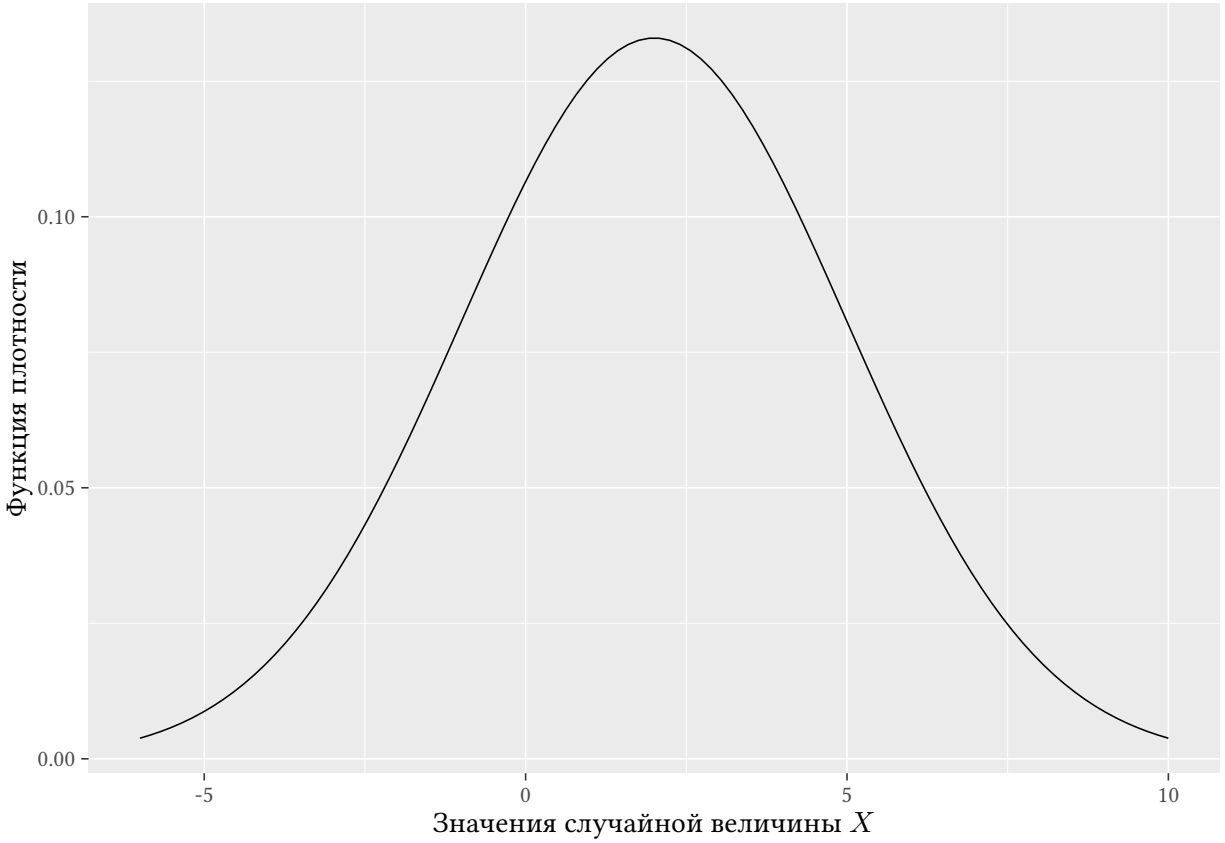
- Функция для создания случайной выборки из нормального $N(2; 9)$ распределения — rnorm:

```
x <- rnorm(100, mean = 2, sd = 3) # случайная выборка из 100 нормальных N(2; 9) величин
head(x, 10) # первые 10 элементов вектора
```

```
##  [1]  6.66716999  0.57177477 -0.27757204  1.27724209 -0.04361137
##  [6]  3.03237541  3.96432732 -0.83357816 -0.09781417  1.97074403
```

- Функция плотности — dnorm:

```
x <- seq(from = -6, to = 10, length.out = 100)
y <- dnorm(x, mean = 2, sd = 3)
qplot(x = x, y = y, geom = "line") + xlab("Значения случайной величины $X$") + ylab("Функция плотности")
```



