Учебник по языку R для начинающих

Борис Демешев

2016-09-14

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()
  # другую тему для ggplot2 выставить?
}
```

Данная версия книги скомпилирована для latex.

```
library("ggplot2") # графики
library("sandwich") # оценка Var для гетероскедастичности
library("lmtest") # тест Бройша-Пагана
library("dplyr") # манипуляции с данными
library("broom")
library("data.table")
library("reshape2")
library("tidyr")
```

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

Windows и Macos: заходим на официальный сайт и скачиваем.

Linux на примере Ubuntu 16.04 Xenial:

```
sudo echo "deb http://cran.rstudio.com/bin/linux/ubuntu xenial/" | sudo tee -a /etc/apt/sources.list
gpg --keyserver keyserver.ubuntu.com --recv-key E084DAB9
gpg -a --export E084DAB9 | sudo apt-key add -
sudo apt-get update
sudo apt-get install r-base r-base-dev
```

Вместо классического R можно поставить конкурирующий дистрибутив R от Microsoft, MRO. Отличия MRO от классического R: - работает только с 64-битным процессором - использует быструю библиотеку MKL для линейной алгебры - по умолчанию ставит не самые свежие пакеты, а версии пакетов выпущенные на заданную дату. Это удобная особенность позволяет легко воспроизвести результаты работы, даже если какой-нибудь пакет кардинально изменился.

Опытные пользователи могут самостоятельно настроить R на использование более быстрых библиотек для линейной алгебры.

Macos: настраиваем оптимизированную под мак библиотеку:

```
cd /Library/Frameworks/R.framework/Resources/lib
ln -sf /System/Library/Frameworks/Accelerate.framework/Frameworks/vecLib.framework/Versions/Current/libBLAS.dyl
```

Macos: восстанавливаем дефолтную библиотеку R:

```
cd /Library/Frameworks/R.framework/Resources/lib
ln -sf libRblas.0.dylib libRblas.dylib
```

На линуксе можно поэкспериментировать с выбором разных BLAS

1.1.2. Rstudio

Rstudio — это гламурная оболочка, позволяющая удобно работать с R. Никаких новых возможностей к R не добавляет, но делает многие вещи гораздо более удобными, особенно для новичка.

Оболчка Rstudio открытая и бесплатная. Есть платная версия Rstudio, которая включает в себя поддержку пользователей. И иногда Rstudio путают с программой для восстановления данных R-studio, которая полностью платная.

Текущую версию Rstudio для всех платформ можно ...

Любители заглянуть вперёд и опробовать новые фишки могут поставить preview-версию Rstudio ... Теоретически подобная бета-версия может быть менее устойчивой чем обычная, но по моему опыту никаких проблем со стабильностью не было.

1.1.3. LaTeX

Латех — это система для создания красивых структурированных текстов. Пользоваться Вордом для написания курсовых или научных статей — это дурной вкус. Латех дарит возможность писать красивые математические формулы. В силу простоты текстового формата для хранения документов Латех идеально подходит для взаимодействия с другими программами. Связав R и Латех с помощью грамотного программирования, можно в один клик обновлять сложные отчёты при получении новых данных или избегать ошибок, связанных с копированием результатов вычислений в презентацию.

Под Linux основной дистрибутив латеха — это TeXLive.

После ручной инсталляции нам надо убедить Ubuntu, что ей не надо ставить латех из своего репозитория:

```
sudo apt-get install equivs freeglut3
mkdir -p /tmp/tl-equivs && cd /tmp/tl-equivs
wget http://www.tug.org/texlive/files/debian-equivs-2016-ex.txt
cp -f debian-equivs-2016-ex.txt texlive-local
equivs-build texlive-local
sudo dpkg -i texlive-local_2016-2_all.deb
```

Под Macos основной дистрибутив латеха — это Mactex.

Под Windows существует два конкурирующих дистрибутива — TexLive и Miktex. Я лично рекомендую TexLive, но Miktex тоже подойдёт.

Важно ставить полную версию латеха со всеми пакетами. Это может потребовать более часа, и занять несколько гигабайт жёсткого диска, но красивые документы того стоят.

Если нет необходимости делать красивые структурированные документы в pdf формате, то без Латеха можно обойтись.

В данном учебнике не будет серьёзного введения в латех, мы только затронем вопрос взаимодействия R-латех.

Хорошими введениями в тех будут: Воронцов, ... [1]

1.1.4. git-клиент

ubuntu

```
git config --global user.name "Ivan Ivanov"
git config --global user.email Ivan_Ivanov@ivanov_mail.com
```

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 можно на atom.io. Изнутри самого редактора Atom можно установить кучу плагинов. С помощью плагинов можно подсвечивать синтаксис различных языков или превратить Atom в приличную среду разработки :)

1.1.6. STAN

...

[1] Единственным неудачным советом в книжке Воронцова, пожалуй, является выбор кодировки CP1251, правильнее выбирать UTF-8.

1.1.7. jupyter

Скачиваем дистрибутив Anaconda.

Под Ubuntu и Macos запускаем R из командной строки. Именно из командной строки, а не из под Rstudio, иначе чуда не произойдёт.

Далее инструкция с github.com/IRkernel/IRkernel:

```
install.packages(c('repr', 'IRdisplay', 'crayon', 'pbdZMQ', 'devtools'))
devtools::install_github('IRkernel/IRkernel')
IRkernel::installspec()  # to register the kernel in the current R installation
```

Из командной строки запускаем jupyter:

```
jupyter notebook
```

Для корректного сохранения блокнота в pdf на Ubuntu/Macos

```
pip install nbbrowserpdf
```

Чтобы работало сохранение pdf с русскими буквами нужно руками подменить дефолтный шаблон.

1.1.8. LyX

Многие пользователи привыкшие к Ворду тяжело переходят на латех. Ведь в текстовом редакторе не видно формулу в привычном виде, а виден только её код. Редактор LyX заполняет эту нишу: используя за кадром тех, он показывает пользователю формулы, таблицы и рисунке в естественном виде. При этом LyX отлично взаимодействует с R.

Недостатком LyX, пожалуй, является его невысокая популярность. Пользователи, боящиеся латеха, сидят в Ворде, а те, кто понял прелести латеха, уже не хотят ничего вспомогательного :)

Windows и Macos: скачиваем с официального сайта.

Ubuntu:

```
sudo add-apt-repository ppa:lyx-devel/release
sudo apt-get update
sudo apt-get install lyx
```

1.1.9. Шрифты

Просмотрщик шрифтов Ubuntu (16.04)

```
sudo add-apt-repository ppa:font-manager/staging
sudo apt-get update
sudo apt-get install font-manager
```

В pdf-версии данного учебника используется шрифт Linux Libertine. Он открытый и бесплатный и содержит буквы как русского, так и многих других алфавитов. Скачать его можно здесь: ...

Шрифты могут быть записаны в разных форматах. Основные — это TTF, OTF и Post-script. Рядовой пользователь может смело выбирать TTF или OTF, а Post-script нужен скорее дизайнерам и типографам.

1.1.10. Pandoc

Pandoc — это конвертер текстовых форматов. Он умеет превращать файлы в формате маркдаун во всё, что движется: tex, pdf, docx, epub, html…

Ubuntu (16.04):

```
sudo apt-get install pandoc pandoc-citeproc
```

1.1.11. 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] 0.3484626 0.8518711 0.8754703 0.4079006 0.4753590 0.3851461 0.7262199
```

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] -0.08058304  0.40790056 -2.00467553
```

Можно узнать, сколько значений s больше нуля:

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

```
## [1] 7
```

Еще полезная штучка — количество элементов в векторе:

