Санкт-Петербургский
государственный
университет

# Знакомство с R

М.А. Варфоломеева, PhD
В.М. Хайтов, к.б.н.

Биологический факультет, СПбГУ

# Почему R это круто?

М.А. Варфоломеева, PhD
В.М. Хайтов, к.б.н.

Биологический факультет, СПбГУ

# Новые впечатления и сообщество единомышленников

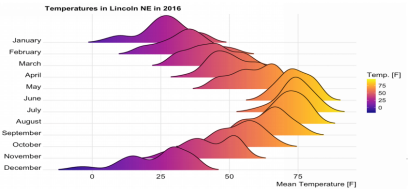

https://www.shutterstock.com/ru/image-photo/businessman-touching-global-network-data-exchanges-566877226

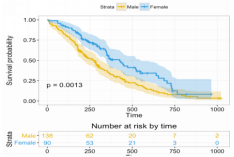
- R — это инструмент познания мира. Вы получите много новых впечатлений и встретите большое и дружественное сообщество единомышленников.

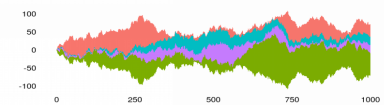
# Огромный набор методов

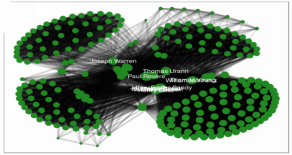

- R — это открытый кросс-платформенный язык статистического программирования.
- В R реализованы практически все статистические методы, как классические, так и ультрасовременные.
- Доступно более 10 000 пакетов и их число растет. К пакетам есть подробные и понятные справки.

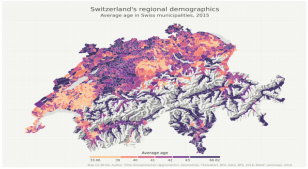
# Визуализация данных

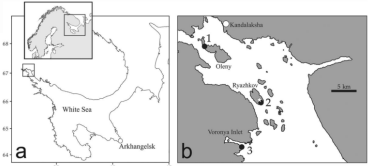



В R можно создавать практически любые иллюстрации:

- графики
- карты
- сложные комбинированные рисунки

https://timogrossenbacher.ch/2016/12/beautiful-thematic-maps-with-ggplot2-only/

http://www.ggplot2-exts.org/gallery/

Khaitov et al, 2018

# Гибкий и воспроизводимый код для анализа данных


https://www.shutterstock.com/ru/image-photo/toy-magician-casting-spell-not-person-425596252?src=9IFUp7OtjFojD9EEo8ZPfQ-1-18

- Функции языка R подобны магическим заклинаниям, они совершают чудеса при обработке данных.
- Ход анализа данных может быть задокументирован. Вы напишете собственные магические тексты в виде кода на R, с помощью которого можно воспроизвести все этапы анализа.

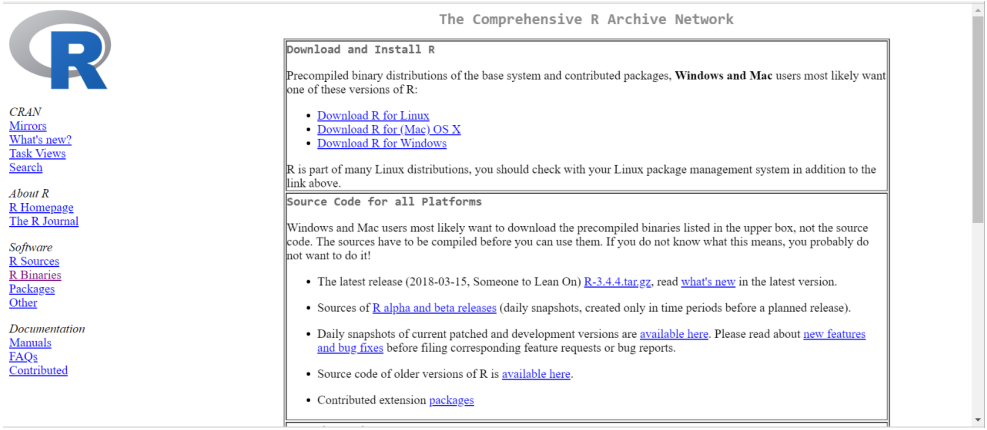
# Установка и настройка R и RStudio

В.М. Хайтов, к.б.н.

Биологический факультет, СПбГУ

# Загрузка R

## Сайт CRAN https://cran.r-project.org/



The Comprehensive R Archive Network

*CRAN*
Mirrors
What's new?
Task Views
Search

*About R*
R Homepage
The R Journal

*Software*
R Sources
R Binaries
Packages
Other

*Documentation*
Manuals
FAQs
Contributed

**Download and Install R**

Precompiled binary distributions of the base system and contributed packages, **Windows and Mac** users most likely want one of these versions of R:

- Download R for Linux
- Download R for (Mac) OS X
- Download R for Windows

R is part of many Linux distributions, you should check with your Linux package management system in addition to the link above.

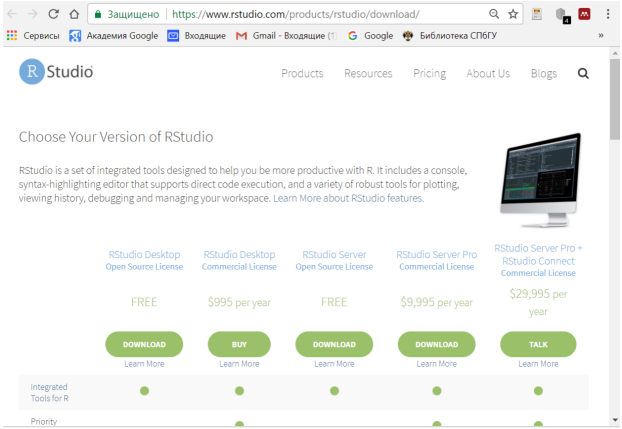
**Source Code for all Platforms**

Windows and Mac users most likely want to download the precompiled binaries listed in the upper box, not the source code. The sources have to be compiled before you can use them. If you do not know what this means, you probably do not want to do it!

- The latest release (2018-03-15, Someone to Lean On) R-3.4.4.tar.gz, read what's new in the latest version.
- Sources of R alpha and beta releases (daily snapshots, created only in time periods before a planned release).
- Daily snapshots of current patched and development versions are available here. Please read about new features and bug fixes before filing corresponding feature requests or bug reports.
- Source code of older versions of R is available here.
- Contributed extension packages

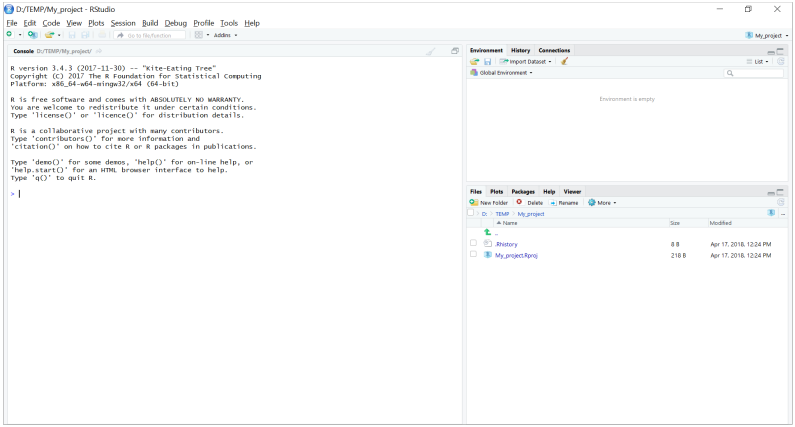
# Загрузка RStudio

https://www.rstudio.com/products/rstudio/download/

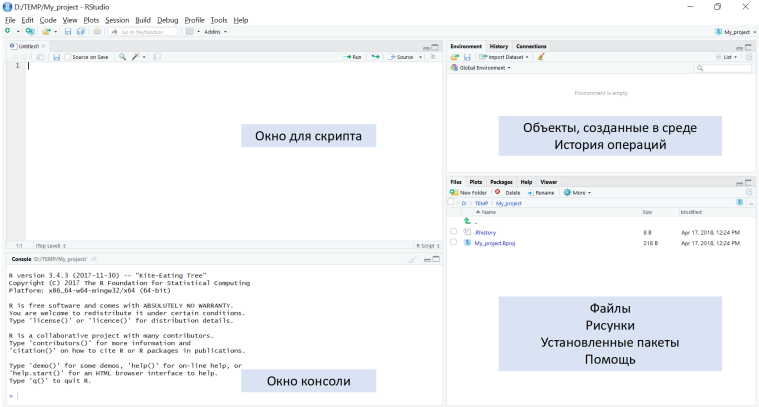


# Начальный вид окон RStudio

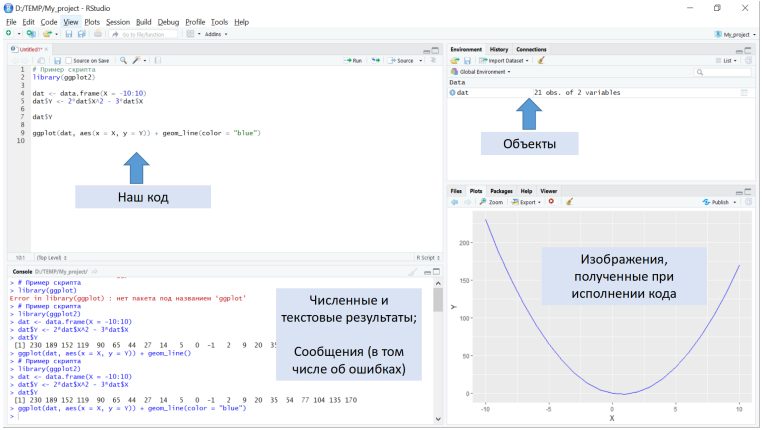
Сразу после открытия создаем окно для скрипта: File → New File → R Script



