r\_automatisation

r Sys.Date()

### Поиск информации в сети

Функция RSiteSearch() перенаправляет на search.r-project.org, где представлена информация по искомому термину. Обратите внимание, что термин должен быть заключён в кавычки. Пример: RSiteSearch("biostatistics"):

### Аспекты использования R Studio

##### find-replace

**Find** — это поле, в которое вводится искомый паттерн из символов, или, простыми словами, искомое слово;  
**Replace** — это поле, в которое вводится паттерн, на который заменяется паттерн из Find;  
**Next, Prev, All** соответственно переключают на следующий искомый паттерн, предыдущий или выбирают все сразу;  
**Replace, All** выбирают, производить ли замену только в одном случае или сразу во всех. При этом, если нажимать Replace раз за разом, то замена будет производиться для следующего паттерна;  
**In selection** переключает поиск в режим “искать только в выделенном заранее сегменте”;  
**Match case** -учитывать регистр (большая буква или маленькая) или нет; **Whole word** -искать ли только слово целиком или включить в результат слова, внутри которых находится искомое слово;  
**Regex** переключает поиск в режим регулярных выражений.  
Интересный факт: если мы хотим добавить к строке какую-то ещё строку, например, ко всем “data” добавить “data\_c1”, нам достаточно заменить строку “data” на строку “data\_c1”.

##### Кнопка полной очистки памяти

Удалит все данные, переменные и функции. Её стоит использовать, когда мы хотим выполнить весь код заново, чтобы убедиться, что он работает без ошибок.

##### Структура папки проекта

Прежде всего, для чистоты проекта лучше создавать папку **data** рядом с остальными вышеперечисленными файлами. Уже в ней можно создать следующие папки:  
**originals** — сюда просто складываем все исходные файлы, нужные для проекта. Важной особенностью этой папки должно являться то, что файлы в ней мы ни в коем случае не меняем. Таким образом всегда останется резервная чистая копия, если что-то пойдёт не так в других папках. Для работы с файлами мы просто копируем файлы из этой папки в другие;  
**pics** — сюда сохраняем картинки/графики и прочую графическую информацию, появляющуюся в ходе анализа;  
**raw** — сырые файлы данных. Именно из этой папки мы будем читать данные и загружать их в R для последующего анализа. В ней так же не стоит менять файлы, чтобы гарантировать, что все изменения, проводимые в R, проводятся на основе тех же данных, что есть у коллег;  
**documents** — здесь лежит вся документация, такая как протоколы, SAPы (планы статистического анализа), листы регистрации и подобное.

