**F<-** - знак присвоения (горячая клавиша Alt+клавиша "минус"). Так же можно использовать знак "=", но принято, что лучше первый вариант.

В различных языках программирования **не всегда 1 == 1**. Чтобы исключить такие ошибки можно воспользоваться функцией **near()** из пакета **dplyr.**

Если изменил какую-то базовую функцию, то её нужно просто удалить из глобального окружения.

**Alt+Shift+K** - доступ ко всем хоткеям.

**ctrl + shift + m** - хоткей для пайпоператора %>%.

**#### комментарий ####** - позволяет свернуть код, который лежит под комментарием до следующего комментария.

Атомарные вектора - все элементы одного типа.

regular expressions (регулярные выражения) можно узнать в поиске в справке.

**Основные типы:**

**1. logical(TRUE/FALSE).**

**2. integer(целые числа).**

**3. numeric/double (числа с плавающей точкой).**

**4. complex (комплексные числа).**

**5. character (строки).**

**6. raw (байтовые последовательности).**

String(строка) - элемент вектора типа character.

library(help = "datasets") - выдаст список достпуных открытых встроенных датасетов.

typeof(вектор x) - выдаст тип вектора. is.\*(где \* обозначает любой из типов) - выдаст TRUE или FALSE.

**Естественным считается приведение слева направо по цепочке logical - integer - numeric/double - character. Это значит, что если в векторе записаны разные типы, онни будут приведены к одному типу и если он правее в цепочке, значит к нему и**

**приведут.**

Фактор - гибрид целочисленного (integer) и строкового (character) вектора. factor - обычно качественные переменные.

levels - все уникальные значения, встречающиеся в факторе.

Если мы хотим увидеть непосредственное действие функции (формулу, метод и т.д.), то нужно напечатать название функции без скобок. Например sd без скобок выдаст формулу стандартного отклонения.

Если в выводе есть .C, .Call, .Fortran, .External, .Internal, .Primitive, то это обращение к скомпелированному коду: нужно смотреть исходный код R (наприер var).

Если в выводе есть UseMethod или standartGeneric, то это method dispatch для классов S3/S4 (полиморфизм, например plot).

**x %in% y** - проверяет, есть ли вложение x в переменной y. Например 1:5 %in% c(1, 2, 5) вернёт TRUE FALSE TRUE.

as.\* - принудительное приведение к определённому типу.

**floor(x)** - округляет число с плавающей точкой вниз до ближайшего целого.

**read.csv('...', na.strings = '...')** - символ под кавычками будет переведён в NA.

Клавиша "Tab" при вводе слова выдаёт подсказку для быстрого ввода.

**Ctrl + Enter** - выполнение операции.

**== - знак равентсва. != - знак неравенства**.

**Если щёлкнуть на функцию и нажать F1** - выдаст справку по функции.

1 : 67 создаст последовательность чисел от одного до 67. Создание переменной будет включать всю последовательность.

c(...) - combine, concatenation - объединяет значения под скобкой в одну переменную.

Можно создать пустой вектор q <- c().

[...] - вызов элемента, который указан под этим номером. Первый под номером один и т.д. (В Пайтоне первый номер под цифрой ноль).

Если нужно указать неколько символов из квадратной скобки, то нужно в этой скобке опять указать [c(1, 2, 3...)], либо [1, 2, 3...].

Если указать номер, которого нет, программа выдаст значение NA.

[...] - вызов подходящего индекса. Например, если в переменной несколько чисел, мы хотим узнать какие из них больше нуля, можно ввести:

var > 0

var[var > 0]. Программа выдаст только те значения, которые больше нуля.

x[-length(x)] - вернёт все элементы, кроме последнего.

Если к переменной, содержащей несколько чисел прибавить какое-либо число, это число прибавится к каждому числу в переменной.

Если такую переменную сравнить с каким-либо числом в ответе будут логические высказывания по этому сравнению для каждого числа (TRUE or FALSE).

(x <- y) - если результат присвоения заключить в круглые скобки, то он автоматически попадёт в печать.