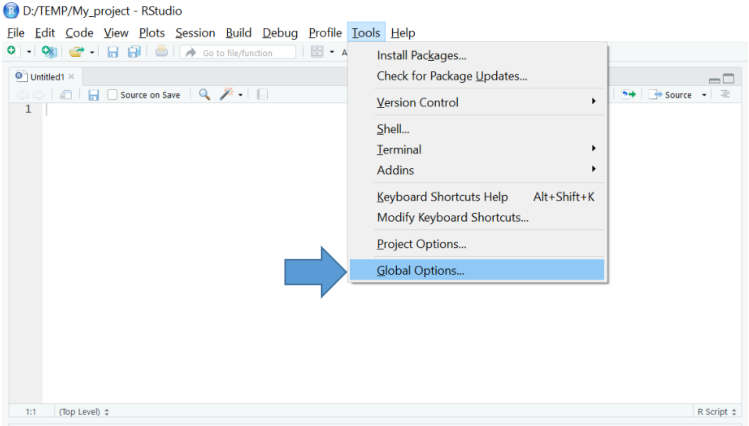
# Четыре основных окна RStudio



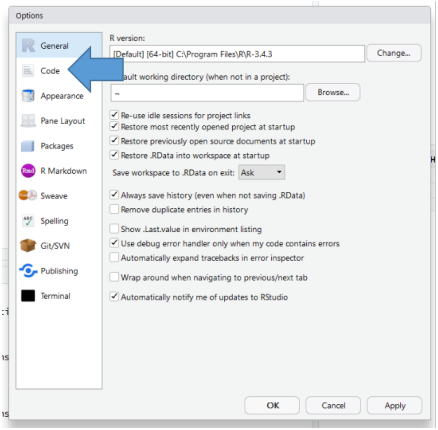
# Вид окон во время работы



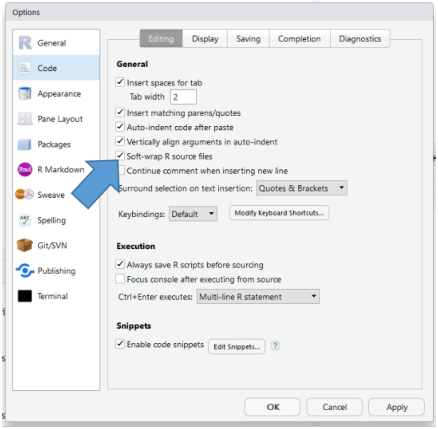
# Важные настройки RStudio



# Настройка окна скрипта



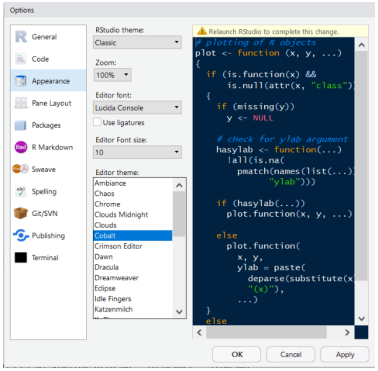
# Перенос в отображении длинных строк




Ставим галочку рядом с опцией

Soft-wrap R source files

# Вид RStudio по своему вкусу




Tools → Global options → Appearance

# Организация рабочего пространства

В.М. Хайтов, к.б.н.

Биологический факультет, СПбГУ

# До начала работы

1. Создайте папку, где будут храниться ВСЕ материалы этого курса. Эта папка будет нашей **рабочей директорией**.

Например:

```
D:\Coursera\Linmodr\
```

В эту папку помещайте ВСЕ файлы со скриптами.

Отсюда же запускайте эти файлы из проводника или иного файл-менеджера.

2. Внутри папки `Linmodr` создайте папку `data`.

В нее помещайте все файлы с данными для анализа.

```
D:\Coursera\Linmodr\data\
```

# Открываем готовый скрипт

Скачайте с сайта файл `first_steps.R` и поместите его в рабочую директорию.

Откройте этот скрипт в `RStudio`

File → Open File

# Кракозябры вместо русских букв?

Нужно поменять кодировку

File → Reopen with Encoding

Из предложенного списка выберете UTF-8.

В том же окне поставьте галочку `"Save as default encoding for source files"`

# Рабочая директория

R всегда “нацелен” на определенную папку в вашей операционной системе — эта папка называется **рабочая директория**.

Это удобно: все файлы рабочей директории можно открыть, указав путь не от корневрй папки системы, а от этой рабочей директории. Это называется **относительный путь**.

# Проверка пути к рабочей директории

По умолчанию `R` считает, что рабочая директория — это основная пользовательская папка в вашей системе.

Для проверки правильности установки рабочей директории в окне скрипта выполните команду `getwd()` (get work directory).

```
getwd()
```

Для этого поставьте текстовый курсор на строчку с этой командой и нажмите `Ctrl + Enter`.

# Установка пути к рабочей директории

Допустим, нашей рабочей директорией должна быть `D:\Coursera\Linmodr`

Способ 1:

Выберите в меню Session → Set working Directory → Choose Directory...

Способ 2:

Используем команду `setwd()`

```
setwd('D:/Coursera/Linmodr/')
```

# Как получить помощь

В.М. Хайтов, к.б.н.

Биологический факультет, СПбГУ

# Просить помощи — это нормально


https://www.shutterstock.com/ru/image-photo/woman-drowns-sea-698646385

Работая в среде R, необходимо постоянно обращаться за помощью.

Три основных способа получения помощи:

- Локальная справка R
- Google
- Stack Overflow

# Локальная справка R

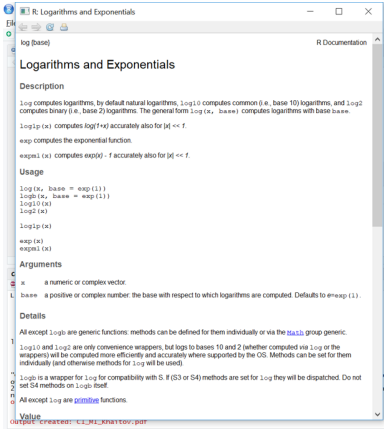
Предположим, вам необходимо получить справку по функции `log()`

```
?log      # Вызов справки через "?"

help(log) # Вызов справки с помощью функции help()
```

То же самое можно сделать, если поставить текстовый курсор на название функции и нажать “F1”.

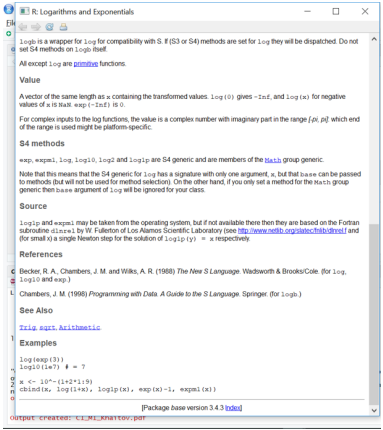
# Окно справки




Разделы:

- Description — общее описание функции
- Usage — шаблон использования функции
- Arguments — аргументы (параметры) функции
- Details — Дополнительные детали

# Окно справки




Разделы:

- Value — результаты работы функции
- References — ссылки на первоисточники
- See Also — похожие функции
- Examples — примеры использования функции

# Код из раздела `Examples` можно выполнять

Строки из раздела “Examples” можно скопировать в буфер обмена, вставить в скрипт и выполнить.

```
log(exp(3))
```

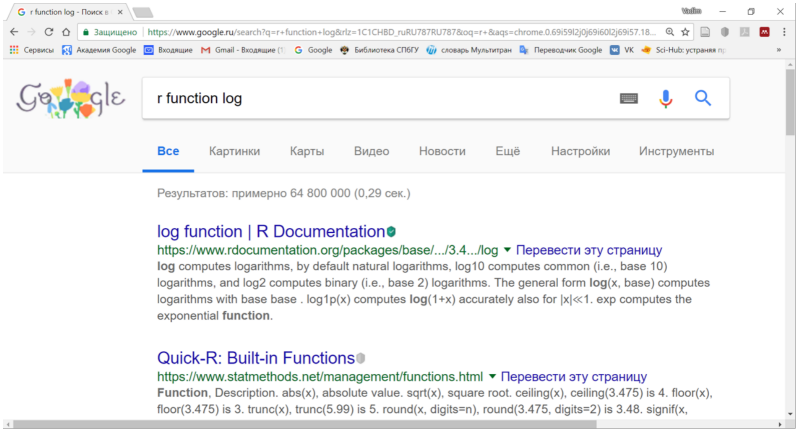
```
# [1] 3
```

```
log10(1e7) # = 7
```

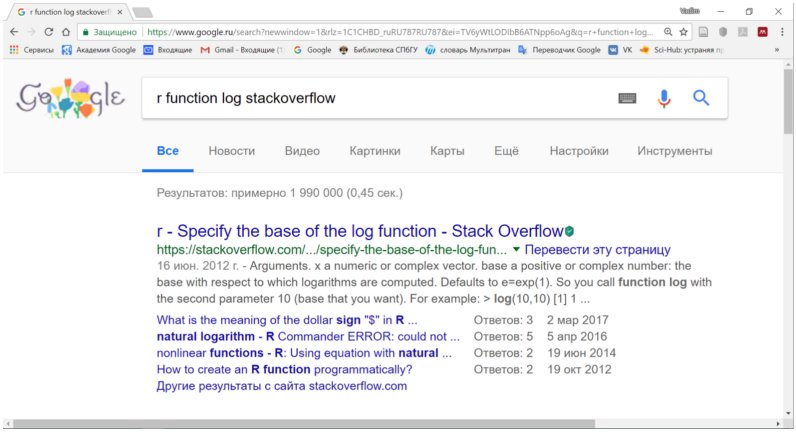
```
# [1] 7
```

Форма записи чисел 1e7 $= 1 \cdot 10^7$ — это так называемая экспоненциальная запись (scientific notation).

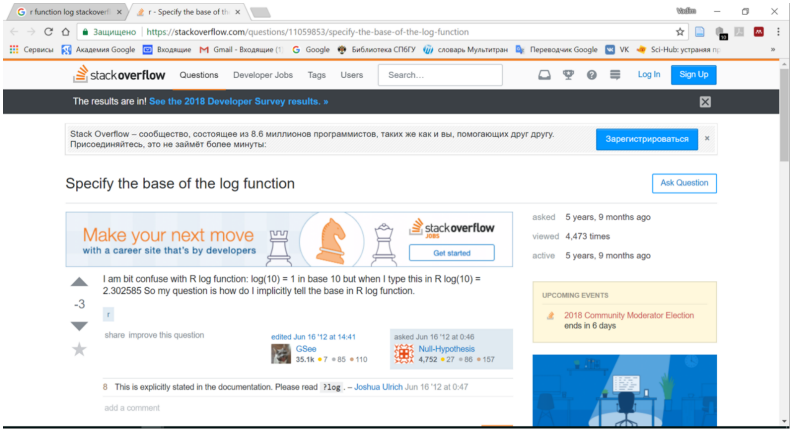
# Google — ваш лучший друг



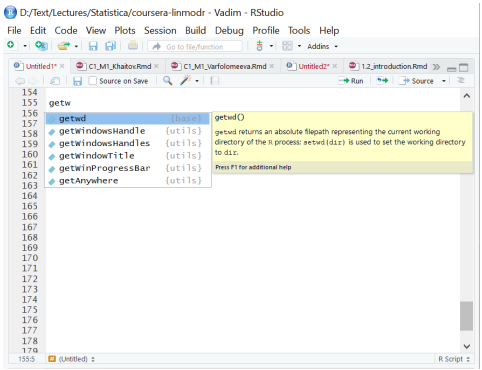
# Сообщество Stack Overflow



# Сообщество Stack Overflow



# RStudio может подсказывать и сама




Начните набирать название функции и нажмите `Tab` или `Ctrl + Space`.

# Не бойтесь сообщений об ошибках и предупреждений

Разобраться в них обычно удается даже при минимальном знании английского языка.

```
sqr(4)
```

```
Error in sqr(4) : could not find function ”sqr”
```

```
sqrt(-1)
```

```
[1] NaN
Warning message:
In sqrt(-1) : NaNs produced
```

# Что делать, если консоль неистово плюсует

Знак `+` в консоли значит, что во введенной команде чего-то не хватает, и `R` ожидает продолжения.

Один из самых частых случаев — потерялась закрывающая скобка.

```
> sqrt(4
+
+
+
+
```

Для выхода из этого положения нажмите `Esc`.

# Установка пакетов

В.М.Хайтов, к.б.н.

Биологический факультет, СПбГУ

# Что такое пакеты

Пакет - набор функций, предназначенных для решения определенного класса задач.