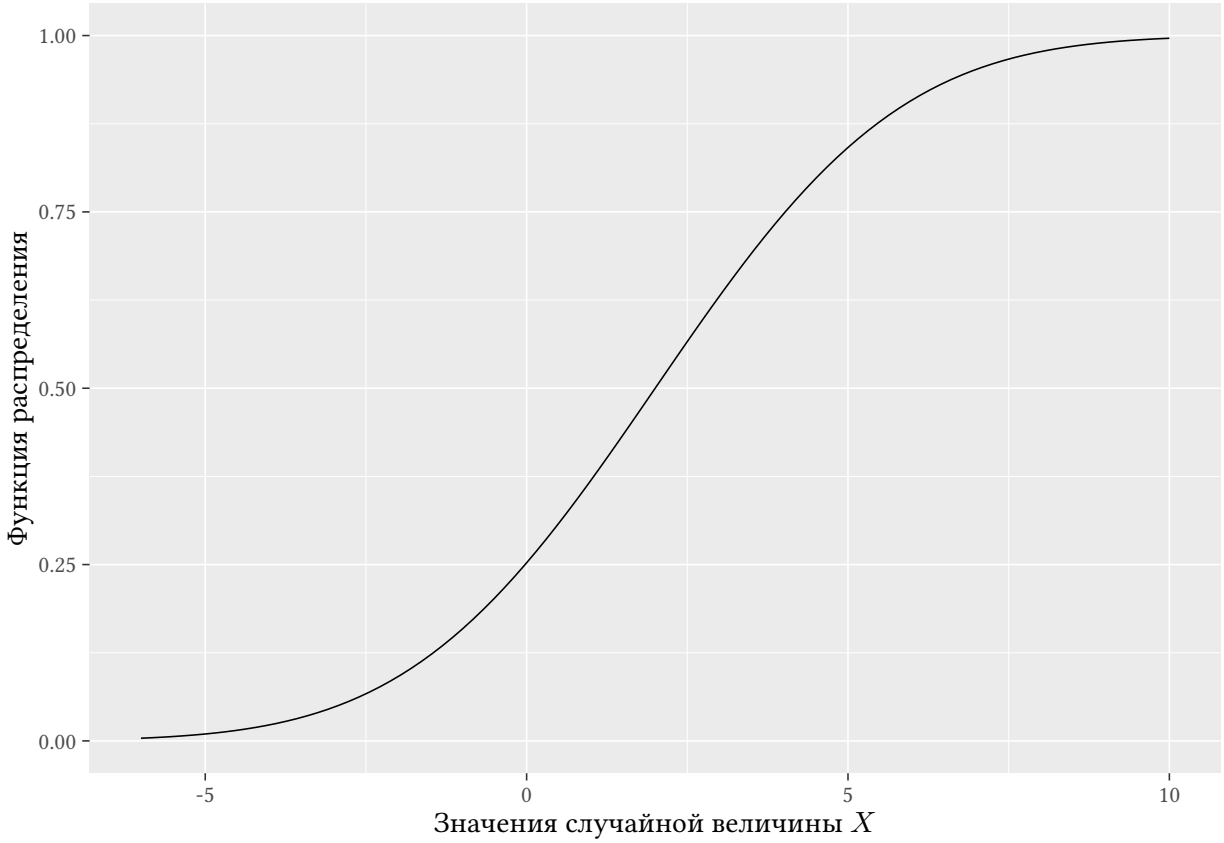
Для дискретных распределений с буквы d начинается название функции, возвращающей вероятность получить заданное значение. Найдём для случайной величины W, имеющей Пуассоновское распределение с параметром $\lambda = 2$ вероятность $P(W = 3)$:

```
dpois(3, lambda = 2)
```

```
## [1] 0.180447
```

- Функция распределения, $F(t) = P(X \leq t)$ — pnorm:

```
x <- seq(from = -6, to = 10, length.out = 100)
y <- pnorm(x, mean = 2, sd = 3)
qplot(x = x, y = y, geom = "line") + xlab("Значения случайной величины $X$") + ylab("Функция распределения")
```



- Квантильная функция или обратная функция распределения, $q(x) = F^{-1}(x)$ — qnorm:

Найдём перцентили 5%, 15% и 90% для нормального $N(2; 9)$ распределения:

```
x <- c(0.05, 0.15, 0.9)
qnorm(x, mean = 2, sd = 3)
```

```
## [1] -2.934561 -1.109300  5.844655
```