```
length(b)
```

```
## [1] 7
```

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 × 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 × 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[2] 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[3] 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.

1.4. 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.

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

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

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:

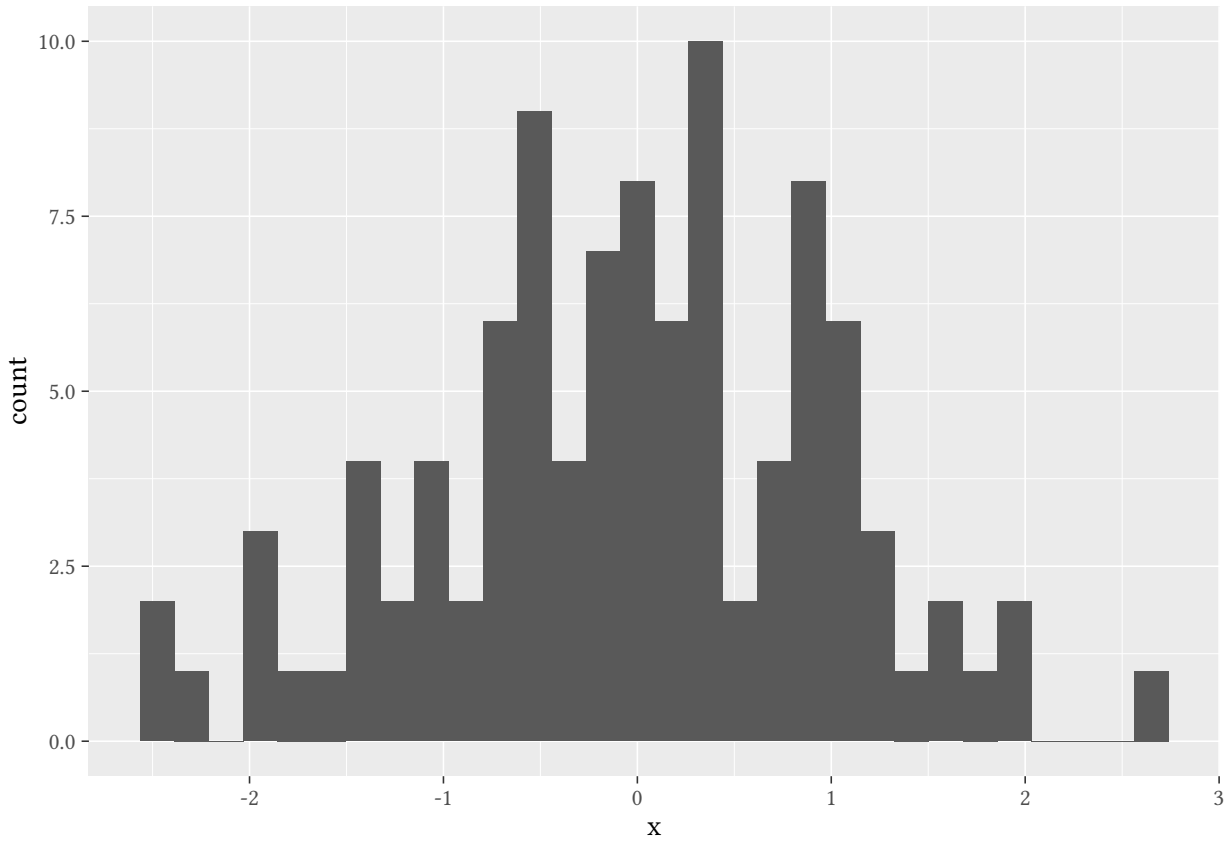
…

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)
```

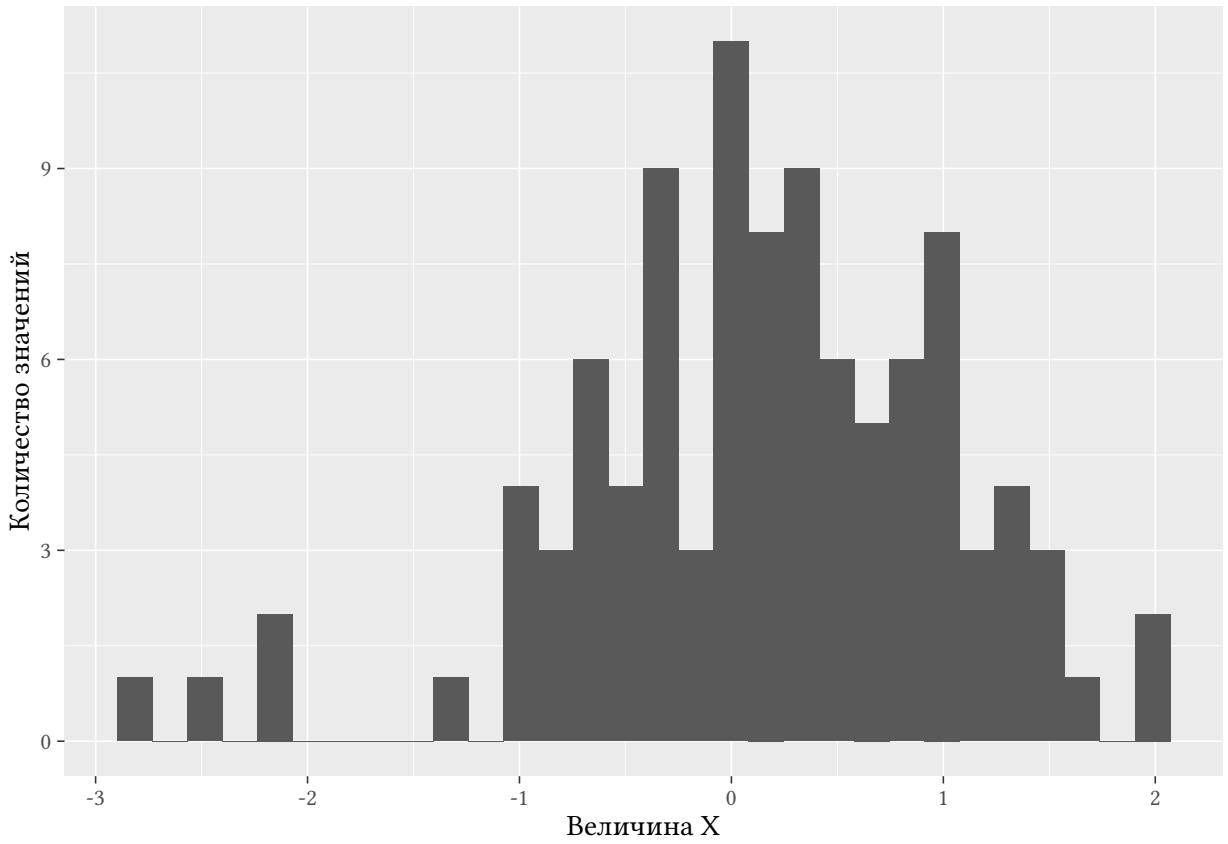
```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



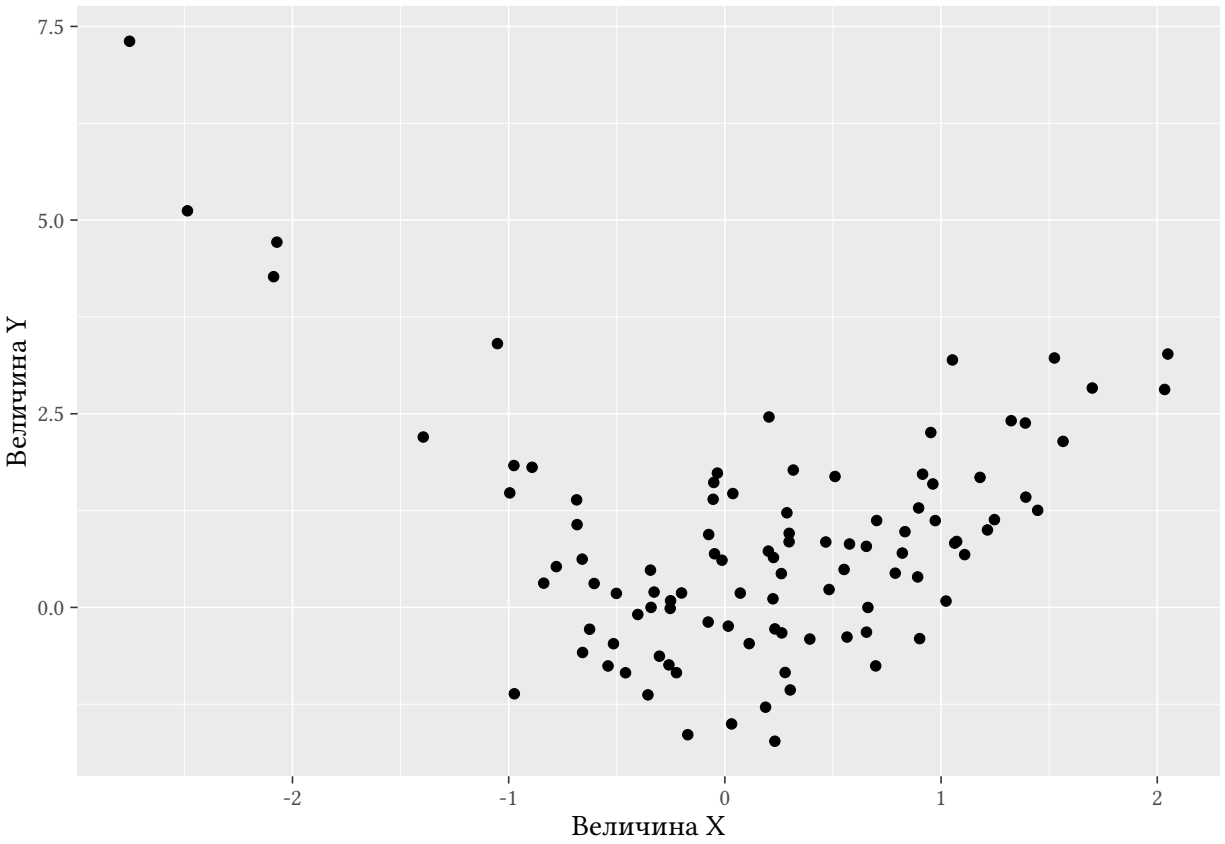
Если же ты хочешь использовать команды некоторого пакета много раз, то проще **подключить** пакет командой library(). При этом не надо будет каждый раз набирать название пакета и двойное двоеточие. Можно просто использовать команды из этого пакеты:

```
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.