Примеры пакетов:

- `ggplot2` — набор функций для рисования графиков
- `dplyr`, `tidyr` — средства управления данными и их преобразования в удобную форму
- `car`, `MASS`, `lmer` — инструменты регрессионного анализа

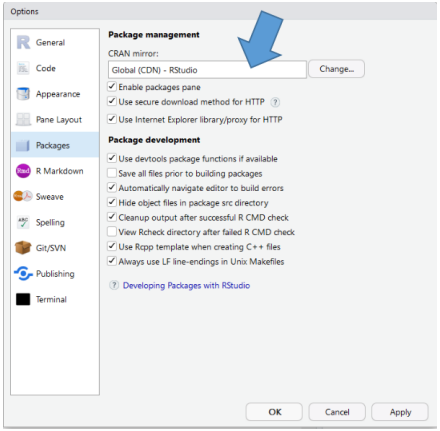
# Установка пакетов

Установка пакетов происходит из Сети. Лучше работать, имея постоянное подключение.

Главный репозиторий находится на сайте `CRAN` (https://cran.r-project.org/).

Если вы работаете под Windows, то запускайте `RStudio` от имени администратора.

# Выбор зеркала




Из множества зеркал `CRAN` лучше выбрать ближайшее к вам.

Tools → Global options → Packages → CRAN mirror

# Установка пакетов в R

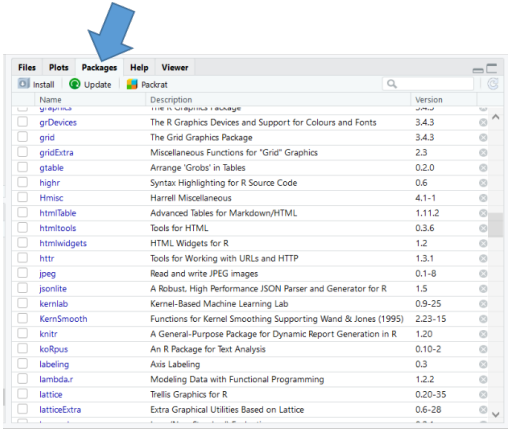
```
install.packages('ggplot2') # Устанавливаем пакет ggplot2
install.packages('tidyr')   # Устанавливаем пакет tidyr
```

Пакеты достаточно один раз установить в локальную библиотеку.

Команда `library()` активирует нужный пакет, если он уже установлен, и выполняется один раз за сеанс работы в R.

```
library(ggplot2) # Активируем пакет ggplot2
```

# Список установленных пакетов



# R как калькулятор. Математические операции

М.А. Варфоломеева, PhD

Биологический факультет, СПбГУ

# R как калькулятор


https://www.shutterstock.com/ru/image-photo/crazy-super-hero-calculator-588734510

R может использоваться для очень сложных задач, но мы начнем с самых простых.

# Организация кода

Строка — основная единица кода.
В `RStudio` код активной строки выполняется целиком после нажатия `Ctrl + Enter`.

Самое главное в коде — это комментарии (обозначаются символом #).
Вы пишете код не для себя, а для себя в будущем.
Лучше, если комментарий отвечает на вопрос: **для чего нужен этот код?**

```
# Это пример использования комментариев в коде

2 + 2 # Все, что написано после значка '#'--- это комментарий
```

```
# [1] 4
```

# Простейшие арифметические действия

```
2 + 2          # Сложение
```

```
# [1] 4
```

```
4 - 1          # Вычитание
```

```
# [1] 3
```

```
3 * 9          # Умножение
```

```
# [1] 27
```

```
1024 / 2       # Деление
```

```
# [1] 512
```

# Степени, корни

```
2 ^ 4          # Возведение в степень
```

```
# [1] 16
```

```
sqrt(9)      # Квадратный корень
```

```
# [1] 3
```

```
27 ^ (1 / 3)  # Кубический корень
```

```
# [1] 3
```

# Логарифмы

По-умолчанию в логарифмах используется основание e.

```
log(100)
```

```
# [1] 4.60517
```

```
log(100, base = 10)
```

```
# [1] 2
```

# Иррациональные константы

```
exp(1)
```

```
# [1] 2.718282
```

```
pi
```

```
# [1] 3.141593
```

# Старшинство математических операторов

| Приоритет | Оператор | Действие |
|---|---|---|
| 1 | ^ | Возведение в степень |
| 2 | - | Отрицательное число |
| 3 | %%<br>%/% | Остаток от деления<br>Деление нацело |
| 4 | *<br>/ | Умножение<br>Деление |
| 5 | +<br>- | Сложение<br>Вычитание |

Порядок действий можно менять при помощи скобок.

```
27 ^ 1 / 3
```

```
# [1] 9
```

```
27 ^ (1 / 3)
```

```
# [1] 3
```

# Внимательно следите за расстановкой скобок

```
1 + 2 * 3 - 1 ^ -1
```

```
# [1] 6
```

```
1 + 2 * (3 - 1) ^ -1
```

```
# [1] 2
```

```
1 + (2 * (3 - 1)) ^ -1
```

```
# [1] 1.25
```

# Переменные

М.А. Варфоломеева, PhD

Биологический факультет, СПбГУ

# Переменные


https://www.shutterstock.com/ru/image-photo/vintage-gift-box-package-address-card-92183338

Переменные — это “коробки”, в которые мы складываем результаты работы того или иного кода.

# Переменные можно создать с помощью оператора присваивания <-

```
имя_переменной <- содержимое
```

```
# Создаем переменную и записываем в нее значение

box_weight <- 1.3 # вес коробки
apples <- 7 # число яблок
```

# Чтобы увидеть содержимое переменной, нужно вызвать ее по имени

Содержимое переменной будет распечатано в консоли.

```
apples
```

```
# [1] 7
```

```
box_weight
```

```
# [1] 1.3
```

# Имена переменных должны быть “говорящими”

В именах можно использовать латинские буквы, цифры, нижнее подчеркивание и точку.
В R регистр имеет значение: name и Name — это разные имена.

| Имя переменной | Качество |
|---|---|
| #_1, 1_x, eye colour | недопустимо |
| mean, sqrt, df | плохо, т.к. эти слова уже заняты в R |
| a, var1, var_1, MyData | не информативно |
| eyecolour, errorcount, distancetowork | лучше, но трудно читать |
| eye_colour, error.count, DistanceToWork | хорошо: информативно, легко читать |

# Стиль оформления кода

Лучше придерживаться единого стиля оформления кода:

- Имена переменных
- Комментарии
- Отступы
- Пробелы

Пример описания стиля:

http://adv-r.had.co.nz/Style.html

# Код — это линейная последовательность действий

https://www.shutterstock.com/ru/image-photo/silhouette-time-lapse-sequence-boy-leaping-472802506

Отдельные инструкции в коде зависят друг от друга. Старайтесь размещать их в логичном порядке.

Важное следствие: к переменным можно обращаться только после того как они были созданы.

# Код — это линейная последовательность действий

https://www.shutterstock.com/ru/image-photo/silhouette-time-lapse-sequence-boy-leaping-472802506

Отдельные инструкции в коде зависят друг от друга. Старайтесь размещать их в логичном порядке.

Важное следствие: к переменным можно обращаться только после того как они были созданы.

Этот код вызовет ошибку:

```
my_variable
```

```
Error: object 'my_variable' not found
```

```
my_variable <- 10
```

Этот код работает:

```
another_variable <- 10
another_variable
```

```
# [1] 10
```

# Переменные можно использовать в расчетах

Радиус апельсина 4.7 см, а толщина кожуры — 1 см. Чего больше в этом апельсине, кожуры или съедобной мякоти?

Объем шара: $V = \frac{4}{3}\pi R^3$

https://www.shutterstock.com/ru/image-photo/ripe-orange-isolated-on-white-background-100359398

# Переменные можно использовать в расчетах

Радиус апельсина 4.7 см, а толщина кожуры — 1 см. Чего больше в этом апельсине, кожуры или съедобной мякоти?

Объем шара: $V = \frac{4}{3}\pi R^3$

https://www.shutterstock.com/ru/image-photo/ripe-orange-isolated-on-white-background-100359398

```
R <- 4.7
r <- R - 1
# весь апельсин
whole_orange <- 4 / 3 * pi * R ^ 3
# объем съедобной части
edible_flesh <- 4 / 3 * pi * r ^ 3
edible_flesh
```

```
# [1] 212.1748
```

```
# объем кожуры
peel <- whole_orange - edible_flesh
peel
```

```
# [1] 222.718
```

# Первые шаги в R

М.А. Варфоломеева, PhD

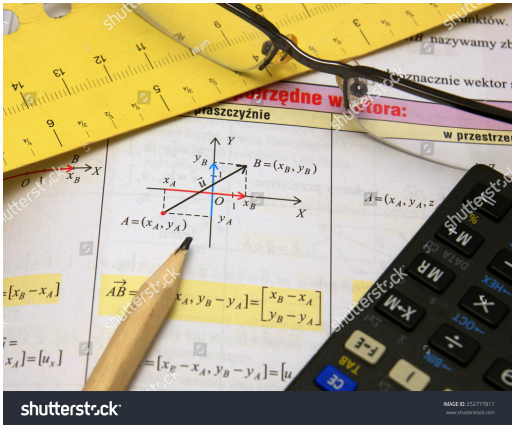
Биологический факультет, СПбГУ

# Векторы и операции с ними

В.М. Хайтов, к.б.н.

Биологический факультет, СПбГУ

# Вектор




https://www.shutterstock.com/ru/image-photo/graphs-charts-vectors-vector-theory-together-252777811

Вектор — это не только направленный отрезок.

В математике и программировании **вектор** - это **одномерный** набор **однотипных** элементов.

Вектор — это базовая структура хранения данных в среде R

# Вектор — одномерный набор элементов.


https://www.shutterstock.com/ru/image-photo/honolulu-hawaii-usa-jan-11-2016-361145654

У каждого элемента есть своя позиция в этой последовательности.

Эта позиция в среде `R` обозначается `[]`

Например

```
  [1]   1   2   3   4   5   6   7   8   9  10  11
 [12]  12  13  14  15  16  17  18  19  20  21  22
 [23]  23  24  25  26  27  28  29  30  31  32  33
 [34]  34  35  36  37  38  39  40  41  42  43  44
 [45]  45  46  47  48  49  50  51  52  53  54  55
 [56]  56  57  58  59  60  61  62  63  64  65  66
 [67]  67  68  69  70  71  72  73  74  75  76  77
 [78]  78  79  80  81  82  83  84  85  86  87  88
 [89]  89  90  91  92  93  94  95  96  97  98  99
[100] 100
```

# Числа и векторы

До сих пор мы рассматривали действия, произведенные над каким-то одним числом.

Одно число — это частный случай вектора: вектор, состоящий из одного элемента.

С векторами, как и с отдельными числами, можно производить операции
как с единым целым.

# Создание векторов

Основные способы создания векторов в R

- Прямое перечисление элементов
- Задание последовательности
- Загрузка данных из внешних источников

# Прямое перечисление элементов с помощью функции `c()`

```
c(1, 0, 3, -10)
```

```
# [1]   1   0   3 -10
```

# Прямое перечисление элементов с помощью функции `c()`

```
c(1, 0, 3, -10)
```

```
# [1]   1   0   3 -10
```

Из-за того, что английская “си” и русская “эс” находятся на одной клавише, часто возникает ошибка

```
c(1, 6, 3, -11)
```

```
Error in c(1, 6, 3, -11) : could not find function ”c”
```

# Векторы и перменные

До сих пор мы рассматривали переменные, как “коробки”, в которых лежат какие-то числа.