Знак & означает "и" - объединение логических условий.

mean(var) - среднее по выборке.

sum(x) - сумма элементов числового вектора x.

sd(x) - стандартное отклонение.

abs(n) - абсолютное значение числа n - модуль.

list(1, x) - позволяет создавать разнотипные переменные (цифры, буквы и т.д.). В отличие от вектора.

Вызов из листа через [[1]][1 ].

list - список.

Доступ к элементам списка:

1. Одинарные скобки **[] -** действуют векторные правила индексирования, возвращаемое значение - подсписок.

2. Двойные скобки **[[]] -** скалярный номер элемента или его полное имя в кавычках, возвращаемое значение - элемент списка.

3. Знак доллара **$** - частичное имя элемента, возвращаемое значение - элемент списка.

Dataframe - двумерная таблица с данными. Аналоги - Excel spreadsheet, SQL-таблица. Большинство реальных данных удобно хранить в формате датафрейма. Обычно: строки - наблюдения, столбцы - переменные. Датафрейм наследует свойства

матрицы (прямоугольная форма) и списка (переменные могут быть разных типов).

length(df) - к дф вернёт количество столбцов (переменных), а не общее количество элементов. names(df) - также вернёт имена столбцов.

data.frame() - основная команда для работы с данными. Не забывать давать названия векторам. Создаёт таблицу.

dir(pattern = "\*.csv") - укажет все файлы с таким разрешением в директории.

dir(pattern = '\*.csv')

grants <- data.frame()

for (i in dir(pattern = '\*.csv')){

temp.df <- read.csv(i)

grants <- rbind(temp.df, grants)

} - обьединит файлы из директории в один датафрейм.

install.packages('...', dependencies = T) - установит библиотеку и все остальные библиотеки, необходимые для её работы.

update.packages() - обновит пакеты.

library(help = "datasets") - команда выдаст список всех доступных датасетов.

data('...') - добавит датасет в рабочую среду.

source('...') - считывает скрипт из рабочей директории.

**Правила переписывания (recycling): как работает арифметика на векторах разной длины?**

**1. Длина результата равна длине большего из векторов.**

**2. Меньший вектор дублируется (переписывается) несколько раз, чтобы длина переписанного вектора совпала с длиной большего.**

**3. Если длина большего вектора не делится нацело на длину меньшего вектора, выдаётся предупреждение.**

Арифметические операции

+         сложение

-         вычитание

\*         умножение

/         деление (5 / 2 = 2.5)

^ или \*\* возведение в степень (5^2 = 25 или 5\*\*2 = 25)

x %% y    остаток от деления (5 %% 2 = 1)

x %/% y   целая часть от деления (5 %/% 2 = 2)

Логические операции

<       меньше

<=      меньше или равно

>       больше

>=      больше или равно

==      проверка на равенство

!=      не равно

!x      не x

x | y   x или y

x & y   x и y

**x %o% y** - перемножит все возможные варианты векторов x и y. (типа комбинаторика).

TRUE можно сокращенно обозначать T

FALSE можно сокращенно обозначать F

read.table(...) read.csv(...) - команды для загрузок готовых датасетов. Если есть проблемы с чтением данных, можно воспользоваться справкой ?read.csv, там много подсказок, какие показатели ввести в функцию, чтоб всё нормально прочиталось.

Если файл находится не в рабочей директории, есть три способа решить эту проблему:

изменить рабочую директорию (в RStudio: Session -> Set Working Directory -> Choose Directory...)

переместить файл в актуальную рабочую директорию

указать полный путь к файлу, например: read.csv("C:/Users/username/documents/evals.csv")

Команда head() выводит верхушку наших данных. Поумолчанию это 6 верхних строчек.

Команда tail() последние шесть наблюдений.

Команда View() (с большой буквы) - покажет данные в таблице в отдельном окне. Щелчок справа по переменной, содержащей данные делает тоже самое.

Эта функция позволяет увидеть до 1000 наблюдений.

Функция str() выводит все имеющиеся переменные из данных, а также показывает, к какому классу относится переменная (сводка об объекте). Сокращение от structure.

summary() выводит некоторые сводные данные по нашим переменным.

names() выводит названия колонок.

Factor (или пункт chr) - номинативные переменные.