Для установки различных пакетов, работающих с Интернетом под Linux может потребоваться установка дополнительных библиотек:

```
sudo apt-get install libssl-dev libcurl4-openssl-dev libxml2-dev
```

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)

Основной объект с которым приходится работать — это табличка с данными. Табличка с данными представляет собой простую двумерную таблицу. Переменные размещаются по столбцам. Каждый столбец может быть своего типа: скажем в одном — числа, а в другом — названия предприятий. Если значение в клетке неизвестно, то там стоит специальное значение NA.

Отличие от матрицы от таблицы с данными состоит в том, что в матрице все клеточки одинакового типа (все — числовые или все — текстовые), а в таблице с данными каждый столбец может быть своего типа.

При обработке данных нам помогут рабочие лошадки:

1. Быстрый взгляд на табличку
2. Отбор наблюдений
3. Отбор и переименование переменных
4. Преобразование и создание новых переменных
5. Агрегирование таблиц
6. Преобразование широких и длинных таблиц

Все эти действия можно сделать в базовом R, но это ужасно неудобно. Жизнь становится прекрасной с пакетами data.table, dplyr, reshape2, broom и tidyr.

Седлаем наших верных коней:

```
library(”dplyr”)
library(”data.table”)
library(”reshape2”)
```

Погнали!

Пакет data.table работает быстрее dplyr, а у dplyr более приятный синтаксис.

Перед работой с табличкой данных с помощью data.table требуется присвоить ей специальный класс:

```
cars2 <- data.table(cars)
```

Содержимое таблиц cars и cars2 абсолютно одинаково, просто они чуть по-разному хранятся и обрабатываются компьютером.

1.9.1. Быстрый взгляд на табличку

С помощью базового R:

```
str(mtcars)
```

```
## ’data.frame’:    32 obs. of  11 variables:
##  $ mpg : num  21 21 22.8 21.4 18.7 18.1 14.3 24.4 22.8 19.2 ...
##  $ cyl : num  6 6 4 6 8 6 8 4 4 6 ...
##  $ disp: num  160 160 108 258 360 ...
```

```
##  $ hp  : num  110 110 93 110 175 105 245 62 95 123 ...
##  $ drat: num  3.9 3.9 3.85 3.08 3.15 2.76 3.21 3.69 3.92 3.92 ...
##  $ wt  : num  2.62 2.88 2.32 3.21 3.44 ...
##  $ qsec: num  16.5 17 18.6 19.4 17 ...
##  $ vs  : num  0 0 1 1 0 1 0 1 1 1 ...
##  $ am  : num  1 1 1 0 0 0 0 0 0 0 ...
##  $ gear: num  4 4 4 3 3 3 3 4 4 4 ...
##  $ carb: num  4 4 1 1 2 1 4 2 2 4 ...
```

С помощью dplyr:

```
glimpse(mtcars)
```

```
## Observations: 32
## Variables: 11
## $ mpg  <dbl> 21.0, 21.0, 22.8, 21.4, 18.7, 18.1, 14.3, 24.4, 22.8, 19....
## $ cyl  <dbl> 6, 6, 4, 6, 8, 6, 8, 4, 4, 6, 6, 8, 8, 8, 8, 8, 8, 4, 4, ...
## $ disp <dbl> 160.0, 160.0, 108.0, 258.0, 360.0, 225.0, 360.0, 146.7, 1...
## $ hp   <dbl> 110, 110, 93, 110, 175, 105, 245, 62, 95, 123, 123, 180, ...
## $ drat <dbl> 3.90, 3.90, 3.85, 3.08, 3.15, 2.76, 3.21, 3.69, 3.92, 3.9...
## $ wt   <dbl> 2.620, 2.875, 2.320, 3.215, 3.440, 3.460, 3.570, 3.190, 3...
## $ qsec <dbl> 16.46, 17.02, 18.61, 19.44, 17.02, 20.22, 15.84, 20.00, 2...
## $ vs   <dbl> 0, 0, 1, 1, 0, 1, 0, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1, 1, ...
## $ am   <dbl> 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, ...
## $ gear <dbl> 4, 4, 4, 3, 3, 3, 3, 4, 4, 4, 4, 3, 3, 3, 3, 3, 3, 4, 4, ...
## $ carb <dbl> 4, 4, 1, 1, 2, 1, 4, 2, 2, 4, 4, 3, 3, 3, 4, 4, 4, 1, 2, ...
```

С помощью broom

```
glance(mtcars)
```

```
##   nrow ncol complete.obs na.fraction
## 1   32   11          32           0
```

и короткие описательные статистики:

```
tidy(mtcars)
```

```
##    column  n       mean         sd  median     trimmed        mad    min
## 1     mpg 32  20.090625   6.0269481  19.200  19.6961538   5.4114900 10.400
## 2     cyl 32   6.187500   1.7859216   6.000   6.2307692   2.9652000  4.000
## 3    disp 32 230.721875 123.9386938 196.300 222.5230769 140.4763500 71.100
## 4      hp 32 146.687500  68.5628685 123.000 141.1923077  77.0952000 52.000
## 5    drat 32   3.596563   0.5346787   3.695   3.5792308   0.7042350  2.760
## 6      wt 32   3.217250   0.9784574   3.325   3.1526923   0.7672455  1.513
## 7    qsec 32  17.848750   1.7869432  17.710  17.8276923   1.4158830 14.500
## 8      vs 32   0.437500   0.5040161   0.000   0.4230769   0.0000000  0.000
## 9      am 32   0.406250   0.4989909   0.000   0.3846154   0.0000000  0.000
## 10   gear 32   3.687500   0.7378041   4.000   3.6153846   1.4826000  3.000
## 11   carb 32   2.812500   1.6152000   2.000   2.6538462   1.4826000  1.000
##        max   range      skew    kurtosis         se
```

```
## 1   33.900  23.500  0.6106550 -0.37276603  1.06542396
## 2    8.000   4.000 -0.1746119 -1.76211977  0.31570933
## 3  472.000 400.900  0.3816570 -1.20721195 21.90947271
## 4  335.000 283.000  0.7260237 -0.13555112 12.12031731
## 5    4.930   2.170  0.2659039 -0.71470062  0.09451874
## 6    5.424   3.911  0.4231465 -0.02271075  0.17296847
## 7   22.900   8.400  0.3690453  0.33511422  0.31588992
## 8    1.000   1.000  0.2402577 -2.00193762  0.08909831
## 9    1.000   1.000  0.3640159 -1.92474143  0.08820997
## 10   5.000   2.000  0.5288545 -1.06975068  0.13042656
## 11   8.000   7.000  1.0508738  1.25704307  0.28552971
```

1.9.2. Отбор наблюдений

С помощью dplyr:

```
head(cars)
```

```
##   speed dist
## 1     4    2
## 2     4   10
## 3     7    4
## 4     7   22
## 5     8   16
## 6     9   10
```

```
cars_filtered <- filter(cars, speed > 10, dist < 20)
head(cars_filtered)
```

```
##   speed dist
## 1    11   17
## 2    12   14
```

Не забывай про эквивалентный трубчатый синтаксис:

```
cars_filtered <- cars %>% filter(speed > 10, dist < 20)
```

> Некоторые другие пакеты помимо dplyr содержат функцию filter. Чтобы избежать конфликта имён можно явно написать dplyr::filter вместо просто filter.

С помощью data.table:

```
cars_filtered <- cars2[speed > 10 & dist < 20]
```

1.9.3. Отбор и переименование переменных:

С помощью dplyr:

```
mtcars2 <- select(mtcars, cylinder = cyl, mpg = mpg, carburator = carb)
```

Помни о трубочках:

```
mtcars2 <- mtcars %>%
  select(cylinder = cyl, mpg = mpg, carburator = carb)
```

> Некоторые другие пакеты помимо dplyr содержат функцию select. Чтобы избежать конфликта имён можно явно написать dplyr::select вместо просто select.