В переменную можно положить и вектор.

```
my_vector <- c(1, 0, 3, -10)
my_vector
```

```
# [1]   1   0   3 -10
```

# Создание вектора при помощи оператора ":"

Оператор : позволяет создать последовательность с шагом 1.

```
my_vector <- 1:10
my_vector
```

```
#  [1]  1  2  3  4  5  6  7  8  9 10
```

Можно создать последовательность в любом направлении

```
my_vector <- 0:-10
my_vector
```

```
#  [1]   0  -1  -2  -3  -4  -5  -6  -7  -8  -9 -10
```

# Векторы с повторяющимися элементами, можно создать с помощью функции `rep()`

```
my_vector_2 <- rep(1, 4)
my_vector_2
```

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

# Функция `rep()` позволяет повторить вектор или его элемент

Можно повторять сразу целый вектор

```
my_vector_3 <- rep(1:5, 4)
my_vector_3
```

```
#  [1] 1 2 3 4 5 1 2 3 4 5 1 2 3 4 5 1 2 3 4 5
```

Можно многократно повторять каждый из элементов вектора

```
my_vector_4 <- rep(1:5, each = 4)
my_vector_4
```

```
#  [1] 1 1 1 1 2 2 2 2 3 3 3 3 4 4 4 4 5 5 5 5
```

# Задать последовательность можно с помощью функции `seq()`

```
my_vector_5 <- seq(from = 1, to = 10, by = 1)
my_vector_5
```

```
#  [1]  1  2  3  4  5  6  7  8  9 10
```

```
my_vector_6 <- seq(from = 1, to = 10, by = 0.5)
my_vector_6
```

```
#  [1]  1.0  1.5  2.0  2.5  3.0  3.5  4.0  4.5  5.0  5.5  6.0  6.5  7.0  7.5  8.0  8.5  9.0
# [18]  9.5 10.0
```

```
my_vector_7 <- seq(from = 1, to = 10, length.out = 20)
my_vector_7
```

```
#  [1]  1.000000  1.473684  1.947368  2.421053  2.894737  3.368421  3.842105  4.315789
#  [9]  4.789474  5.263158  5.736842  6.210526  6.684211  7.157895  7.631579  8.105263
# [17]  8.578947  9.052632  9.526316 10.000000
```

# Операции с векторами

В.М. Хайтов, к.б.н.

Биологический факультет, СПбГУ

# Операции с векторами


https://www.shutterstock.com/image-photo/background-numbers-zero-nine-519010999

В R многие операции векторизованы — это значит, что одно и то же действие очень легко повторить для множества элементов.

Кроме этого, при помощи некоторых функций можно получать информацию о свойствах вектора в целом.

# Прибавляем ко всем элементам одно и то же число

```
my_vector_8 <- seq(1, 10, 1)
my_vector_8
```

```
#  [1]  1  2  3  4  5  6  7  8  9 10
```

```
my_vector_8 + 5
```

```
#  [1]  6  7  8  9 10 11 12 13 14 15
```

# Вычитание числа

```
my_vector_8 - 1
```

```
#  [1] 0 1 2 3 4 5 6 7 8 9
```

Но...

```
1 - my_vector_8
```

```
#  [1]  0 -1 -2 -3 -4 -5 -6 -7 -8 -9
```

# Умножаем все элементы вектора на одно число

```
my_vector_8 * 10
```

```
#  [1]  10  20  30  40  50  60  70  80  90 100
```

# Деление

```
my_vector_8 / 10
```

```
#  [1] 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1.0
```

Но...

```
1 / my_vector_8
```

```
#  [1] 1.0000000 0.5000000 0.3333333 0.2500000 0.2000000 0.1666667 0.1428571 0.1250000
#  [9] 0.1111111 0.1000000
```

# Корни и возведение в степень

```
sqrt(my_vector_8)
```

```
#  [1] 1.000000 1.414214 1.732051 2.000000 2.236068 2.449490 2.645751 2.828427 3.000000
# [10] 3.162278
```

```
my_vector_8^2
```

```
#  [1]   1   4   9  16  25  36  49  64  81 100
```

Но...

```
2 ^ my_vector_8
```

```
#  [1]    2    4    8   16   32   64  128  256  512 1024
```

# Количество элементов в векторе

```
my_vector_9 <- c(rep(1:5, each = 10), 10:20, c(1, 5, 8), my_vector_4)
# С помощью функции c() можно объединить разные векторы в один вектор.


my_vector_9
```

```
#  [1]  1  1  1  1  1  1  1  1  1  1  2  2  2  2  2  2  2  2  2  3  3  3  3  3  3  3  3
# [29]  3  3  4  4  4  4  4  4  4  4  4  4  5  5  5  5  5  5  5  5  5  5 10 11 12 13 14 15
# [57] 16 17 18 19 20  1  5  8  1  1  1  1  2  2  2  2  3  3  3  3  4  4  4  4  5  5  5  5
```

```
length(my_vector_9)
```

```
# [1] 84
```

# Сумма элементов вектора

```
sum(my_vector_9)
```

```
# [1] 389
```

# Уникальные значения вектора

```
unique(my_vector_9)
```

```
#  [1]  1  2  3  4  5 10 11 12 13 14 15 16 17 18 19 20  8
```

# Частоты уникальных значений вектора

```
table(my_vector_9)

# my_vector_9
#  1  2  3  4  5  8 10 11 12 13 14 15 16 17 18 19 20
# 15 14 14 14 15  1  1  1  1  1  1  1  1  1  1  1  1
```

# Минимальное и максимальное значение вектора

```
min(my_vector_9)
```

```
# [1] 1
```

```
max(my_vector_9)
```

```
# [1] 20
```

# Среднее арифметическое значений вектора

```
mean(my_vector_9)
```

```
# [1] 4.630952
```

# Это еще не все...

Мы рассмотрели лишь самую вершину векторного айсберга.

Существует большой раздел математики, работающий с векторами (и матрицами) — линейная алгебра.

Но о ней мы кратко поговорим в следующем курсе.

# Обзор типов данных

М.А. Варфоломеева, PhD

Биологический факультет, СПбГУ

# Статистика имеет дело с данными разных типов


https://www.shutterstock.com/image-vector/vector-illustration-set-coffee-tea-cups-211563346

Характер данных зависит как от объекта исследования, так и от методов, которыми пользуется исследователь.

Для разных типов данных нужен разный аппарат статистических методов и разные способы представления.

# Мерные (интервальные) данные: непрерывные

Могут принимать любые значения.


Пример — температура чая в чашке.

# Мерные (интервальные) данные: непрерывные

Могут принимать любые значения.


Пример — температура чая в чашке.

```
temperature <- c(53, 79.5, 61, 70)
temperature
```

```
# [1] 53.0 79.5 61.0 70.0
```

# Мерные (интервальные) данные: счетные


Принимают только дискретные значения.
Обычно это подсчет чего-нибудь.

Пример — число кусков сахара в чашке.

# Мерные (интервальные) данные: счетные


Принимают только дискретные значения.
Обычно это подсчет чего-нибудь.

Пример — число кусков сахара в чашке.

```
sugar <- c(0, 2, 0, 3)
sugar
```

```
# [1] 0 2 0 3
```

# Категориальные данные с двумя категориями

0 1 1 1


Это информация о наличии/отсутствии одного признака.

Бинарные данные можно представить в виде набора из двух значений.

Пример — черный ли в чашке чай?
Черный — 1 или TRUE, зеленый — 0 или FALSE

# Категориальные данные с двумя категориями


Это информация о наличии/отсутствии одного признака.

Бинарные данные можно представить в виде набора из двух значений.

Пример — черный ли в чашке чай?
Черный — 1 или TRUE, зеленый — 0 или FALSE

```
black_numeric <- c(0, 1, 1, 1)
black_numeric
```

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

```
black_logical <- c(FALSE, TRUE, TRUE, TRUE)
black_logical
```

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

```
black_txt <- c('зеленый', 'черный', 'черный', 'черный')
black_txt
```

```
# [1] "зеленый" "черный"  "черный"  "черный"
```

# Категориальные данные со множеством категорий


Часто это просто слова, описывающие свойства объекта.

Пример — тип напитка в чашке.

# Категориальные данные со множеством категорий


Часто это просто слова, описывающие свойства объекта.

Пример — тип напитка в чашке.

```
drink <- c('чай', 'вода', 'кофе', 'чай')
drink
```

```
# [1] "чай"  "вода" "кофе" "чай"
```

# Категориальные данные с любым числом категорий часто представляют в виде факторов

Внутри фактора значения категорий кодируются числами.
Каждому значению сопоставляется своя текстовая “этикетка” (уровень).

`'чай', 'вода', 'кофе', 'чай'`

Таблица перекодировки:

| Значение | Уровень (level) |
|---|---|
| 1 | `'вода'` |
| 2 | `'кофе'` |
| 3 | `'чай'` |

# Категориальные данные с любым числом категорий часто представляют в виде факторов

Внутри фактора значения категорий кодируются числами.
Каждому значению сопоставляется своя текстовая “этикетка” (уровень).

`'чай', 'вода', 'кофе', 'чай'`

Таблица перекодировки:

| Значение | Уровень (level) |
|---|---|
| 1 | 'вода' |
| 2 | 'кофе' |
| 3 | 'чай' |

```
drink
```

```
# [1] "чай"  "вода" "кофе" "чай"
```

```
drink_fact <- factor(drink) # преобразуем в фактор

drink_fact # нам показывают уровни
```

```
# [1] чай  вода кофе чай
# Levels: вода кофе чай
```

```
as.numeric(drink_fact) # "внутри" фактора записаны числа
```

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

# Ранговые данные


Нечто среднее между мерными и категориальными: дискретные значения упорядочены.

Шаг между соседними рангами не обязательно одинаковый.

Пример — субъективная оценка степени заварки чая: I = так себе, II = средний, III = крепкий

# Ранговые данные


Нечто среднее между мерными и категориальными: дискретные значения упорядочены.

Шаг между соседними рангами не обязательно одинаковый.

Пример — субъективная оценка степени заварки чая: I = так себе, II = средний, III = крепкий

Можно закодировать это в виде текста,
но чтобы указать порядок, нужно превратить его в фактор.

```
strength <- c('так себе', 'средний', 'крепкий', 'средний')
strength
```

```
# [1] "так себе" "средний"  "крепкий"  "средний"
```

# Ранжированные данные лучше хранить в виде упорядоченных факторов

```
strength
```

```
# [1] ”так себе” ”средний”  ”крепкий”  ”средний”
```

```
# создадим упорядоченный фактор
strength_ord <- factor(strength,
                       levels = c('так себе', 'средний', 'крепкий'),
                       ordered = TRUE)
strength_ord             # нам показывают уровни и их порядок
```

```
# [1] так себе средний  крепкий  средний
# Levels: так себе < средний < крепкий
```

```
as.numeric(strength_ord) # ”внутри” фактора записаны числа
```

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

# Нужно уметь различать нулевые и отсутствующие значения


Ноль означает, что что-то было измерено и значение измерения равно нулю.