Если ввести название датафрейма, где хранятся данные и после этого поставить знак $ (например data$...), а дальше нажать на Tab, появится подсказка с названиями всех переменных.

Если выполнить программу, программа выдаст все значения этой переменной.

Также через знак $ можно создавать новые столбцы, называть их и присваивать значения.

Функция nrow() возвращает колчиство строчек датафрейма.

ncol() количество столбцов (колонок, колчиства переменных) датафрейма.

[1, 1] - вызовет первую строчку, первый столбец.

[5, ...] - вызовет всю информацию по каждой переменной для пятой строчки.

[..., 5] - вызовет всю информацию по каждой строчке для пятого столбца.

[..., 2:5] - то же самое для промежутка от второго до пятого столбца.

'матрица'[ , 'число'] - произойдёт схлопывание размерности (выдаст вектор). Если хотим в виде матрицы, но один столбец, то нужно 'матрица'[ , 'число', drop = FALSE].

**subset**

Функция subset позволяет отбирать все данные по какому-либо одному показателю переменной (например из пола выдать все показатели только по женщинам).

Например subset(my\_data, gender == 'female') выдаст всю информацию только по женщинам. Если разбивка по двум показателям, нужно использовать знак &.

df <- subset(mtcars, select = c("wt", "mpg", "disp", "drat", "hp")) - создаст subset из mtcars только с переменными "wt", "mpg", "disp", "drat", "hp".

Subset(mtcars, select = -disp) – выдаст все, кроме disp.

rbind (от rowbind - связать строку) - соединяет два датафрейма одинаковой ширины (с одинаковым количеством столбцов), также должны быть одинаковые имена у переменных.

cbind (от columnbind - связать колонку)

Чтобы просмотреть весь список доступных датасетов, выполните команду library(help = "datasets")

Команда data(mtcars) добавит датасет в рабочую среду.

Команда help(mtcars) выведет информацию о датасете.

Команда my\_data <- mtcars запишет датасет в новую переменную.

rm() - удалить переменную.

if else - проверяют, выполняется ли условия и на основе этого выполняют ту или иную команду. Если нам нужно выполнить несколько команд

(например {print('...'), print(...)}, используются фигурные скобки.

ifelse переводит всю эту сложную конструуцию в одну строчку. Для ifelse не нужно писать print. Эта функция (в отличие от if) позволяет использовать значения, где вектор имеет больше одного значения.

ifelse(условие, значение при истине, значение при лжи). Внутри этой конструкции можно вставить ещё одну конструкцию ifelse. Например ifelse(runif(8) > 2/3, 'Камень', ifelse(runif(8) > 1/3, 'Ножницы', 'Бумага')).

Цикл for позволяет проводить операцию много раз. Например с каждым элементом вектора. for обязательно должен включать какой-либо вектор, по которому он проверяется.

for (i in 1:nrow(mydata))

if (mydata$gender[i] == 'male'){

print(mydata$score[i])

}

} -такая команда выдаст все оценки только для мужчин.

**which(x == a)** - проверяет, какие элементы объекта х равны a, и возвращает вектор, содержащий порядковые номера этих элементов.

**which(x == a, arr.ind = T) -** вернёт номер строк и колонок, соответствующих значений, а также их названия (индексы).

**which.min(max)(x)** - индекс минимального или максимального значения. Но только первого, если есть повторы, такую функцию применять нежелательно. Лучше использовать which(x == max(x))

mydata$quality <- rep(NA, nrow(mydata)) - создаст новую переменную и в ней вектор, состоящий из пустых значений на всей длине датафрейма.

while - оператор, выполняющий программу до тех пор, пока не будет выполнено определённое условие. Для этого оператора нужно задать изначальное значение.

x$y <- factor(x$y, labels = c('...', '...') - придаст названия значениям.

median(...) - выдаст медиану.

sd(...) - среднее отклонение.

range(...) - размах.

**aggregate(x = ..., by = ..., FUN = ...)** - Разбивает данные на подмножества, вычисляет сводную статистику для каждого и возвращает результат в удобном виде. Эта функция поумолчанию выводит данные в формате датафрейма.