Иногда бывает полезно получить случайную выборку из заданного вектора без повторений:

```
sample(1:100, size = 20)
```

```
##  [1] 69 76 46 94 23 38 81 41  9 29 74 21 80 71 63 77 12 96 65 82
```

Или с повторениями:

```
sample(c("Орёл", "Решка"), size = 10, replace = TRUE)
```

```
##  [1] "Решка" "Орёл"  "Решка" "Орёл"  "Орёл"  "Решка" "Решка" "Решка"
##  [9] "Орёл"  "Решка"
```

Можно добавить неравные вероятности:

```
sample(c("Орёл", "Решка"), size = 10, replace = TRUE, prob = c(0.3, 0.7))
```

```
##  [1] "Орёл"  "Орёл"  "Орёл"  "Решка" "Решка" "Решка" "Решка" "Орёл"
##  [9] "Решка" "Решка"
```

Если выполнить команду rnorm(10, mean = 2, sd = 3) на двух разных компьютерах или два раза на одном и том же, то результат будет разный. Не зря же они случайные :) Однако генерирование случайных величин никак не противоречит идее абсолютно точной воспроизводимости исследований. Для того, чтобы получились одинаковые результаты, необходимо синхронизировать генераторы случайных чисел на этих двух компьютерах. Делается это путём задания **зерна** генератора случайных чисел (seed). Зерно также называют **стартовым значением**. В качестве зерна подойдёт любое целое число.

И в результате запуска кода

```
set.seed(42)
rnorm(1, mean = 2, sd = 3)
```

```
## [1] 6.112875
```

все компьютеры выведут число 6.112875.

> Если код содержит генерерирование случайных чисел, то необходимо задавать зерно генератора случайных чисел!

3.2. Базовые статистические тесты

...

3.3. Множественная регрессия

...

Эконометристы любят копаться в остатках :)

https://www.r-bloggers.com/visualising-residuals/

3.4. Квантильная регрессия

Незаслуженно забытой оказывается квантильная регрессия. Коэнкер (ссылка) утверждает, что развитие эконометрики началось именно с квантильной регрессии. Для оценок квантильной регрессии не существует формул в явном виде, поэтому она проиграла классической регрессии среднего с формулой $\hat{\beta} = (X'X)^{-1}X'y$. Сейчас компьютер позволяет начихать на отсутствие явных формул :)

...

3.5. Инструментальные переменные

Каждый исследователь мечтает обнаружить не просто статистическую связь, а причинно-следственную. К сожалению, это не так просто. Фиксируя количество людей с зонтиками на улице и количество осадков но не зная физическую природу дождя, невозможно определить, вызывают ли люди с зонтиками дождь или наоборот. Для выяснения причинно следственных связей необходим случайный эксперимент. Например, можно выбрать несколько случайных дней в году и выгнать толпу знакомых с зонтами на улицу, а затем посмотреть, было ли больше осадков в эти дни.

Инструментальные переменные часто используются исследователями при поиске причинно следственных связей, но это ни в коем случае не означает, что введение инструментальных переменных само по себе гарантирует нахождение причинно-следственной связи!

> Для обнаружения причинно-следственной связи необходимо либо использовать данные случайного эксперимента, либо прекрасно разбираться в закономерностях происходящего.

Этот раздел не о том, как обнаружить причинно-следственную связь, а о том, как реализовать метод инструментальных переменных в R и как его проинтепретировать.

3.6. Гетероскедастичность

...

3.7. Работа с качественными переменными

...

3.8. Логит и пробит с визуализацией

...

3.9. Метод главных компонент

...

3.10. Мультиколлинеарность

...

3.11. LASSO

...

3.12. Метод максимального правдоподобия

...

3.13. Метод опорных векторов

...

3.14. Случайный лес

...

3.15. Экспоненциальное сглаживание

...

3.16. ARMA модели

...

3.17. GARCH

...

3.18. VAR и BVAR

...

Я не верю в пользу от структурных BVAR, поэтому их здесь нет :)

3.19. Панельные данные

...

3.20. Байесовский подход: первые шаги

...

3.21. Байесовский подход: STAN

...

3.22. Карты

Где карта, Билли?

3.23. Дифференциальные уравнения

...

3.24. Задачи оптимизации

...