# Нужно уметь различать нулевые и отсутствующие значения


Ноль означает, что что-то было измерено и значение измерения равно нулю.

```
sugar
```

```
# [1] 0 2 0 3
```

# Нужно уметь различать нулевые и отсутствующие значения


Ноль означает, что что-то было измерено и значение измерения равно нулю.

```
sugar
```

```
# [1] 0 2 0 3
```

Отсутствующее значение или пропуск означает, что объект был,
но измерение не было сделано.
Это обозначается `NA` (“not available”).

# Нужно уметь различать нулевые и отсутствующие значения


Ноль означает, что что-то было измерено и значение измерения равно нулю.

```
sugar
```

```
# [1] 0 2 0 3
```

Отсутствующее значение или пропуск означает, что объект был,
но измерение не было сделано.
Это обозначается `NA` (“not available”).

```
sugar_1 <- c(0, NA, 0, 3)
sugar_1
```

```
# [1]  0 NA  0  3
```

# Логические данные

М.А. Варфоломеева, PhD

Биологический факультет, СПбГУ

# Логические данные


https://www.shutterstock.com/ru/image-photo/true-false-blackboards-on-wood-background-261530159

Логические данные широко используются для проверки выполнения условий и фильтрации данных по этим условиям.

В этом разделе мы разберем основные операции с логическими данными.

# Обозначение логических данных в R

Для обозначения логических данных в R зарезервированы два слова

```
c(TRUE, FALSE)
```

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

и два сокращения (лучше их не использовать, т.к. код хуже читается).

```
c(T, F)
```

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

# Для проверки условий используются операторы сравнения

| Оператор | Действие | Оператор | Действие |
|---|---|---|---|
| == | равно | != | не равно |
| > | больше | >= | больше или равно |
| < | меньше | <= | меньше или равно |

Кроме операторов сравнения есть функции, названия которых начинаются на `is` (например, `is.numeric()`, `is.na()`), оператор сопоставления `%in%` и т.п.

```
?Comparison # справка об операторах сравнения
?match      # о сопоставлении
```

# Проверяем выполнение условий для числовых векторов

```
sugar # этот вектор мы создали раньше
```

```
# [1] 0 2 0 3
```

В каких чашках ровно три куска сахара?

```
sugar == 2
```

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

В каких чашках 0 либо 3 куска сахара?

```
sugar %in% c(0, 3)
```

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

В каких чашках больше двух кусков сахара?

```
sweet <- sugar > 2
sweet
```

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

# Проверяем выполнение условий для текстовых векторов

```
strength # этот вектор мы создали раньше
```

```
# [1] ”так себе” ”средний”  ”крепкий”  ”средний”
```

### В каких чашках крепкий чай?

```
strength == 'крепкий'
```

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

### В каких чашках средний или крепкий чай?

```
strength %in% c('крепкий', 'средний')
```

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

### В каких чашках некрепкий чай?

```
mild <- strength != 'крепкий'
mild
```

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

# Логические операторы используются для объединения результатов нескольких проверок

**& — И**

| Действие | Результат |
|---|---|
| TRUE & TRUE | TRUE |
| TRUE & FALSE | FALSE |
| FALSE & FALSE | FALSE |

**| — ИЛИ**

| Действие | Результат |
|---|---|
| TRUE \| TRUE | TRUE |
| TRUE \| FALSE | TRUE |
| FALSE \| FALSE | FALSE |

**! — отрицание**

| Действие | Результат |
|---|---|
| ! TRUE | FALSE |
| ! FALSE | TRUE |

```
?Logic # Справка о логических операторах
```

# Можно проверить несколько условий сразу

В каких чашках некрепкий чай **И** больше двух кусков сахара?

Можно решить в два действия...

```
mild <- strength != 'крепкий'
mild
```

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

```
sweet <- sugar > 2
sweet
```

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

```
mild_and_sweet <- mild & sweet
mild_and_sweet
```

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

# Можно проверить несколько условий сразу

В каких чашках некрепкий чай **И** больше двух кусков сахара?

Можно решить в два действия...

```
mild <- strength != 'крепкий'
mild
```

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

```
sweet <- sugar > 2
sweet
```

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

```
mild_and_sweet <- mild & sweet
mild_and_sweet
```

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

или в одно действие.

```
mild_and_sweet <- strength != 'крепкий' & sugar > 2
mild_and_sweet
```

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

# Логические векторы умеют превращаться в числовые из 0 и 1

```
mild <- strength != 'крепкий'
mild
```

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

```
as.numeric(mild)
```

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

# Логические векторы умеют превращаться в числовые из 0 и 1

```
mild <- strength != 'крепкий'
mild
```

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

```
as.numeric(mild)
```

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

Количество значений `TRUE` —
это сумма элементов логического вектора.

```
sum(mild)
```

```
# [1] 3
```

# Логические векторы умеют превращаться в числовые из 0 и 1

```
mild <- strength != 'крепкий'
mild
```

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

```
as.numeric(mild)
```

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

Количество значений `TRUE` —
это сумма элементов логического вектора.

```
sum(mild)
```

```
# [1] 3
```

Доля значений `TRUE` среди всех значений —
это среднее значение логического вектора.

```
mean(mild)
```

```
# [1] 0.75
```

# Поиск элементов вектора по номеру

М.А. Варфоломеева, PhD

Биологический факультет, СПбГУ

# Поиск элементов вектора по номеру


https://www.shutterstock.com/ru/image-photo/collection-house-numbers-one-twentyfive-176481500

Для анализа часто бывает нужно выбрать лишь часть данных или исключить определенные элементы.

# Оператор [ ] извлекает из вектора элементы

```
имя_вектора[адрес]
```

В качестве `адреса` может использоваться

- порядковый номер элемента или вектор с номерами (обсудим в этом разделе),
- логический вектор (обсудим в следующем разделе).

# Оператор [ ] извлекает из вектора элементы

```
имя_вектора[адрес]
```

В качестве `адреса` может использоваться

- порядковый номер элемента или вектор с номерами (обсудим в этом разделе),
- логический вектор (обсудим в следующем разделе).

Потренируемся извлекать кусочки из вектора аббревиатур названий месяцев.

```
month.abb
```

```
#  [1] "Jan" "Feb" "Mar" "Apr" "May" "Jun" "Jul" "Aug" "Sep" "Oct" "Nov" "Dec"
```

```
month.abb[1] # элемент с номером 1
```

```
# [1] "Jan"
```

# Пробуем извлечь несколько элементов из вектора

Самый очевидный код — простое перечисление элементов — вызовет ошибку.

```
month.abb[11, 12, 1] # этот код вызовет ошибку
```

```
Error in month.abb[11, 12, 1] : incorrect number of dimensions
```

Дело в том, что вектор — одномерный объект и у его элементов всего один адрес.
Нужно использовать **вектор** из номеров.

# Для извлечения нескольких элементов используем вектор из их номеров

Попробуем извлечь только зимние месяцы в правильном порядке.

Можно создать вектор номеров заранее...

```
# Индексы созданы заранее
id_winter <- c(11, 12, 1)
month.abb[id_winter]
```

```
# [1] "Nov" "Dec" "Jan"
```

# Для извлечения нескольких элементов используем вектор из их номеров

Попробуем извлечь только зимние месяцы в правильном порядке.

Можно создать вектор номеров заранее...

```
# Индексы созданы заранее
id_winter <- c(11, 12, 1)
month.abb[id_winter]
```

```
# [1] "Nov" "Dec" "Jan"
```

или использовать их сразу же внутри `[ ]`.

```
# Индексы используются сразу
month.abb[c(11, 12, 1)]
```

```
# [1] "Nov" "Dec" "Jan"
```

# Для извлечения нескольких элементов используем вектор из их номеров

Попробуем извлечь только зимние месяцы в правильном порядке.

Можно создать вектор номеров заранее...

```
# Индексы созданы заранее
id_winter <- c(11, 12, 1)
month.abb[id_winter]
```

```
# [1] "Nov" "Dec" "Jan"
```

или использовать их сразу же внутри `[ ]`.

```
# Индексы используются сразу
month.abb[c(11, 12, 1)]
```

```
# [1] "Nov" "Dec" "Jan"
```

Несколько элементов подряд извлечь очень легко.

```
month.abb[1:3] # Несколько элементов подряд
```

```
# [1] "Jan" "Feb" "Mar"
```

# К чему приведет обращение к несущестующему элементу?

При попытке обратиться к номеру элемента, которого нет в векторе, получится `NA`.

```
month.abb[13] # в этом векторе нет 13-го элемента
```

```
# [1] NA
```

# Исключение элементов

Знак минус перед номером элемента или группой номеров элементов приводит к их исключению.

Год без января.

```
month.abb[-1] # удаление элемента
```

```
#  [1] "Feb" "Mar" "Apr" "May" "Jun" "Jul" "Aug" "Sep" "Oct" "Nov" "Dec"
```

Год без зимних месяцев.

```
month.abb[-c(1, 11, 12)]  # удаление нескольких элементов
```

```
# [1] "Feb" "Mar" "Apr" "May" "Jun" "Jul" "Aug" "Sep" "Oct"
```

# Поиск элементов вектора по условию

М.А. Варфоломеева, PhD

Биологический факультет, СПбГУ

# Поиск элементов вектора по условию


https://www.shutterstock.com/ru/image-photo/true-false-question-vintage-letterpress-wood-373414852

Поиск по условию — задача, которая часто возникает, когда нужно оставить в анализе только часть данных.

# Месяцы и времена года

Вектор названий месяцев `month.abb` у нас уже есть.

```
month.abb
```

```
#  [1] ”Jan” ”Feb” ”Mar” ”Apr” ”May” ”Jun” ”Jul” ”Aug” ”Sep” ”Oct” ”Nov” ”Dec”
```

Создадим для экспериментов вектор с названиями времен года.

```
season <- c('Winter', 'Winter', rep(c('Spring', 'Summer', 'Autumn'), each = 3), 'Winter' )
season
```

```
#  [1] ”Winter” ”Winter” ”Spring” ”Spring” ”Spring” ”Summer” ”Summer” ”Summer” ”Autumn”
# [10] ”Autumn” ”Autumn” ”Winter”
```

# Сопряженные векторы

Векторы `season` и `month.abb` одинаковой длины, а их элементы соответствуют друг другу. Такие сопряженные векторы можно использовать для перекрестного поиска значений.

```
cbind(season, month.abb)
```

```
#       season   month.abb
#  [1,] ”Winter” ”Jan”
#  [2,] ”Winter” ”Feb”
#  [3,] ”Spring” ”Mar”
#  [4,] ”Spring” ”Apr”
#  [5,] ”Spring” ”May”
#  [6,] ”Summer” ”Jun”
#  [7,] ”Summer” ”Jul”
#  [8,] ”Summer” ”Aug”
#  [9,] ”Autumn” ”Sep”
# [10,] ”Autumn” ”Oct”
# [11,] ”Autumn” ”Nov”
# [12,] ”Winter” ”Dec”
```

# Ищем элементы вектора по условию

`вектор[логический_вектор]`

Эта команда отберет только элементы `вектора`, где в `логическом_векторе` стоит `TRUE`.