Можно записать короче **aggregate(hp ~ vs, m, mean)**. Указывем переменную, которую хотим разбить, далее переменную, по которой разбиваем, далее откуда берём данные, далее функцию, которую применяем. Можно указать несколько разбивающих

переменных через знак + **(hp ~ vs + am, ..., ...). aggregate(cbind(mpg, disp) ~ am + vs, m, sd)** - если две и более разбиваемых переменных.

**attach(df)** - закрепит путь к датафрейму, далее можно будет не писать название датафрейма, а только название его переменных.

**colnames(mean\_hp\_vs) <- c('VS', 'Mean HP')** (в скобках пример) - переименует колонки.

Чтобы установить новую библиотеку, нужно зайти сверху в Tools -> Install packages.

Чтобы запустить пакет из библиотеки, нужно написать команду library(...).

Функция describe из пакета(библиотеки) psych имеет много полезных свойств для статистики.

describeBy - описывает всякие статистические свойства для определённых групп. Если групп больше одной, нужно написать group = list(группы, группы, ...). В этой функции команда digits = 1 оставит один знак после запятой.

Команда fast = TRUE оставит только самую базовую статистику.

describeBy(m$qsec, group = list(m$vs, m$am), mat = T, digits = 1, fast = T) - выдаст основную статистику по скорости разгона автомобиля для разных типов двигателей и разных коробок передач.

**sum(is.na(...))** - покажет, есть ли пропущенные наблюдения. is.na(...) - покажет, какие из наблюдений пропущены.

Пропущенные значения не могут быть посчитаны. Но мы можем их не учитывать с помощью команды na.rm = TRUE, используемой внутри другой команды. Например mean(m$mpg, na.rm = T).

Функция aggregate автоматически игнорирует пропущенные значения. Если мы хотим, чтоб не игнорировала, можно указать функцию na.action = и указать, что с ними делать. describe тоже игнорирует.

hist(...) - построит гистограмму. hist(..., breaks = ...) изменит ширину столбца. hist(..., xlab = '...') назовёт гистограмму.

boxplot(x ~ y, df) - создаст боксплот.

plot(x, y) - создаст скаттерплот.

mosaicplot(...) - мозаичный график.

getwd() - команда, показывающая рабочую директорию.

write.csv(df, "df.scv") - сохранит датафрейм в директорию.

save(my\_mean, file = "my\_mean.RData") - сохранит переменную.

df$am <- as.factor(df$am)

levels(df$am) <- c('наименование') - переименует. Но am должно быть фактором. Это можно сделать в одну строчку, например df$status <- factor(df$status, labels = c('Not funded', 'Funded')).

**table()** - позволяет посмотреть, сколько наблюдений в том или ином классе.

**dim()** - позволяет посмотреть размерность таблицы (сколько столбцов).

**prop.table()** - показывает пропорции (проценты).

**rle(x)** - похожа на table - вычисляет длины и значения одинаковых значений в векторе.

barplot() - создаст столбчатую диаграмму.

barplot(t2, legend.text = T, args.legend = list(x = 'topright'), beside = T) - создаст столбчатую диаграмму с легендой.

mosaicplot() - создаст мозаичный график. - размер прямоугольников соответствует количеству наблюдений.

- цвет прямоугольников - величине значимости отклонения ожидаемых и наблюдаемых частот в этой ячейке.

- если значения стандартизированных остатков больше 3х, можно считать, что в этой ячейке зафиксированы значимые отклонения.

**binom.test(x = ..., n = ..., p = ...)** - x - количество исходов, n - количество наблюдений, p - априорная вероятность. Биноминальный тест позволяет ответить на вопрос, насколько эмпирическое распределение некоторого ряда событий отличается от теоретически

предсказанного. Показывает p-уровень значимости.

**chisq.test(полученные, p = (ожидаемые), rescale.p = T)** - тест хиквадрат Пирсона. Если есть несколько значений таблицы сопряжённости, то сначало надо создать переменную, в которой связать строки через rbind(...) и только потом использовать

тест хиквадрат.