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 творениями мэтров :)

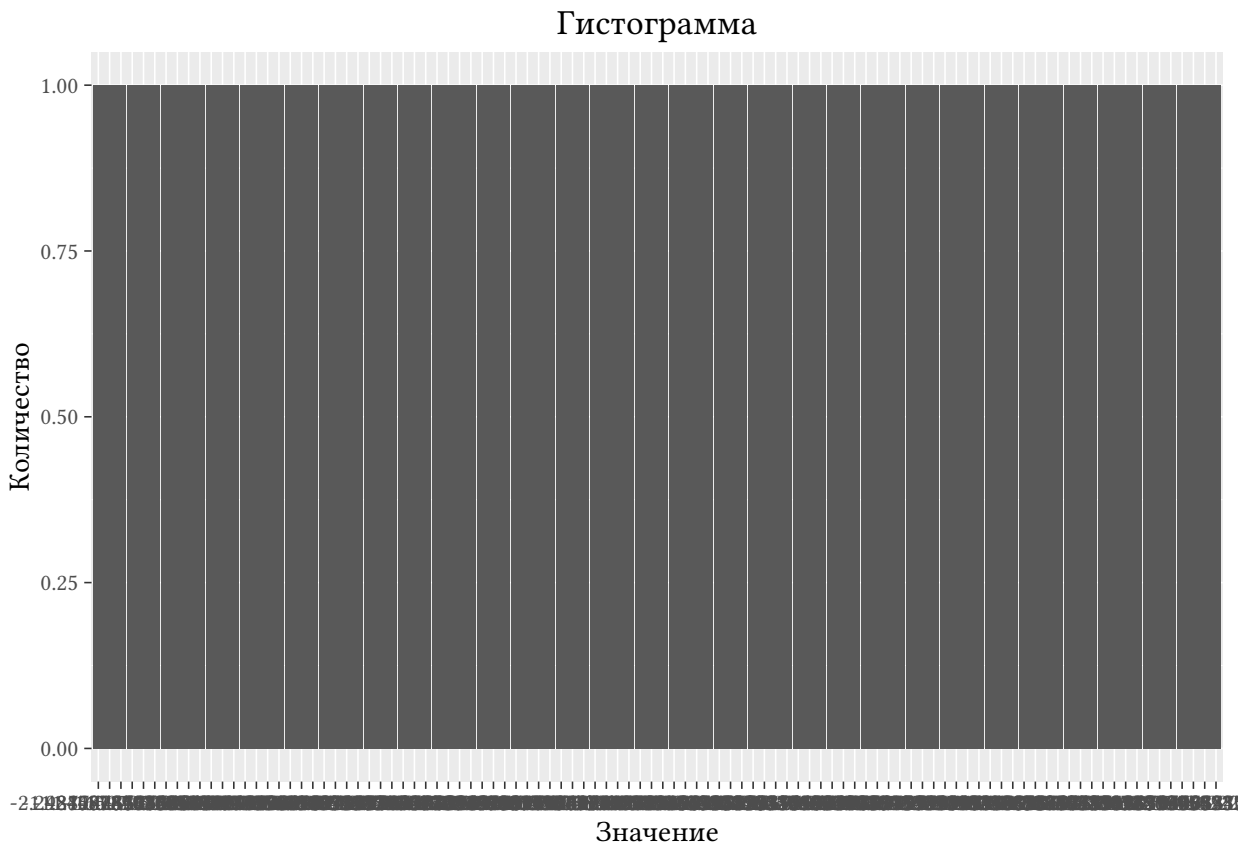
Tufte, link

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

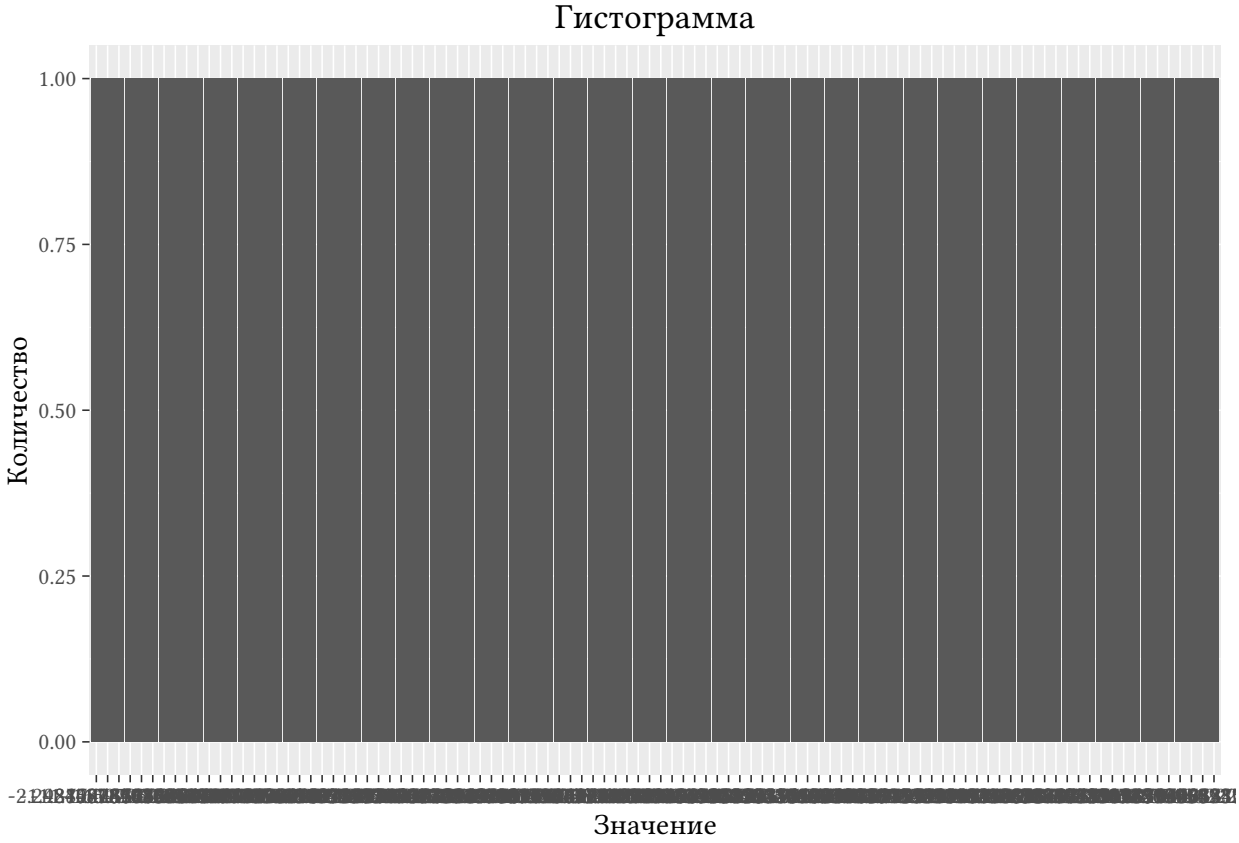
Простенькую гистограмму можно построить, например, с помощью функции qplot() из пакета ggplot2:

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



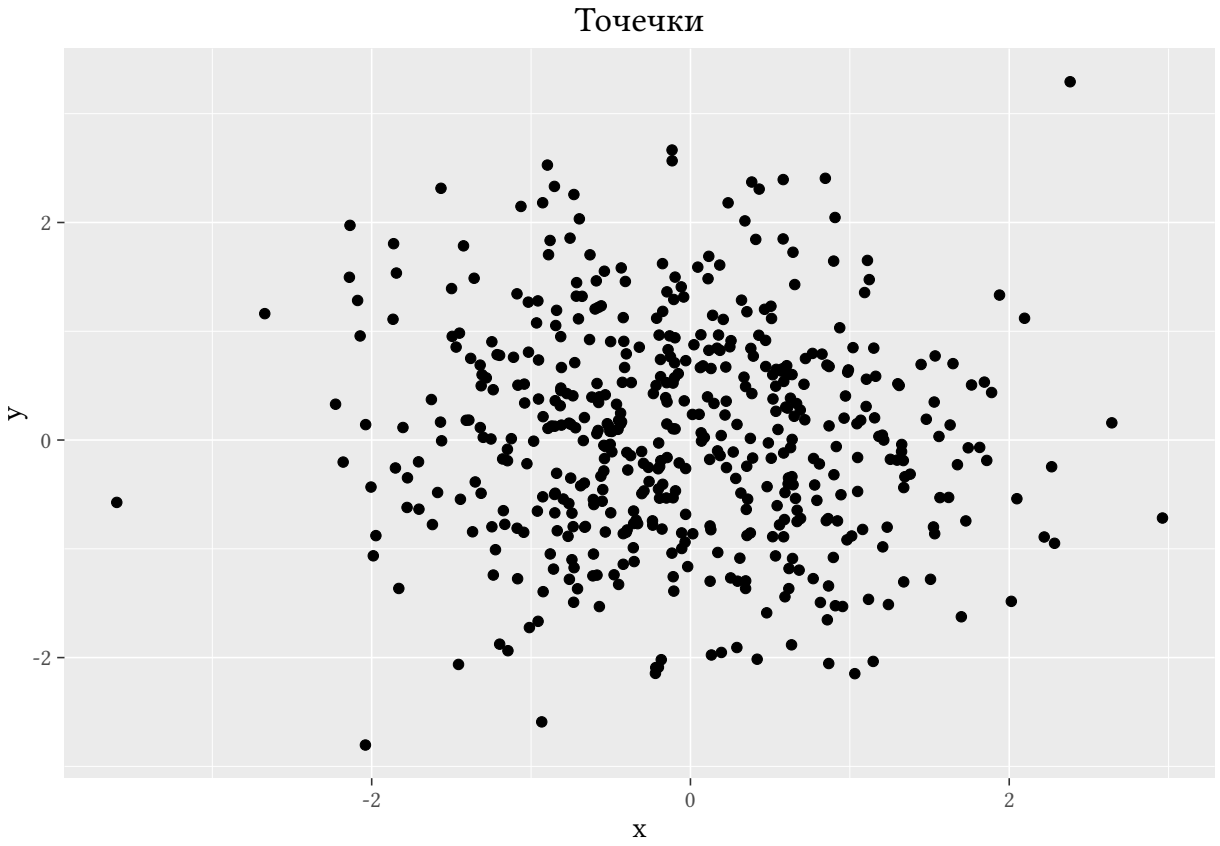
Ту же гистограммку можно построить с помощью функции ggplot() из того же пакета, предварительно преобразовав s в необходимый для ggplot() формат data frame:

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



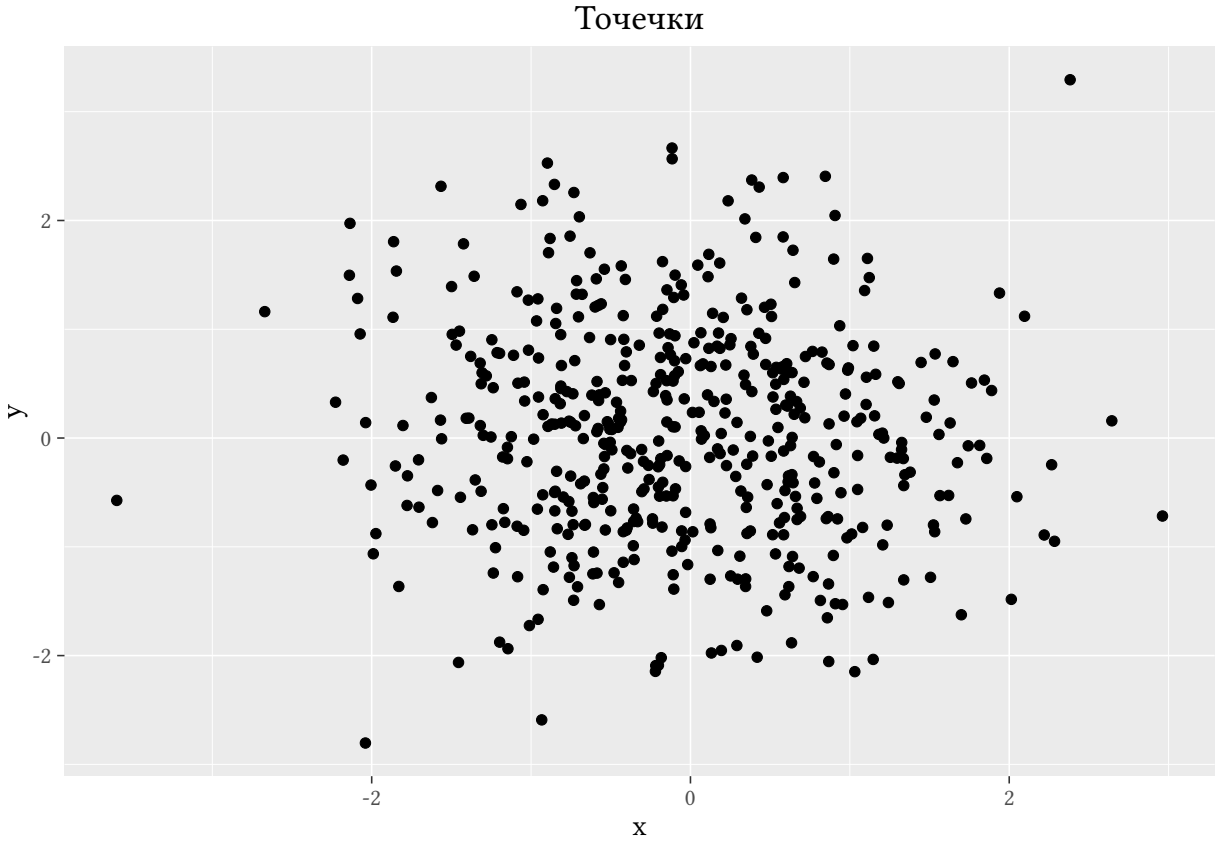
И еще простенький график:

```
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:

- Поисковик он-лайн курсов по 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:

Под macos:

Поставить менеджер пакетов Homebrewer.

```
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. клонируется ветка gh-pages командой

```
git clone -b gh-pages https://${GITHUB_TOKEN}@github.com/${TRAVIS_REPO_SLUG}.git book-output
```

Переменная TRAVIS_REPO_SLUG будет сама определена сервисом Travis и будет равна названию репозитория.

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
```

Для того, чтобы эти опции не были проигнорированы .travis.yml должен содержать строку sudo: required.

Если бинарный пакет слишком старый или вообще отсутствует, можно проинсталлировать его обычным образом, для этого добавляем строки

```
r_packages:
  - devtools
  - rio
```

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

Пора вспомнить о том, что R ориентирован на статистику :)

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]  5.660605  9.519370  9.373239  9.165667  5.838755  7.011480  7.652897
##  [8] 10.044383  7.403635  4.902527
```

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

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

```
##  [1]  4.45243145 -0.07253117 -1.16238827  6.47715155  4.44736874
##  [6]  7.35479212 -1.25095809  3.23920987  1.74533700 -1.92930405
```

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] 0.1246556
```

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] 0.148033
```

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 $N(2; 9)$:

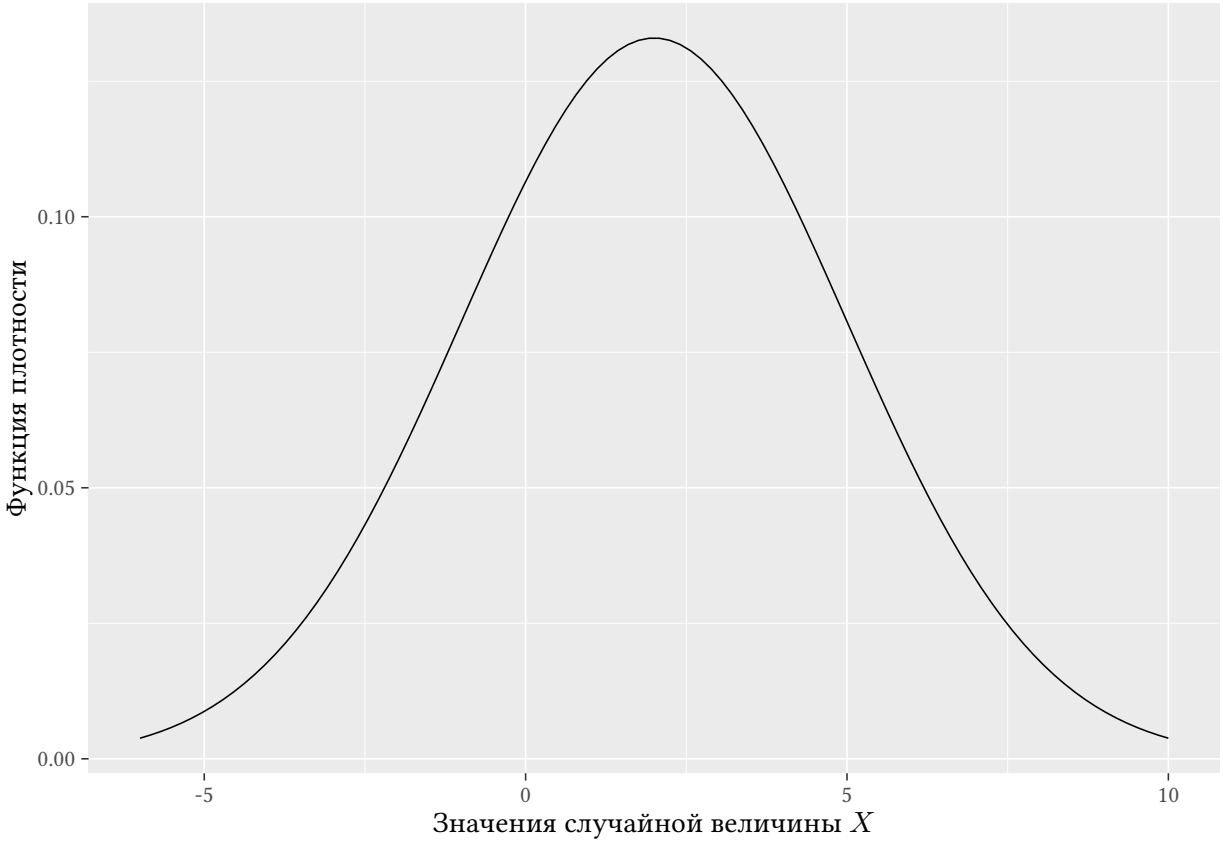
- Функция для создания случайной выборки из нормального $N(2; 9)$ распределения — rnorm:

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

```
##  [1]  3.39083355  0.10326833 -0.61697742  6.58170820 -2.37069964
##  [6] -1.66829234 -0.02299885  1.19205745  6.34309904  1.74290439
```

- 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")
```



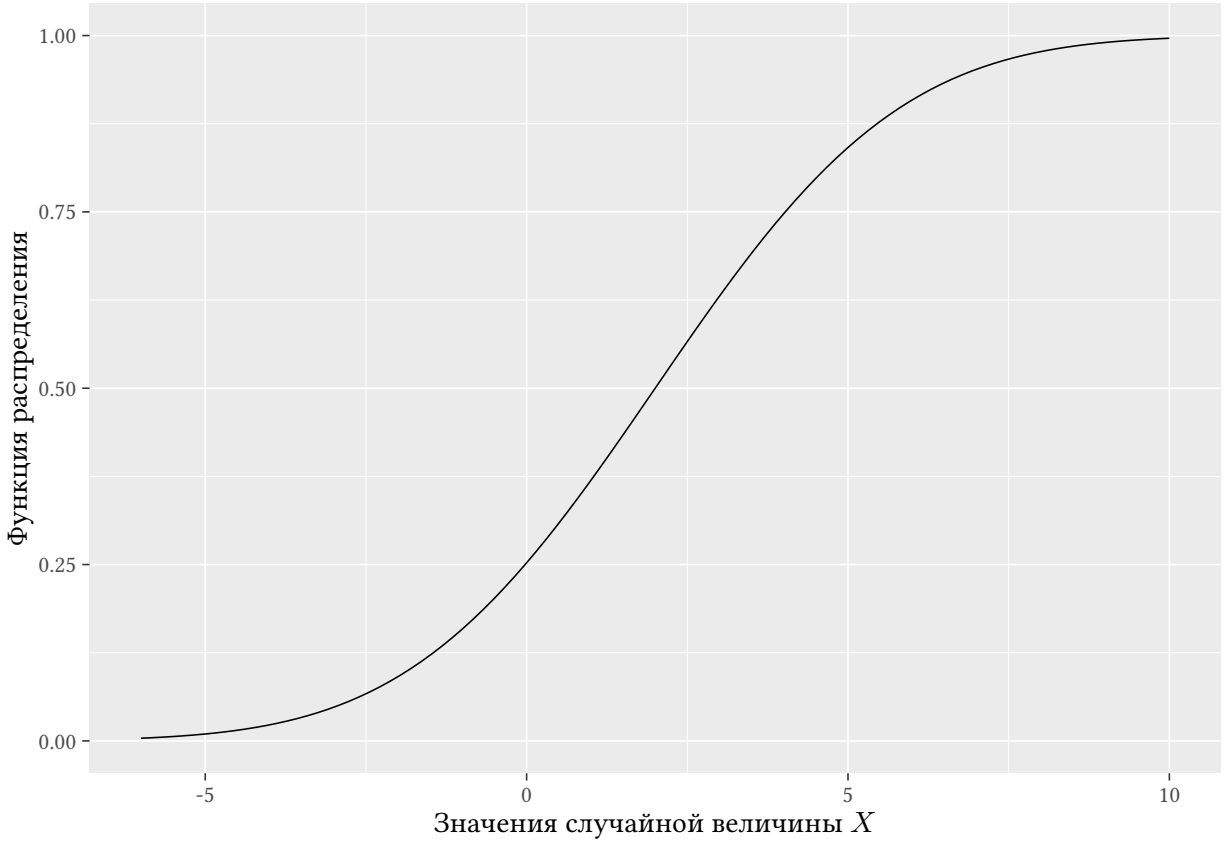
Для дискретных распределений с буквы d начинается название функции, возвращающей вероятность получить заданное значение. Найдём для случайной величины W, имеющей Пуассоновское распределение с параметром $\lambda = 2$ вероятность $P(W = 3)$:

```
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] 88 96 76 57  8 71 58 84 54 29 33 65 39 85 79 78 92  1 35  5
```

Или с повторениями:

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

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

Можно добавить неравные вероятности:

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

```
##  [1] "Решка" "Орёл"  "Решка" "Решка" "Орёл"  "Решка" "Орёл"  "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 готовой 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. Записывая 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.

Использование инструментальных переменных не гарантирует нахождение причинно-следственной связи! Однако они могут быть полезны.

> 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

Подключаем нужные пакеты:

```
library("ggplot2") # графики
library("sandwich") # оценка Var для гетероскедастичности
library("lmtest") # тест Бройша-Пагана
library("dplyr") # манипуляции с данными
```

Условная гетероскедастичность — $Var(\epsilon_i|x_i) \neq const$.

Последствия

- Нарушены предпосылки теоремы Гаусса-Маркова. Получаемые $\hat{\beta}_{ols}$ не являются эффективными.
- Несмещенность оценок $\hat{\beta}_{ols}$ сохраняется
- Нарушены предпосылки для использования t-статистик. При использовании стандартных формул t-статистики не имеют t-распределения. Доверительные интервалы и проверка гипотез по стандартным формулам даёт неверные результаты.

Что делать?

- Если нужны несмещенные оценки — ничего
- Если нужно проверять гипотезы или строить доверительные интервалы, то нужно использовать правильную формулу для $\widehat{Var}(\hat{\beta})$.
- Если нужны эффективные оценки и есть представление о том, какого вида может быть гетероскедастичность то можно взвесить наблюдения. Оценить регрессию:

$$\frac{y_i}{\hat{\sigma}_i} = \beta_1 \frac{1}{\hat{\sigma}_i} + \beta_2 \frac{x_i}{\hat{\sigma}_i} + \frac{\epsilon_i}{\hat{\sigma}_i}$$

- Задать другой вопрос:
  - Вместо регрессии $y_i = \beta_1 + \beta_2 x_i + \epsilon_i$ попробовать оценить регрессию $\ln(y_i) = \beta_1 + \beta_2 \ln(x_i) + \epsilon_i$.
  - Оценить квантильную регрессию

Файл с данными по стоимости квартир в Москве доступен по ссылке goo.gl/zL5JQ

```
filename <- "datasets/flats_moscow.txt"
flats <- read.table(filename, header = TRUE)


ggplot(flats) + geom_point(aes(x = totsp, y = price)) +
  labs(x = "Общая площадь квартиры, кв.м.", y = "Цена квартиры, тысяч у.е.",
  title = "Стоимость квартир в Москве")
```

Строим регрессию стоимости на общую площадь.

```
m1 <- lm(price ~ totsp, data = flats)
summary(m1)
```

```
##
## Call:
## lm(formula = price ~ totsp, data = flats)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -120.58  -17.44   -3.56   10.99  444.52
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept) -62.04484    3.71178  -16.72   <2e-16 ***
```

```
## totsp        2.59346    0.04973   52.15   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 33.96 on 2038 degrees of freedom
## Multiple R-squared:  0.5716, Adjusted R-squared:  0.5714
## F-statistic:  2719 on 1 and 2038 DF,  p-value: < 2.2e-16
```

Покажем доверительные интервалы для коэффициентов из регрессии.

```
confint(m1)
```

```
##                2.5 %     97.5 %
## (Intercept) -69.324117 -54.765570
## totsp         2.495927   2.690998
```

Посмотрим на $\widehat{Var}(\hat{\beta}|X)$.

```
vcov(m1)
```

```
##             (Intercept)       totsp
## (Intercept)  13.7772925 -0.180775186
## totsp        -0.1807752  0.002473516
```

Если не хотим смотреть на всё summary, а только на то, что касается бет с крышкой.

```
coeftest(m1)
```

```
##
## t test of coefficients:
##
##               Estimate Std. Error t value  Pr(>|t|)
## (Intercept) -62.044844   3.711778 -16.716 < 2.2e-16 ***
## totsp         2.593462   0.049734  52.146 < 2.2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Как было сказано ранее: если в данных присутствует гетероскедастичность, но нужно проверить гипотезы и строить доверительные интервалы, то нужно использовать правильную оценку ковариационной матрицы, она выглядит так:

$$\widehat{Var}_{HC}(\hat{\beta}|X) = (X'X)^{-1}X'\hat{\Omega}X(X'X)^{-1}$$

$$\hat{\Omega} = diag(\hat{\sigma}_1^2, \ldots, \hat{\sigma}_n^2)$$

Есть разные способы состоятельно оценить отдельные дисперсии $\hat{\sigma}_i^2$, подробно можно почитать в описании пакета sandwich. Наиболее популярный на сегодня — это так называемый “HC3”:

$$\hat{\sigma}_i^2 = \frac{\hat{\epsilon}_i^2}{(1-h_{ii})^2}$$

Здесь h_{ii} — диагональный элемент матрицы-шляпницы (hat matrix), $\hat{y} = Hy$, $H = X(X'X)^{-1}X'$. Матрицу-шляпницу также называют проектором, т.к. она проецирует вектор y на линейную оболочку регрессоров.

Посмотрим на матрицу $\widehat{Var}_{HC}(\hat{\beta}|X)$ в R.