### Структура R Markdown

##### Заголовок

**Заголовок YAML**, окружённый —, он не обязательно должен присутствовать, но в нём можно указывать (ниже примеры):

Пример YAML-заголовка  
Заголовок : title: “Example title” — отображается в начале документа большим шрифтом;  
Подзаголовок: subtitle: “Example subtitle” — отображается под заголовком шрифтом поменьше;  
Автор: author: “User Name”;  
Дата: date: “r Sys.Date()” — в данном примере дата будет обновляться при каждом выводе документа в печать;  
Вывод: output: word\_document — в данном примере вывод настроен на вывод документа формата .docx.  
Чанки. Вставляются с помощью **Ctrl + Alt + I** или просто с помощью написания

```в начале чанка и закрывающих в конце чанка```

##### Чанк

В фигурных скобках после указания r как используемого в чанке языка можно через пробел указать название чанка, чтобы позже было проще ориентироваться; Шестерёнка в правом верхнем углу позволяет настраивать в графическом режиме параметры чанков;  
Кнопка в виде белого треугольника с зелёной полоской выполнит все фрагменты кода выше данного текущего чанка. Также можно сделать с помощью постановки курсора в чанк и горячих клавишей **Ctrl + Alt + P**;  
Крайний правый зелёный треугольник просто выполнит весь код в чанке.

**echo = TRUE** — показывать ли код при печати документа. Обычно это не нужно, поэтому есть смысл установить FALSE;  
error = FALSE — показывать ли при печати ошибки;  
**eval = TRUE** — выполнять ли код в чанке. Обычно мы действительно хотим, чтобы код в чанке выполнялся, однако иногда есть смысл временно или навсегда выключить один или несколько чанков из кода.  
**fig.width = 7 и fig.height = 7**

##### Первый чанк

В нём же следует писать и код подключения пакетов.  
Сюда же можно рекомендовать класть и собственноручно написанные функции с тем, чтобы они автоматически подгружались при запуске и чтобы не случилось внезапно так, что функция используется раньше, чем будет объявлена.

##### Текст

`r mean (1:10)` кусок кода будет вычислен в тексте

1. Первый элемент
2. Первый вложенный
3. Второй вложенный

* Ненумерованный
  + Первый вложенный ненумернованный
  + Второй вложенный ненумернованный

##### Создание автоматического оглавления: toc, toc-title

Для этого используется параметр **toc: yes**.

Дополнительно можно ограничить количество возможных уровней списков в оглавлении с помощью соответствующего параметра. В примере количество уровней ограничено тремя: toc\_depth: 3.

Наконец, по умолчанию заголовок оглавления будет “Table of Contents”, но зачастую мы хотим изменить его. Для этого используется параметр toc-title: “Оглавление” как в примере. Обратите внимание, что, в отличие от остальных параметров, связанных с автоматическим оглавлением, этот параметр пишется без отступов.

### Пакеты

##### Две функции для подключения пакетов к текущему проекту

**library(package\_name)** (название пакета можно писать без кавычек) — основная функция, которой стоит пользоваться всегда. Дело в том, что в случае, если пакет ещё не установлен, эта функция вернёт ошибку и прервёт выполнение кода. Таким образом мы не окажемся в ситуации, когда обнаружим отсутствие нужных функций уже тогда, когда напишем какой-то работающий частично код;

**require(package\_name)** — эта функция возвращает FALSE вместо ошибки, что удобно, когда мы пишем сложные функции, зависящие от наличия возможности подключить пакет.

Но, если мы по какой-то причине не хотим явно подключать проект, можно использовать функции напрямую из установленного пакета с помощью синтаксиса packagename::functionname(). Например: dplyr::select().

Отключение detach(package:dplyr). В этом примере мы выгружаем пакет dplyr и его функции становятся недоступны. Внутри функции обязательно нужно писать package: (с двоеточием), а потом название пакета без кавычек.

##### dplyr

Основные его функции (глаголы, говорящие, что именно мы делаем):

mutate() — изменяет переменные, добавляет новые;  
select() — выбирает переменные;  
filter() — фильтрует объекты по условиям;  
summarise() — вычисляет сводные статистики;  
arrange() — сортирует по переменным;  
group\_by() — группирует по значениям переменных;  
\*\_join() — группа глаголов для склеивания двух таблиц по ключу.

##### tidyr

Поворачивать данные, то есть, преобразовывать их в длинный и широкий форматы: pivot\_longer(), pivot\_wider();  
Разворачивать данные из вложенных списков в простые таблицы: unnest\_longer(), unnest\_wider();  