**chisq.test(...)$stdres** - покажет стандартизированные остатки.

Если наблюдений мало, лучше использовать критерий Фишера **- fisher.test(...)**

**shapiro.test(df**) - позволяет узнать о нормальности распределения (p > 0,05 - распределение статистически не отличается от нормального). Игнорирует пропущенные значения.

**bartlett.test(Sepal.Length ~ Species, df1)** - позволяет узнать о гомогенности дисперсий.

**scale()** - функция, позволяющая совершить стандартизацию вектора, то есть делает его среднее значение равным нулю, а стандартное отклонение - единице (Z-преобразование).

**t.test** - тест Стьюдента.

**t.test(Sepal.Length ~ Species, df1)** - проведёт t-тест, который чаще всего применяется для проверки равенства средних значений в двух выборках. (p < 0,05 - средние статистически отличаются).

**t.test(df1$Sepal.Length, mu = 8)** - сравнит среднее значение Sepal.Length со средним значением, равным восьми.

**t.test(df1$Sepal.Length, df1$Sepal.Width, paired = T)** - проведёт тест для зависимых переменных.

**mean\_cl\_normal(...)** - покажет границы доверительных интервалов.

**wilcox.test(...)** - (тест Вилкоксона или тест Мана - Уитни) похож на t-тест, но применяется либо когда данных мало, либо когда распределение не слишком нормальное.

~ - указвыет зависимость переменной от другой переменной.

x ~ y + z - зависит от двух переменных.

x ~ y:z - зависит от взаимодействия переменных между собой.

x ~ y + z + y:z - и то, и то. - x ~ y \* z.

x ~ (y + z + q)^2 - все предикторы взаимодействуют по парам.

x ~ y + Error(subject/y) - несколько повторений на одном предикторе. Если один испытуемый участвует в разных экспериментах, то лучше использовать дисперсионный анализ с Error. aov(z ~ y + Error(x / y)), где z — объясняемая

переменная, x — та переменная, за счет которой ожидается получить воздействие на объясняющую переменную / переменные

**aov(x ~ y, df)** - запускает дисперсионный анализ.

**summary(aov(...)** - покажет остатки, сумму квадратов и т.д**. Если F - значительно больше нуля и/или P < 0,05, то предиктор значимо предсказывает значение**.

**model.tables(df, 'mean')** - выдаст общее среднее и средние в каждой группе.

**TukeyHSD(...) -** проводит множественные попарные сравнения с автоматической поправкой Тьюки.

function(x, y){ - создание своих функций. Буквы внутри функии не создают новых переменных в программе. Если нужно создать переменную в глобальном окружении, тогда нужно использовать знак <<-

s <- x + y

d <- x - y

return(c(s, d))

}

Очень простые функции (например x \* 2) можно писать без фигурных скобок и без return.

function(x) x \* 2

**rnorm(x, y, z)** - создаст нормальное распределение. Выдаст x раз со средним y и стандартным отклонением z.

**runif(x, y, z)** - создаст равномерное распределение. Выдаст x раз случайных чисел в промежутке от y до z.

rm(...) - удалит переменную.

df[is.na(df)] <- mean(df, na.rm = T) - заменит пропущенные значения на средние значения.

my\_na\_rm <- function(x){ - создаст функцию, которая будет заменять пропущенные значения на средние.

x[is.na(x)] <- mean(x, na.rm = T)

return(x)

}

**cor.test(x$..., x$...)** - проведёт тест на корреляцию (коэффициент корреляции Пирсона). - cor.test(~ mpg + hp, df)

**cor(df) -** выдаст коэффициент корреляции для всего датафрейма.

**pairs(df)** - создаст диаграмму рассеивания для каждой пары.

rnorm(100) - создаст нормальное распределение из 100 чисел.

lm(x ~ y, df) - создаст регрессионную модель. По умолчанию игнорирует пропущенные значения. (Обычно для линейной регрессии)

lm(x ~ ., df) - создаст модель по всем предикторам из датафрейма. Если мы не указываем имя переменной, а указываем номер, тогда нужно будет указать датафрейм без неё. Например lm(df[[1]] ~ ., df[-1])

lm(x ~ .^2, df) - со всеми взаимодействиями второго уровня (все взаимодействуют по парам).

summary(lm) - более развёрнутые данные. Интерсепт показывает, чему будет равно значение зависимой переменной, если независимые будут равны нулю. Adjusted R-squared - это обычный  R-squared минус штраф за количество предикторов. Чем больше разница между   Adjusted R-squared и Multiple R-squared тем больше вероятность что мы переобучили модель

summary(lm(am ~ hp, mtcars)) – hp под интерцептом - это slope. **Slope** показывает, на сколько процентов изменится **зависимая** переменная при изменении **независимой** на один процент.