Например, чтобы найти весенние месяцы, нужно определить, какие элементы в векторе `season` будут равны `'Spring'`.

```
season == 'Spring'
```

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

Этот логический вектор можно использовать для отбора значений.

```
month.abb[season == 'Spring']
```

```
# [1] ”Mar” ”Apr” ”May”
```

# Сложные варианты индексации

Условие может быть каким угодно, главное, чтобы сопряженный логический вектор имел столько же элементов, что и исходный вектор.

```
month.abb [season != 'Spring']
```

```
# [1] ”Jan” ”Feb” ”Jun” ”Jul” ”Aug” ”Sep” ”Oct” ”Nov” ”Dec”
```

Или

```
month.abb [season %in% c('Spring', 'Winter')]
```

```
# [1] ”Jan” ”Feb” ”Mar” ”Apr” ”May” ”Dec”
```

# Выбираем четные и нечетные элементы вектора

```r
# Вектор, в котором 1 - нечетное число, 0 - нечетное
1:length(month.abb) %% 2
```

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

```r
# Логический вектор где TRUE для нечетных элементов
is_odd <- as.logical(1:length(month.abb) %% 2)
is_odd
```

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

# Выбираем четные и нечетные элементы вектора

```r
# Вектор, в котором 1 - нечетное число, 0 - нечетное
1:length(month.abb) %% 2
```

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

```r
# Логический вектор где TRUE для нечетных элементов
is_odd <- as.logical(1:length(month.abb) %% 2)
is_odd
```

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

```r
month.abb[is_odd] # Нечетные элементы
```

```
# [1] "Jan" "Mar" "May" "Jul" "Sep" "Nov"
```

```r
month.abb[! is_odd] # Четные элементы
```

```
# [1] "Feb" "Apr" "Jun" "Aug" "Oct" "Dec"
```

# Датафреймы и операции с ними

М.А. Варфоломеева, PhD

Биологический факультет, СПбГУ

# Датафрейм

| | name | height | mass | hair_color |
|---|---|---|---|---|
| 1 | Luke Skywalker | 172 | 77 | blond |
| 2 | C-3PO | 167 | 75 | *NA* |
| 3 | R2-D2 | 96 | 32 | *NA* |
| 4 | Darth Vader | 202 | 136 | none |
| 5 | Leia Organa | 150 | 49 | brown |
| 6 | Chewbacca | 228 | 112 | brown |
| 7 | Han Solo | 180 | 80 | brown |
| 8 | Yoda | 66 | 17 | white |
| 9 | Shmi Skywalker | 163 | *NA* | black |

Датафрейм — это таблица данных (аналог листа в Excel или Calc).

- Столбцы — переменные
- Строки — наблюдения

# Данные о персонажах “Звездных войн”


https://www.shutterstock.com/ru/image-photo/bologna-italy-march-1-2015-star-257048308

Работу с датафреймами мы разберем на примере фрагмента данных о персонажах “Звездных войн” из базы SWAPI (https://swapi.co/).

Скачайте файл `tiny_starwars.RData` и положите его в папку `data` внутри вашей рабочей папки с материалами курса (см. видео “Организация рабочего пространства”).

# Откроем данные из файла `tiny_starwars.RData`

```
load(file = 'data/tiny_starwars.RData')
```

Если все было сделано правильно, то в вашем рабочем пространстве появится переменная `SW`.

```
SW # вывод всего датафрейма в консоль
```

```
#             name height mass hair_color gender        species n_films
# 1 Luke Skywalker    172   77      blond   male          Human       5
# 2          C-3PO    167   75       <NA>   <NA>          Droid       6
# 3          R2-D2     96   32       <NA>   <NA>          Droid       7
# 4    Darth Vader    202  136       none   male          Human       4
# 5    Leia Organa    150   49      brown female          Human       5
# 6      Chewbacca    228  112      brown   male        Wookiee       5
# 7       Han Solo    180   80      brown   male          Human       4
# 8           Yoda     66   17      white   male Yoda's species       5
# 9 Shmi Skywalker    163   NA      black female          Human       2
```

# Как посмотреть, что находится в датафрейме?

Для больших датафреймов удобнее отображать лишь часть строк.

```
head(SW)      # первые несколько строк
```

```
#             name height mass hair_color gender species n_films
# 1 Luke Skywalker    172   77      blond   male   Human       5
# 2          C-3PO    167   75       <NA>   <NA>   Droid       6
# 3          R2-D2     96   32       <NA>   <NA>   Droid       7
# 4    Darth Vader    202  136       none   male   Human       4
# 5    Leia Organa    150   49      brown female   Human       5
# 6      Chewbacca    228  112      brown   male Wookiee       5
```

```
tail(SW, n = 2) # последние несколько строк
```

```
#             name height mass hair_color gender        species n_films
# 8           Yoda     66   17      white   male Yoda's species       5
# 9 Shmi Skywalker    163   NA      black female          Human       2
```

# Сколько строк и столбцов, как называются переменные?

```
nrow(SW)  # Количество строк
```

```
# [1] 9
```

```
ncol(SW)  # Количество столбцов
```

```
# [1] 7
```

```
colnames(SW) # Имена переменных в датафрейме
```

```
# [1] ”name”       ”height”     ”mass”       ”hair_color” ”gender”     ”species”
# [7] ”n_films”
```

# Подробная информация о структуре датафрейма

При помощи функции `str()` можно посмотреть сразу на имена переменных, их тип данных и примеры значений.

```
str(SW)
```

```
# 'data.frame': 9 obs. of  7 variables:
#  $ name      : chr  ”Luke Skywalker” ”C-3PO” ”R2-D2” ”Darth Vader” ...
#  $ height    : int  172 167 96 202 150 228 180 66 163
#  $ mass      : num  77 75 32 136 49 112 80 17 NA
#  $ hair_color: chr  ”blond” NA NA ”none” ...
#  $ gender    : chr  ”male” NA NA ”male” ...
#  $ species   : chr  ”Human” ”Droid” ”Droid” ”Human” ...
#  $ n_films   : int  5 6 7 4 5 5 4 5 2
```

# Есть ли пропущенные значения и сколько их?

```
is.na(SW) # матрица, где TRUE соответствует NA в исходных данных
```

```
#        name height  mass hair_color gender species n_films
#  [1,] FALSE  FALSE FALSE      FALSE  FALSE   FALSE   FALSE
#  [2,] FALSE  FALSE FALSE       TRUE   TRUE   FALSE   FALSE
#  [3,] FALSE  FALSE FALSE       TRUE   TRUE   FALSE   FALSE
#  [4,] FALSE  FALSE FALSE      FALSE  FALSE   FALSE   FALSE
#  [5,] FALSE  FALSE FALSE      FALSE  FALSE   FALSE   FALSE
#  [6,] FALSE  FALSE FALSE      FALSE  FALSE   FALSE   FALSE
#  [7,] FALSE  FALSE FALSE      FALSE  FALSE   FALSE   FALSE
#  [8,] FALSE  FALSE FALSE      FALSE  FALSE   FALSE   FALSE
#  [9,] FALSE  FALSE  TRUE      FALSE  FALSE   FALSE   FALSE
```

```
colSums(is.na(SW)) # суммируем число NA по столбцам
```

```
#       name     height       mass hair_color     gender    species    n_films
#          0          0          1          2          2          0          0
```

# Работа с переменными датафрейма при помощи $

М.А. Варфоломеева

Биологический факультет, СПбГУ

# Любую переменную датафрейма можно извлечь для анализа


https://www.shutterstock.com/ru/image-photo/nonthaburi-thailand-february-16-2017-lego-581529553

Мы продолжим разбирать работу с датафреймами на примере фрагмента данных о персонажах “Звездных войн” из базы SWAPI (https://swapi.co/).

Данные содержатся в файле `tiny_starwars.RData`.

# Обращение к переменным датафрейма при помощи $

Значок доллара $ позволяет извлечь из датафрейма любую переменную по ее имени.

```
имя_датафрейма$имя_переменной
```

```
SW$name
```

```
# [1] "Luke Skywalker" "C-3PO"          "R2-D2"          "Darth Vader"    "Leia Organa"
# [6] "Chewbacca"      "Han Solo"       "Yoda"           "Shmi Skywalker"
```

```
SW$species
```

```
# [1] "Human"          "Droid"          "Droid"          "Human"          "Human"
# [6] "Wookiee"        "Human"          "Yoda's species" "Human"
```

```
SW$mass
```

```
# [1]  77  75  32 136  49 112  80  17  NA
```

# С переменными можно работать как с векторами

Можно извлекать из переменных отдельные значения по их номерам.

```
SW$name[c(1, 6)] # Как зовут существ из строк 1 и 6 в датафрейме?

# [1] ”Luke Skywalker” ”Chewbacca”
```

# С переменными можно работать как с векторами

Можно извлекать из переменных отдельные значения по их номерам.

```
SW$name[c(1, 6)] # Как зовут существ из строк 1 и 6 в датафрейме?

# [1] ”Luke Skywalker” ”Chewbacca”
```

Можно использовать переменные в вычислениях.

```
table(SW$species) # Сколько существ разных видов в датафрейме?

#
#          Droid          Human        Wookiee Yoda's species
#              2              5              1              1
```

# Можно извлекать из переменных значения по условию и использовать их в вычислениях

```
sum(SW$species == 'Human') # Сколько всего людей?

# [1] 5

SW$name[SW$species == 'Human'] # Как зовут людей?

# [1] ”Luke Skywalker” ”Darth Vader”    ”Leia Organa”    ”Han Solo”       ”Shmi Skywalker”

mean(SW$mass[SW$species == 'Human'], na.rm = TRUE) # Какова средняя масса людей?

# [1] 85.5
```

# При помощи знака $ можно добавлять переменные в датафрейм

```
имя_датафрейма $имя_новой_переменной
```

```
SW$height_m <- SW$height / 100  # Переводим рост из сантиметров в метры
SW$height_m
```

```
# [1] 1.72 1.67 0.96 2.02 1.50 2.28 1.80 0.66 1.63
```

Созданные переменные будут добавлены в конец датафрейма.

```
head(SW, 2)
```

```
#             name height mass hair_color gender species n_films height_m
# 1 Luke Skywalker    172   77      blond   male   Human       5     1.72
# 2          C-3PO    167   75       <NA>   <NA>   Droid       6     1.67
```

# Работа с фрагментами датафрейма при помощи [ , ]

М.А. Варфоломеева

Биологический факультет, СПбГУ

# Любой произвольно выбранный фрагмент датафрейма можно извлечь для анализа


https://www.shutterstock.com/ru/image-photo/malaysia-feb-18-2018-mini-figure-1062484475

Мы продолжим разбирать работу с датафреймами на примере фрагмента данных о персонажах “Звездных войн” из базы SWAPI (https://swapi.co/).

Данные содержатся в файле `tiny_starwars.RData`.

# Любой элемент датафрейма можно извлечь при помощи [ , ]

```
[номер_строки , номер_столбца]
```

Датафрейм — это двумерная таблица, поэтому у его элементов два адреса.

```
head(SW)
```

```
#             name height mass hair_color gender species n_films height_m
# 1 Luke Skywalker    172   77      blond   male   Human       5     1.72
# 2          C-3PO    167   75       <NA>   <NA>   Droid       6     1.67
# 3          R2-D2     96   32       <NA>   <NA>   Droid       7     0.96
# 4    Darth Vader    202  136       none   male   Human       4     2.02
# 5    Leia Organa    150   49      brown female   Human       5     1.50
# 6      Chewbacca    228  112      brown   male Wookiee       5     2.28
```

Какой рост у Дарта Вейдера?

```
SW[4, 2] # Вызываем элемент из 4-й строки, 2-го столбца (рост Дарта Вейдера)

# [1] 202
```

# Любую переменную тоже можно извлечь при помощи [ , ]

В квадратных скобках [ , ] можно указать индекс или имя столбца и получить тот же результат.