Наоборот, делать из таблиц вложенные переменные: nest(), unnest();  
Разделять и объединять столбцы по разделителю строк: separate(), unite();  
Заполнять отсутствующие значения определёнными значениями или удалять их: complete(), drop\_na(), fill(), replace\_na().

##### readr, readxl, haven

Три основных пакета, которые помогают читать данные. readr предназначен для чтения самых распространённых форматов данных: .csv, .txt, *.tsv;*  
*redxl помогает читать файлы* .xlsx или, иными словами, всё, что пересылается в формате Excel;  
haven читает форматы \*.sas7bdat, .sap, .dta, .sav, .por, то есть данные из SAS.

##### purrr

Этот пакет может поначалу казаться сложным для понимания, однако на самом деле он просто расширяет функционал уже известного семейства функций **apply()**. Функция **map()** и её расширения позволяют итерироваться по элементам векторов или списков самыми разными способами.

##### tibble

1. **tibble** в отличие data.frame он не приводит строки к факторам автоматически. Это нужно сделать самостоятельно, однако таким образом мы всегда в точности знаем, что находится в каждой переменной датафрейма;
2. Имена переменных остаются такими же, какими были. Например, в переменной с названием “variable name” пробел не будет заменён на точку, вместо этого имя будет окружено апострофами (обычно находятся на букве “ё”), что сохраняет ожидания от названий;
3. tibble оценивает переменные лениво- при создании мы можем объявлять одну переменную на основе другой (но эта другая должна быть указана первой).

##### ggplot2 и ggpubr

Для визуализации данных существует сразу два пакета: базовый ggplot

Дополнительные интересные особенности вроде автоматического расчёта и добавления на график p-value, реализованы в пакете ggpubr.

##### flextable

* печать статистических таблиц ##### stringr
* работа со строками ##### lubridate
* работа с датами ##### DescTools, psych  
  полезные функции для групп статистик, корреляционных матриц

### Загрузка дф

data <- read\_tsv("data\_tsv.tsv", quote = "\"")

## Rows: 100 Columns: 13  
## ── Column specification ────────────────────────────────────────────────────────  
## Delimiter: "\t"  
## chr (3): Группа, Пол, Группа крови  
## dbl (10): Возраст, Рост, Базофилы\_E1, Эозинофилы\_E1, Гемоглобин\_E1, Эритроци...  
##   
## ℹ Use `spec()` to retrieve the full column specification for this data.  
## ℹ Specify the column types or set `show\_col\_types = FALSE` to quiet this message.

### Выбор столбцов

##### dplyr::select(), where()

Такие формы записи верны:

data %>% select(`Группа крови`, `Возраст`, function(x) anyNA(x))  
data %>% select(`Группа`, function(x) !is.factor(x))  
data %>% select(`Группа`, (function(x) is.factor(x)) | where(function(x) is.numeric(x)))  
# data %>% select(function(x) any(str\_detect(x, "Жен")))

Обратить внимание что если в результате проверки условия появляется NA, то операция И/ИЛИ не проходит и будет ошибка, как тут:

data %>% select(function(x) sd(x, na.rm = TRUE) > 1)  
часть переменных строковые и sd выдаст NA

##### tidyselect::contains(), tidyselect::matches(), tidyselect::starts\_with(), tidyselect::ends\_with()

##### tidyselect::all\_of(), tidyselect::any\_of()

Для того, чтобы функция выдавала ошибку, если отсутствует хотя бы одна из переменных, внутри select() следует использовать all\_of(). Мы можем выбирать переменные более мягко - any\_of().

##### Иерархия операторов

Приоритет логических операторов от высшего к низшему.

x < y, x > y, x <= y, x => y, x == y, x != y (операторы сравнения)  
!x (логическое НЕ)  
x & y (логическое И)  
x | y (логическое ИЛИ)

##### tidyselect::everything()

Есть простая функция, которая просто значит “все остальные столбцы, кроме тех, которые уже были указаны”. Например, так мы можем выбрать определённые столбцы, а потом просто все остальные, которые будут выведены в том порядке, в котором они были, за исключением тех, что мы уже явно взяли.

##### Выбор с одновременным изменением имён

Мы можем выбирать переменные, одновременно изменяя их имена.

##### Почему стоит использовать имена, а не номера

Если мы много времени провели, используя только базовый R, то может показаться хорошей идеей использовать индексы столбцов для их выбора.  
Но почему этого не стоит делать  
Этого не стоит делать, потому что местоположения столбцов могут меняться, а вот их имена мы скорее всего уже знаем.