**Intercept** показывает, чему будет равна **зависимая** переменная, если **независимая** переменная будет равна нулю.

coef(lm(x ~ y, df) – выдаст коэффициенты (intercept и slope).

boot(…) – проведёт bootstrap-ирование. Подробности в справке.

anova(x, y) - покажет, если разница в дисперсиях моделей. anova(lm(…))

**kruskal.test(x ~ y, df) –** сравнит группы между собой. Непараметрическая функция. Применяется, если распределение значимо отличается от нормального и/или дисперсии негомогенны.

step(fit\_full, direction = 'backward') - автоматический отбор моделей (будет удалять предикторы, которые плохо предсказывают модель и остановится на первом оптимальном варианте). Аргумент scope задаст границы поиска.

**confint(...)** - показывает доверительные интервалы. **Если доверительный интервал накрывает 0, это означает, что гипотезу о равенстве коэффициента 0 отвергать не следует => из модели данную переменную можно исключить. Но исключать надо**

**строго по одной, начиная с наименее значимой. Свободный член (intercept) при необходимости можно исключить строго на последнем шаге.**

fitted\_values\_mpg <- data.frame(mpg = df$mpg, fitted = fit3$fitted.values) - создаст датафрейм, в котором будут указаны реальные значения и значения, предсказанные регрессионной моделью. (df - mtcars, fit3 - предварительно созданная

регрессионная модель).

**predict(fit3, new\_data\_frame)** - предскажет значения для нового датафрейма по уже имеющейся модели.

df$cyl <- factor(df$cyl, labels = c('four', 'six', 'eight')) - переведёт количество цилиндров в фактор (название) и придаст имена.

Если в линейной регрессии одна из переменных номинативная (фактор), то одно из значений этой переменной будет указано как интерсепт, а также будет указано, на сколько изменится значение при переходе с одного уровня этой переменной на

другой уровень.

df$coefficients - покажет коэффициенты регрессии.

# Сначала перебором вводим все варианты зависимых переменных

x <- c ("mpg", "disp", "drat", "hp",

"mpg + disp", "mpg + drat", "mpg + hp",

"disp + drat", "disp + hp",

"drat + hp",

"mpg + disp + drat", "mpg + disp + hp", "mpg + drat + hp",

"disp + drat + hp",

"mpg + disp + drat + hp")

# Создаем будущий вектор для значений Adj R^2

r.sq <- NA

# Заполняем вектор

for (i in c (1 : length (x))) {

formula <- as.character (c ("wt ~", x [i]))

fit <- lm (formula, mtcars)

r.sq [i] <- summary (fit) $ adj.r.squared

}

# Самый большой - искомый. В формулу вводим результат следующей команды:

x [which.max (r.sq)] - создаст цикл, который проверит различные комбинации регрессионного анализа для разных переменных и выдаст результат с максимальным R^2 ajusted.

**glm()** - Функцию glm() в R можно использовать для подбора обобщенных линейных моделей. Эта функция особенно полезна для подбора моделей логистической регрессии, моделей регрессии Пуассона и других сложных моделей. Обычно используется, когда зависимая и независимые переменные являются факторами (номинативными переменными).

**glm(x ~ y + z, df, family = 'binomial')** - для биномиальной модели. Чтобы создать модель без предикторов **(intercept only model),** запись должна быть формы **glm(x ~ 1, df, family = ‘binomial’). Intercept** здесь обычно будет означать значение логарифма. **exp(glm(…))** – вернёт шансы.