Существа каких видов есть в нашем датасете?

```
# Эти варианты кода дадут одинаковый результат:
SW$species       # при помощи знака `$`
SW[ , 6]         # переменная номер 6
```

```
SW[ , 'species'] # переменная вид
```

```
# [1] "Human"          "Droid"          "Droid"          "Human"          "Human"
# [6] "Wookiee"        "Human"          "Yoda's species" "Human"
```

```
unique(SW[ , 'species'])
```

```
# [1] "Human"          "Droid"          "Wookiee"        "Yoda's species"
```

# Можно извлечь сразу несколько переменных

Для этого нужно использовать внутри `[ , ]` вектор индексов или имен столбцов.

В скольких фильмах участвовали персонажи из нашего датасета?

```
# Эти варианты кода дадут одинаковый результат:
SW[ , c(1, 7)]              # Столбцы 1 и 7
```

```
SW[ , c("name", "n_films")] # Имена и число фильмов
```

```
#             name n_films
# 1 Luke Skywalker       5
# 2          C-3PO       6
# 3          R2-D2       7
# 4    Darth Vader       4
# 5    Leia Organa       5
# 6      Chewbacca       5
# 7       Han Solo       4
# 8           Yoda       5
# 9 Shmi Skywalker       2
```

# Нужные строки можно извлечь по номерам или по условию

Можно использовать номера строк, если они известны заранее.

```
SW[c(8, 1), ] # Строки 8 и 1 - это мастер Йода и Люк Скайуокер
```

```
#             name height mass hair_color gender        species n_films height_m
# 8           Yoda     66   17      white   male Yoda's species       5     0.66
# 1 Luke Skywalker    172   77      blond   male          Human       5     1.72
```

Можно использовать поиск по условию, если номера строк точно не известны.

```
SW[SW$species == 'Droid', ] # Строки с данными про дроидов
```

```
#    name height mass hair_color gender species n_films height_m
# 2 C-3PO    167   75       <NA>   <NA>   Droid       6     1.67
# 3 R2-D2     96   32       <NA>   <NA>   Droid       7     0.96
```

# Мы можем выбрать только нужные данные, используя комбинацию фильтров

Допустим, мы хотим исключить из анализа дроидов (строки 2 и 3) и оставить только переменные, кодирующие имя рост и вес.

```
# Эти варианты кода дадут одинаковый результат:
SW[-c(2,3), -c(4:6, 8)]     # удаляем строки и столбцы по номерам
```

```
SW[SW$species != 'Droid', c('name', 'height', 'mass')] # выбираем по условию и именам
```

```
#             name height mass
# 1 Luke Skywalker    172   77
# 4    Darth Vader    202  136
# 5    Leia Organa    150   49
# 6      Chewbacca    228  112
# 7       Han Solo    180   80
# 8           Yoda     66   17
# 9 Shmi Skywalker    163   NA
```

# Создание датафреймов с нуля

М.А. Варфоломеева

Биологический факультет, СПбГУ

# Создание датафреймов с нуля


https://www.shutterstock.com/ru/image-vector/hand-crafted-stamp-brown-round-grunge-512055955

Иногда бывает нужно создать датафрейм вручную:

- Для рисования графиков в ggplot2, если исходные данные в виде вектора
- Для расчета предсказаний линейной регрессии

и т.д.

# Датафрейм можно создать из векторов одинаковой длины

```
p_name <- c('Luke Skywalker', 'Darth Vader', 'Obi-Wan Kenobi', 'Yoda')
planet <- c('Tatooine', 'Tatooine', 'Stewjon', NA)
starships <- c(2, 1, 5, 0)

jedi <- data.frame(name = p_name,
                   homeworld = planet,
                   n_starships = starships)
jedi
```

```
#             name homeworld n_starships
# 1 Luke Skywalker  Tatooine           2
# 2    Darth Vader  Tatooine           1
# 3 Obi-Wan Kenobi   Stewjon           5
# 4           Yoda      <NA>           0
```

# Текстовые векторы в датафреймах

По-умолчанию функция data.frame() превратит текстовые векторы в факторы.

```
str(jedi)
```

```
# 'data.frame': 4 obs. of  3 variables:
#  $ name       : Factor w/ 4 levels "Darth Vader",..: 2 1 3 4
#  $ homeworld  : Factor w/ 2 levels "Stewjon","Tatooine": 2 2 1 NA
#  $ n_starships: num  2 1 5 0
```

Это не всегда хорошо.

Например, имена нет смысла делать факторами, т.к. они не повторяются.

# Можно избежать превращения текста в факторы

```
jedi <- data.frame(name = p_name,
                   homeworld = planet,
                   n_starships = starships,
                   stringsAsFactors = FALSE)
str(jedi)
```

```
# 'data.frame': 4 obs. of  3 variables:
#  $ name       : chr  ”Luke Skywalker” ”Darth Vader” ”Obi-Wan Kenobi” ”Yoda”
#  $ homeworld  : chr  ”Tatooine” ”Tatooine” ”Stewjon” NA
#  $ n_starships: num  2 1 5 0
```

# И вручную преобразовать то, что нужно

```
jedi$homeworld <- factor(jedi$homeworld)
str(jedi)
```

```
# 'data.frame': 4 obs. of  3 variables:
#  $ name       : chr  ”Luke Skywalker” ”Darth Vader” ”Obi-Wan Kenobi” ”Yoda”
#  $ homeworld  : Factor w/ 2 levels ”Stewjon”,”Tatooine”: 2 2 1 NA
#  $ n_starships: num  2 1 5 0
```

# Загрузка внешних данных в R

В.М. Хайтов, к.б.н.

Биологический факультет, СПбГУ

# Роковой рейс “Титаника”


https://www.shutterstock.com/ru/image-photo/old-passenger-ship-rides-over-atlantic-760982245

Скачайте с сайта файлы `Titanic_xls_book.xls` и `Titanic.csv` и положите их в папку `data` внутри рабочей директории.

Информация в файлах идентична, но записана в разных форматах.

Данные: Dawson, 1995
Источник: пакет `datasets`

# Чтение .xls / .xlsx файлов в R

```
library(readxl) # Пакет для чтения файлов Excel

titanic <- read_xls("data/Titanic_xls_book.xls", sheet = "Лист1")
head(titanic)
```

```
# # A tibble: 6 x 5
#   Class Sex    Age   Survived  Freq
#   <chr> <chr>  <chr> <chr>    <dbl>
# 1 1st   Male   Child No         0.
# 2 2nd   Male   Child No         0.
# 3 3rd   Male   Child No        35.
# 4 Crew  Male   Child No         0.
# 5 1st   Female Child No         0.
# 6 2nd   Female Child No         0.
```

Результат чтения — объект класса `tibble`, большая часть его свойств такая же, как у `data.frame`.

```
?tibble # Справка о tibble
```

# Формат Excel: за и против

**Pro**

- Легко и быстро набивать данные.
- Можно загрузить в R без дополнительной подготовки.

**Contra**

- Excel используют далеко не все пользователи.
- Нужен специальный пакет для перевода данных в R.
- Среда Excel не требовательна к структурированию данных — существует вероятность ошибок (пропуски строк, лишние пробелы и т.п.).
- Файлы могут иметь очень большой размер.

# Не только Excel...


Помимо бинарных файлов, содержащих данные (.xls, .dbf, .sas, .sta), которые могут читаться только специальными программами существует и универсальный — текстовый формат.

Текстовые файлы (.txt, .csv) - могут читаться любыми системами, понимающими использованную кодировку текста.

# Файлы, содержащие значения, разделенные запятыми (CSV, Comma-Separated Values)

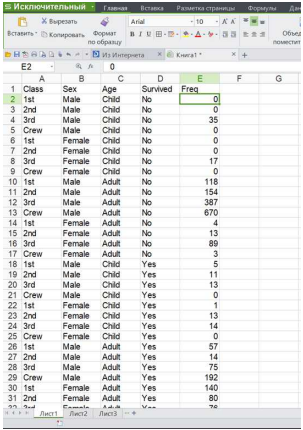
Это наиболее удобный тип файлов.

Значения разделены некоторым разделителем (запятой, точкой с запятой, пробелом, знаком табуляции и т.п.).

CSV-файлы можно просматривать и редактировать, к в электронных таблицах (Excel), так и в любом текстовом редакторе (Word, Notepad).

CSV-файлы небольшие по размеру.

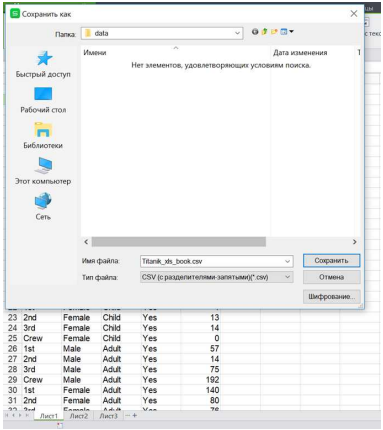
# Создание .csv файла в Excel




| | A | B | C | D | E | F | G |
|---|---|---|---|---|---|---|---|
| 1 | Class | Sex | Age | Survived | Freq | | |
| 2 | 1st | Male | Child | No | 0 | | |
| 3 | 2nd | Male | Child | No | 0 | | |
| 4 | 3rd | Male | Child | No | 35 | | |
| 5 | Crew | Male | Child | No | 0 | | |
| 6 | 1st | Female | Child | No | 0 | | |
| 7 | 2nd | Female | Child | No | 0 | | |
| 8 | 3rd | Female | Child | No | 17 | | |
| 9 | Crew | Female | Child | No | 0 | | |
| 10 | 1st | Male | Adult | No | 118 | | |
| 11 | 2nd | Male | Adult | No | 154 | | |
| 12 | 3rd | Male | Adult | No | 387 | | |
| 13 | Crew | Male | Adult | No | 670 | | |
| 14 | 1st | Female | Adult | No | 4 | | |
| 15 | 2nd | Female | Adult | No | 13 | | |
| 16 | 3rd | Female | Adult | No | 89 | | |
| 17 | Crew | Female | Adult | No | 3 | | |
| 18 | 1st | Male | Child | Yes | 5 | | |
| 19 | 2nd | Male | Child | Yes | 11 | | |
| 20 | 3rd | Male | Child | Yes | 13 | | |
| 21 | Crew | Male | Child | Yes | 0 | | |
| 22 | 1st | Female | Child | Yes | 1 | | |
| 23 | 2nd | Female | Child | Yes | 13 | | |
| 24 | 3rd | Female | Child | Yes | 14 | | |
| 25 | Crew | Female | Child | Yes | 0 | | |
| 26 | 1st | Male | Adult | Yes | 57 | | |
| 27 | 2nd | Male | Adult | Yes | 14 | | |
| 28 | 3rd | Male | Adult | Yes | 75 | | |
| 29 | Crew | Male | Adult | Yes | 192 | | |
| 30 | 1st | Female | Adult | Yes | 140 | | |
| 31 | 2nd | Female | Adult | Yes | 80 | | |

Исходный файл в Excel

# Создание .csv файла




Файл → Сохранить как →
CSV(с разделителями запятыми)

# Создание .csv файла


Внимание! Русифицированный Excel использует в качестве разделителя точку с запятой.