```
vcovHC(m1)
```

```
##             (Intercept)       totsp
## (Intercept)  61.7662920 -0.89503034
## totsp        -0.8950303  0.01303563
```

Применяя правильную $\widehat{Var}_{HC}(\hat{\beta}|X)$, проверим гипотезы.

```
coeftest(m1, vcov = vcovHC(m1))
```

```
##
## t test of coefficients:
##
##              Estimate Std. Error t value  Pr(>|t|)
## (Intercept) -62.04484    7.85915 -7.8946 4.716e-15 ***
## totsp         2.59346    0.11417 22.7151 < 2.2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Доверительный интервал надо строить руками. За основу мы возьмем табличку с правильными стандартными ошибками и прибавим и вычтем их от оценок коэффициентов.

```
conftable <- coeftest(m1, vcov = vcovHC(m1))
ci <- data_frame(estimate = conftable[, 1],
                 se_hc = conftable[, 2],
                 left_95 = estimate - qnorm(0.975)*se_hc,
                 right_95 = estimate + qnorm(0.975)*se_hc)
ci
```

```
## # A tibble: 2 × 4
##    estimate     se_hc    left_95   right_95
##       <dbl>     <dbl>      <dbl>      <dbl>
## 1 -62.044844 7.8591534 -77.448501 -46.641186
## 2   2.593462 0.1141737   2.369686   2.817239
```

Ещё есть малоизвестный пакет Greg Макса Гордона и в нём есть готовая функция:

```
Greg::confint_robust(m1)
```

```
##                  2.5 %     97.5 %
## (Intercept) -77.448501 -46.641186
## totsp         2.369686   2.817239
```

Пакет Greg не доступен на официальном репозитории CRAN, ставится с github командой devtools::install_github("gforge/Greg").

В условиях гетероскедастичности обычная t-статистика не имеет ни t-распределения при нормальных ε_i, ни даже асимптотически нормального. И такая же проблема возникает с

F-статистикой для сравнения вложенных моделей. Отныне привычная F-статистика

$$F = \frac{(RSS_R - RSS_{UR})/r}{RSS_{UR}/(n - k_{UR})}$$

не имеет ни F-распределения при нормальных остатках, ни χ^2-распределения асимптотически. В частности, для проверки незначимости регрессии в целом некорректно использовать F-статистику равную $F = \frac{ESS/(k-1)}{RSS/(n-k)}$.

Если для теста Вальда использовать корректную оценку ковариационной матрицы, то его можно применять в условиях гетероскедастичности. Допустим, мы хотим проверить гипотезу H_0: $A\beta = b$, состоящую из q уравнений, против альтернативной о том, что хотя бы одно равенство нарушено. Тогда статистика Вальда имеет вид:

$$W = (A\beta - b)'(A \cdot \widehat{Var}_{HC}(\hat{\beta})A')^{-1}(A\beta - b)$$

и имеет асимптотически χ^2-распределение с q степенями свободы.

В R реализуется с помощью опции для команды waldtest. Покажем на примере гипотезы о незначимости регрессии в целом:

```
m0 <- lm(price ~ 1, data = flats) # оцениваем ограниченную модель (регрессия на константу)
waldtest(m0, m1, vcov = vcovHC(m1))
```

```
## Wald test
##
## Model 1: price ~ 1
## Model 2: price ~ totsp
##   Res.Df Df      F    Pr(>F)
## 1   2039
## 2   2038  1 515.97 < 2.2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Переходим ко взвешенному МНК. Взвешенный МНК требует знания структуры гетероскедастичности, что редко встречается на практике. Предположим, что в нашем случае $Var(\epsilon_i|totsp_i) = const \cdot totsp_i$.

В R вектор весов weights должен быть обратно пропорционален дисперсиям, т.е. $w_i = 1/\sigma_i^2$.

```
m2 <- lm(price ~ totsp, weights = I(1 / totsp), data =flats)
summary(m2)
```

```
##
## Call:
## lm(formula = price ~ totsp, data = flats, weights = I(1/totsp))
##
## Weighted Residuals:
##     Min      1Q  Median      3Q     Max
## -11.571  -1.994  -0.591   1.235  39.088
##
```

```
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept) -52.50302    3.53967  -14.83   <2e-16 ***
## totsp         2.46290    0.04931   49.95   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 3.58 on 2038 degrees of freedom
## Multiple R-squared:  0.5504, Adjusted R-squared:  0.5501
## F-statistic:  2495 on 1 and 2038 DF,  p-value: < 2.2e-16
```

3.6.1. Тесты на гетероскедастичность

Графическое обнаружение гетероскедастичности:

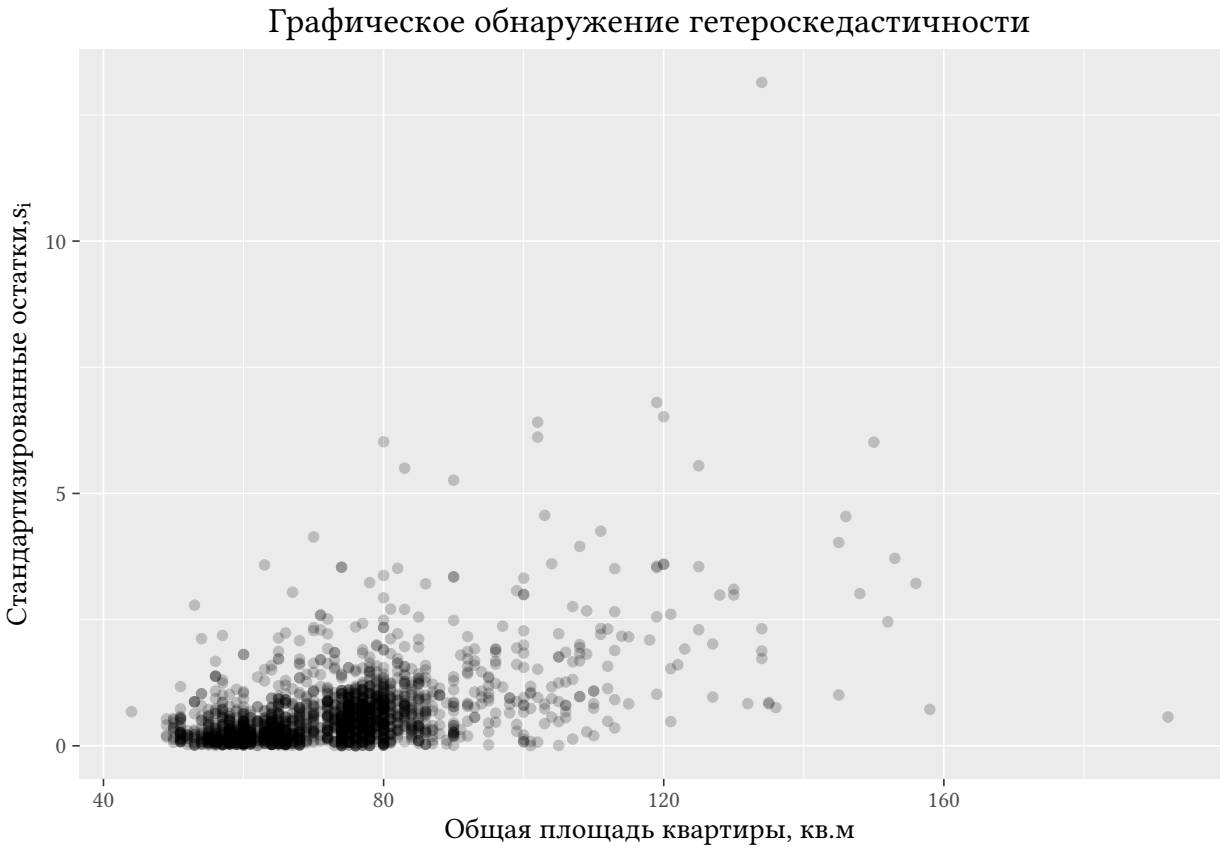
- Оценивается исходная регрессия и из неё получаются остатки $\hat{\varepsilon}_i$.
- Если верить в гомоскедастичность, то дисперсия вектора остатков определяется по формуле $Var(\hat{\varepsilon}|X) = \sigma^2(I - X(X'X)^{-1}X')$. Т.е. дисперсия у остатков разная (!) даже если $Var(\varepsilon_i|X) = \sigma^2$. Поэтому остатки стандартизуют по формуле:

$$s_i = \frac{\varepsilon_i}{\sqrt{\hat{\sigma}^2(1 - h_{ii})}}$$

  Остатки полученные после такого преобразования называют стандартизированными или иногда стьюдентизированными.
- Можно построить график зависимости величины s_i^2 или $|s_i|$ от переменной, предположительно повинной в гетроскедастичности.

В R:

```
m1.st.resid <- rstandard(m1) # получаем стандартизированные остатки
ggplot(aes(x = totsp, y = abs(m1.st.resid)), data =flats) + geom_point(alpha = 0.2) +
  labs(x = "Общая площадь квартиры, кв.м", y = expression(paste("Стандартизированные остатки, ", s[i])),
  title = "Графическое обнаружение гетероскедастичности")
```



Иногда для простоты пропускают второй шаг со стандартизацией остатков и в результате могут ошибочно принять принять разную $Var(\hat{\varepsilon}_i)$ за гетероскедастичность, т.е. за разную $Var(\varepsilon_i)$. Впрочем, если гетероскедастичность сильная, то её будет видно и на нестандартизированных остатках.

3.6.2. Тест Бройша-Пагана, Breusch-Pagan:

Есть две версии теста Бройша-Пагана: авторская и современная модификация.

Предпосылки авторской версии:

- нормальность остатков, $\varepsilon_i \sim N(0, \sigma_i^2)$
- $\sigma_i^2 = h(\alpha_1 + \alpha_2 z_{i2} + \ldots + \alpha_p z_{ip})$
- у функции h существуют первая и вторая производные
- тест асимптотический

Суть теста: Используя метод максимального правдоподобия посчитаем LM-статистику. При верной H_0 она имеет хи-квадрат распределение с $p-1$ степенью свободы.

Оказывается, что LM-статистику можно получить с помощью вспомогательной регрессии. Авторская процедура:

1. Оценивается регрессия $y_i = \beta_1 + \beta_2 x_i + \varepsilon_i$
2. Переходим к $g_i = \frac{n}{SSR}\hat{\varepsilon}_i$
3. Строим регрессию g_i на $\alpha_1 + \alpha_2 z_{i2} + \ldots + \alpha_p z_{ip}$

4. $LM = \frac{ESS}{2}$

Современная модификация выглядит (неизвестный рецензент Коэнкера) так:

1. Оценивается регрессия $y_i = \beta_1 + \beta_2 x_i + \varepsilon_i$
2. Оценивается регрессия $\hat{\varepsilon_i^2}$ на переменные вызывающие гетероскедастичность
3. При верной H_0 асимптотически:
$$nR^2 \sim \chi^2_{p-1},$$
где p — число оцениваемых коэффициентов во вспомогательной регрессии. По смыслу $(p-1)$ — это количество факторов, от которых потенциально может зависеть дисперсия $Var(\varepsilon_i)$.

Тест Бройша-Пагана в R:

```
bptest(m1) # версия Коэнкера
```

```
##
##  studentized Breusch-Pagan test
##
## data:  m1
## BP = 201.95, df = 1, p-value < 2.2e-16
```

```
bptest(m1, studentize = FALSE) # классика Бройша-Пагана
```

```
##
##  Breusch-Pagan test
##
## data:  m1
## BP = 2366, df = 1, p-value < 2.2e-16
```

У современной модификации Бройша-Пагана более слабые предпосылки:

- H_0: $Var(\varepsilon_i) = \sigma^2$, нормальность не требуется
- $E(\varepsilon_i^4) = const$
- тест асимптотический
- (? Коэнкер в статье критикует что-то про мощность ?)

3.6.3. Тест Уайта. White test

Предпосылки:

- H_0: $Var(\varepsilon_i) = \sigma^2$, нормальность не требуется
- $E(\varepsilon_i^4) = const$
- тест асимптотический

Процедура:

1. Оценивается регрессия $y_i = \beta_1 + \beta_2 x_i + \varepsilon_i$
2. Строим регрессию $\hat{\varepsilon_i^2}$ на исходные регрессоры и их квадраты
3. Асимптотически nR^2 имеет хи-квадрат распределение

Тест Уайта в R:

```
bptest(m1, varformula = ~ totsp + I(totsp^2), data = flats)
```

```
##
##  studentized Breusch-Pagan test
##
## data:  m1
## BP = 247.35, df = 2, p-value < 2.2e-16
```

Тест Уайта является частным случаем современной модификации теста Бройша-Пагана. В тесте Бройша-Пагана во вспомогательной регресии можно брать любые объясняющие переменные. В тесте Уайта берутся исходные регрессоры, их квадраты и попарные произведения. Если в R попросить сделать тест Бройша-Пагана и не указать спецификацию вспомогательной регрессии, то он возьмет только все регрессоры исходной.

3.6.4. Тест Goldfeld-Quandt

Предпосылки:

- нормальность остатков, $\varepsilon_i \sim N(0, \sigma_i^2)$
- Наблюдения упорядочены по возростанию гетероскедастичности
- тест точный (неасимптотический)

Процедура:

1. Упорядочить наблюдения в том порядке, в котором подозревается рост гетероскедастичности
2. Выкинуть некий процент наблюдений по середине, чтобы подчеркнуть разницу в дисперсии между начальными и конечными наблюдениями.
3. Оценить исходную модель по наблюдениям из начала выборки и по наблюдениям конца выборки
4. Получить, соответственно, RSS_1 и RSS_2

При верной H_0

$$\frac{RSS_2}{RSS_1} \sim F_{r-k,r-k},$$

где r — размер подвыборки в начале и в конце.

Тест Голдфельда-Квандта в R:

```
flats2 <- flats[order(flats$totsp), ] # сменим порядок строк в табличке h
m3 <- lm(price ~ totsp, data = flats2)
gqtest(m3, fraction = 0.2) # проведем GQ тест выкинув посередине 20% наблюдений
```

```
##
##  Goldfeld-Quandt test
##
## data:  m3
## GQ = 8.2121, df1 = 814, df2 = 814, p-value < 2.2e-16
```

3.6.5. Тестирование и борьба с гетероскедастичностью

Чайники часто используют такой *ошибочный* подход:

Протестировать гетероскедастичность. Если тесты выявили гетероскедастичность, то бороться с ней (использовать устойчивые стандартные ошибки или применять взвешенный мнк). Если тесты не выявили гетероскедастичность, то не бороться с ней.

Этот подход неверен!

Правильно делать так:

- Если есть теоретические основания ожидать гетероскедастичность и наблюдений достаточно, то использовать устойчивые стандартные ошибки вне зависимости от результатов тестирования.
- Если есть теоретические основания ожидать гетероскедастичность и наблюдений мало, то отказаться от девичьих грёз об эффективности оценок и проверке гипотез. Довольствоваться возможной несмещенностью оценок.

Тут обычно спрашивают: “А зачем же тогда тестировать на гетероскедастичность, если это решение ни на что не влияет?”. Ответ прост — чтобы узнать, есть ли гетероскедастичность. :) Впрочем, если есть теоретические основания её ожидать, то можно и не тестировать.

3.6.6. Прогнозирование при гетероскедастичности

При прогнозировании строят доверительный интервал для среднего значения зависимой переменной и предиктивный интервал для конкретного будущего значения. С доверительным интервалом похоже надо код писать руками:

```
# ...
```

С предиктивным интервалом чуть сложнее. Поскольку предиктивный интервал строиться для конкретного наблюдения, то нам надо знать дисперсию этого наблюдения. Идея робастных ошибок состоит в том, чтобы проверять гипотезы не делая точных предположений о структуре гетероскедастичности. Поэтому при использовании робастных стандартных ошибок традиционно предиктивные интервалы не строят.

Хм. А если взять парадигму рандомных регрессоров и оценку безусловной дисперсии?

Если же случилось чудо и структура гетероскедастичности доподлинно известна, то мы спокойно используем взвешенный МНК и указываем вес для новых наблюдений.

```
# ...
```

3.6.7. Дополнительно:

- Описание пакета sandwich с очень правильным изложением современного взгляда на гетероскедастичность
- Breusch, Pagan, Simple test for heteroskedasticity
- White, A Heteroskedasticity-Consistent Covariance Matrix Estimator and a Direct Test for Heteroskedasticity

- Koenker, A note on studentizing a test for heteroscedasticity

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

...