# Чтение .csv файла в R

```
read.table('data/Titanic.csv')
```

```
#                             V1
# 1  Class;Sex;Age;Survived;Freq
# 2           1st;Male;Child;No;0
# 3           2nd;Male;Child;No;0
# 4          3rd;Male;Child;No;35
# 5          Crew;Male;Child;No;0
# 6         1st;Female;Child;No;0
# 7         2nd;Female;Child;No;0
# 8        3rd;Female;Child;No;17
# 9        Crew;Female;Child;No;0
# 10        1st;Male;Adult;No;118
```

С дефолтными настройками получается плохо, так как не указаны важные параметры функции `read.table()`.

# Чтение .csv файла в R

```
read.table('data/Titanic.csv',
           sep = ';',
           header = TRUE)
```

```
#    Class    Sex   Age Survived Freq
# 1    1st   Male Child       No    0
# 2    2nd   Male Child       No    0
# 3    3rd   Male Child       No   35
# 4   Crew   Male Child       No    0
# 5    1st Female Child       No    0
# 6    2nd Female Child       No    0
# 7    3rd Female Child       No   17
# 8   Crew Female Child       No    0
# 9    1st   Male Adult       No  118
# 10   2nd   Male Adult       No  154

                        ...
```

Параметр `sep =';'` сообщает функции, что в качестве разделителя использован знак точка с запятой.

Параметр `header = TRUE` сообщает функции, что первую строку в файле надо рассматривать, как заголовки, соответствующие именам переменных.

# Чтение .csv файла в R

Остается только положить загруженные данные в некоторую переменную.

```
titanic <- read.table('data/Titanic.csv', sep = ';', header = TRUE)
```

# Внешние данные из Сети

Данные могут храниться, как на локальном компьютере, так и в Сети.

Можно вместо пути к локальному файлу указать путь к сетевому ресурсу.

```
read.table("http://www.foo.foo.edu/foo/foo/data/titanic.csv", sep = ";", header = TRUE)
```

# Опрятные данные (Tidy data)

В.М. Хайтов, к.б.н.

Биологический факультет, СПбГУ

# Исходные данные часто приходится приводить в порядок

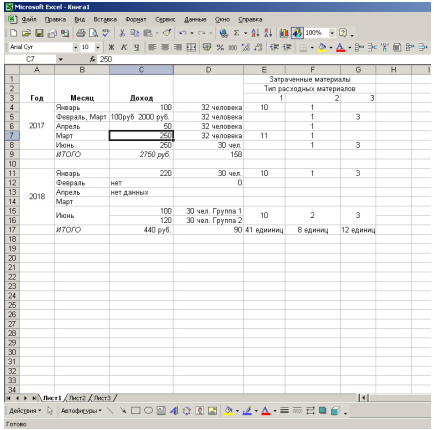

https://www.shutterstock.com/image-photo/magic-broom-witch-hat-on-white-731844499

На подготовку данных к анализу уходит до 80% времени.

Существуют определенные правила предоставления данных.

Данные, построенные в соответствии с этими требованиями, называются **tidy data**, или **опрятные данные**.

# Проблемы начинаются уже в электронных таблицах




| | A | B | C | D | E | F | G | H |
|---|---|---|---|---|---|---|---|---|
| 1 | | | | | Затраченные материалы | | | |
| 2 | | | | | Тип расходных материалов | | | |
| 3 | Год | Месяц | Доход | | 1 | 2 | 3 | |
| 4 | | Январь | 100 | 32 человека | 10 | 1 | | |
| 5 | | Февраль, Март | 100руб 2000 руб. | 32 человека | | 1 | 3 | |
| 6 | 2017 | Апрель | 50 | 32 человека | | 1 | | |
| 7 | | Март | 250 | 32 человека | 11 | 1 | | |
| 8 | | Июнь | 250 | 30 чел. | | 1 | 3 | |
| 9 | | *ИТОГО* | *2750 руб.* | 158 | | | | |
| 10 | | | | | | | | |
| 11 | | Январь | 220 | 30 чел. | 10 | 1 | 3 | |
| 12 | | Февраль | нет | 0 | | | | |
| 13 | 2018 | Апрель | нет данных | | | | | |
| 14 | | Март | | | | | | |
| 15 | | Июнь | 100 | 30 чел. Группа 1 | | | | |
| 16 | | | 120 | 30 чел. Группа 2 | 10 | 2 | 3 | |
| 17 | | *ИТОГО* | 440 руб. | 90 | 41 единиц | 8 единиц | 12 единиц | |

**Основные ошибки**

- Объединенные ячейки
- Отсутствующие заголовки столбцов, вместо них стоят какие-то числа (или ничего не стоит) или слишком длинные имена заголовков
- В одной ячейке находится сразу несколько значений
- Разнородные данные в пределах одного столбца
- Нет стандартного обозначения пропущенных значений

# Tidy data: Основные принципы

- Данные (Dataset) — это набор значений (Values) количественных, текстовых, логических.
- Каждое значение принадлежит какой-то переменной (Variable) и какому-то наблюдению (Observation).
- При планировании структуры датасета надо помнить, что проще работать с колонками, как единым целым, чем со строками. Поэтому удобно, когда в таблице столбцы — переменные, строки — наблюдения.
- Каждая переменная имеет имя (заголовок столбца).

# Tidy data: Основные принципы

- Строки (наблюдения) именовать можно, но не обязательно. Если строки имеют имя, то оно должно быть уникальным (отличаться от остальных имен строк хотя бы номером).
- Каждая переменная содержит только один тип данных.
- Все значения одной переменной измерены в одинаковых единицах (например, для всех наблюдений измерения сделаны в сантиметрах).
- Пропущенные значения маркируются единообразно (например, пустые ячейки, или `NA`).

# Управление данными

В среде R создано несколько удобных пакетов для преобразования данных:

- reshape2
- dplyr
- tidyr

# Широкий и длинный формат данных

Одни и те же данные можно представить в широком и длинном формате.

Широкий формат — каждая строка содержит информацию о нескольких наблюдениях.

Длинный формат — каждая строка содержит информацию о единственном наблюдении.

# Данные о погибших на “Титанике” в широком и длинном формате


https://www.shutterstock.com/ru/image-photo/old-passenger-ship-rides-over-atlantic-760982245

С представлением данных в широком и длинном формате мы познакомимся на примере “Титаника” (`Titanic.csv`)

Данные: Dawson, 1995
Источник: пакет `datasets`

# Широкий формат данных

```
titanic <- read.table('data/Titanic.csv',
                      sep = ';',
                      header = TRUE)
head(titanic, 8)
```

```
#   Class    Sex   Age Survived Freq
# 1   1st   Male Child       No    0
# 2   2nd   Male Child       No    0
# 3   3rd   Male Child       No   35
# 4  Crew   Male Child       No    0
# 5   1st Female Child       No    0
# 6   2nd Female Child       No    0
# 7   3rd Female Child       No   17
# 8  Crew Female Child       No    0
```

В каждой строке содержится информация о многих пассажирах — это широкий формат.

Широкий формат удобнее для чтения и презентации данных в табличном виде, но сложнее для обработки.

# Длинный формат данных

```
library(tidyr)
long_titanic <- uncount(titanic, weights = Freq)
head(long_titanic, 10)
```

```
#     Class  Sex   Age Survived
# 3     3rd Male Child       No
# 3.1   3rd Male Child       No
# 3.2   3rd Male Child       No
# 3.3   3rd Male Child       No
# 3.4   3rd Male Child       No
# 3.5   3rd Male Child       No
# 3.6   3rd Male Child       No
# 3.7   3rd Male Child       No
# 3.8   3rd Male Child       No
# 3.9   3rd Male Child       No
```

Каждая строка содержит информацию о каждом отдельном человеке.

Длинный формат не удобен для чтения, но значительно удобнее для обработки и визуализации.

Подходящую функцию для преобразования в длинный формат можно найти в пакете `tidyr`.

# Что мы знаем и что будет дальше

М.А. Варфоломеева, PhD

Биологический факультет, СПбГУ

# Что мы знаем?


https://www.shutterstock.com/ru/image-photo/red-mug-book-stack-on-table-723557938

R — это специализированный язык статистического программирования.

Чтобы освоить R, как и любой другой язык, нужно много тренироваться.

# Что будет дальше?


https://www.shutterstock.com/ru/image-photo/road-171584552

Сейчас мы обсудили только самые основы языка и особенности организации данных для анализа.

В следующих модулях вы узнаете, как строить графики и делать базовые статистические расчеты.

# Используемые источники


https://www.shutterstock.com/ru/image-photo/businessman-touching-global-network-data-exchanges-566877226
https://www.shutterstock.com/ru/image-photo/toy-magician-casting-spell-not-person-425596252?src=9IFUp7OtjFojD9EEo8ZPfQ-1-18
https://www.shutterstock.com/ru/image-photo/woman-drowns-sea-698646385
https://www.shutterstock.com/ru/image-photo/crazy-super-hero-calculator-588734510
https://www.shutterstock.com/ru/image-photo/graphs-charts-vectors-vector-theory-together-252777811
https://www.shutterstock.com/ru/image-photo/honolulu-hawaii-usa-jan-11-2016-361145654
https://www.shutterstock.com/ru/image-photo/vintage-gift-box-package-address-card-92183338
https://www.shutterstock.com/ru/image-photo/silhouette-time-lapse-sequence-boy-leaping-472802506
https://www.shutterstock.com/ru/image-photo/ripe-orange-isolated-on-white-background-100359398

https://www.shutterstock.com/image-vector/vector-illustration-set-coffee-tea-cups-211563346
https://www.shutterstock.com/image-photo/double-sided-wall-glass-cup-coffee-116343187
https://www.shutterstock.com/image-photo/cup-hot-earl-grey-tea-isolated-229500454
https://www.shutterstock.com/image-photo/cup-tea-isolated-on-white-615600377
https://www.shutterstock.com/image-photo/cup-tea-isolated-on-white-background-281634209
https://www.shutterstock.com/image-photo/mug-black-tea-on-white-background-94380859
https://www.shutterstock.com/image-photo/sugar-cube-on-white-background-646458415
https://www.shutterstock.com/image-photo/glass-water-105772223

https://www.shutterstock.com/ru/image-photo/true-false-blackboards-on-wood-background-261530159
https://www.shutterstock.com/ru/image-photo/collection-house-numbers-one-twentyfive-176481500
https://www.shutterstock.com/ru/image-photo/true-false-question-vintage-letterpress-wood-373414852
SWAPI, https://swapi.co/
https://www.shutterstock.com/ru/image-photo/bologna-italy-march-1-2015-star-257048308
https://www.shutterstock.com/ru/image-photo/nonthaburi-thailand-february-16-2017-lego-581529553
https://www.shutterstock.com/ru/image-photo/malaysia-feb-18-2018-mini-figure-1062484475
https://www.shutterstock.com/ru/image-vector/hand-crafted-stamp-brown-round-grunge-512055955
https://www.shutterstock.com/ru/image-photo/old-passenger-ship-rides-over-atlantic-760982245
Dawson, R.J.M., 1995. The "unusual episode" data revisited. Journal of Statistics Education, 3(3).
https://www.shutterstock.com/ru/image-photo/red-mug-book-stack-on-table-723557938
https://www.shutterstock.com/ru/image-photo/road-171584552

**Санкт-Петербургский государственный университет**

www.spbu.ru