**summary(glm(…))** – более развёрнутая статистика.

Если в модели есть и факторы, и количественные переменные, то интерцепт будет показывать значения при количественной равной нулю, а сама количественная будет показывать, насколько изменится интерцепт при изменении количественной на единицу.

**predict(fit, type = 'response')** - предсказание для биномиальной модели в процентах.

**apply((df or matrix), (1 or 2), function)** - функция, которая позволяет к df или матрице, к строчке или столбцу (1 или 2) применить другую функцию.

Например apply(mtcars, 2, sd) - вернёт стандартное отклонение для каждого столбца.

Если есть пропущенные значения apply(mtcars, 2, sd, na.rm = T)

apply позволяет прописывать простые функции прямо внутри себя. apply(df, 1|2, function(x) x[x < 0])

colMeans()

colSums() - среднее и суммы для колонок и строчек.

rowMeans()

rowSums()

set.seed() - устанавливает определённое "семя" для случайных значений. При одинаковом значении семени случайные значения будут повторяться.

seq.along() - создаёт последовательность от одного до "количество колонок в наших данных". = 1:ncol().

norm\_test <- apply(iris[, 1:4], 2, function(x) shapiro.test(x)$p.value) - проведёт тест на нормальность распределения для четырёх столбцов iris и вернёт p.value.

**Apply**

**lapply(list, function)** - как apply, но для списка (list-а).

**sapply()** - тоже самое, только изначально пытается выдать вектор или матрицу, а если не получается, то выдаёт список. sapply - от слова simplify - упрощённый. Т.к. вектор или матрица проще, чем список (list)

**tapply(vector, index, function)** - тоже самое, **что aggregate.**

**vapply(list, function, FUN.VLAUE = type, ...)** - тоже самое, что sapply, только мы сами указываем, что мы хотим получить в результате применения функции.

**grepl(..., ...)** - функция, которая показывает, есть ли первый аргумент во втором.

**by(df, df$x, function)** - группирует датафрейм по определённой переменной.

**colMeans** - функция, которая покажет средние в датафрейме (для датафрейма просто mean не подойдёт).

**by(iris[1:4], iris$Species, function(x) sapply(x, function(col) shapiro.test(col)))** - пременяет тест **Шапиро-Уилка** для каждой количественной колонки, разбитой по четырём видам.

my\_data <- data\_frame(x = rnorm(10000), y = rnorm(10000), f = factor(rep(1:2, 5000))) - (data\_frame из библиотеки dplyr) - создаст датафрейм с тремя столбцами, первые два - 10000 чисел нормального распределения, третий - фактор с

повторами 1 и 2.

Особенности пакета dplyr смотреть в "Установленных библиотеки".

**order(df$x)** - покажет индексы, вызвав которые получишь от меньших значений к большим. df[order(df$x)] - отсортирует df от меньших к большим.

**colnames(df)** - выдаст имена колонок.

**gsub(' ', '', df)** - функция, которая находит все знаки (первый показатель) и заменяет их на другие (второй показатель).

**grep('слово', vector)** - покажет в какие элементы вектора входит искомое слово.

**grepl('[A-Z]', df)** - покажет, есть ли буквы в датафрейме.

**grepl('буква.', df)** - буква, за которой следует любой другой символ. Не покажет те результаты, где эта буква стоит в конце слова.

tolower(текст) - переведёт в нижний регистр.

toupper(текст) - переведёт в верхний регистр.

as.data.frame(do.call(rbind, df)) - сгруппируем датафрейм "необычным" способом.

**do.call** - применение заданной функции к списку аргументов.

**seq(число, число2, число3)** - последовательность от первого числа до второго с шагом в третье число**. seq(число, число 2, length = число 3)** - последовательность с равными шагами длиной число 3.

**diag(матрица)** - выдаст пересекающиеся значения в матрице (например: если провести cor(df) - получится матрица, где в некоторых строках будут пересекаться одинаковые переменные и их значение будет равно 1. С помощью **diag(matrix) -> 0**

их можно превратить в нули и уже проще определять, какое значение **корреляции** максимальное).

**sapply(df, function(x) na.omit(x))** - пропустит **NA**.

**df[complete.cases(df), ]** - также пропустит NA.

**sapply(df function(x) x[!is.na(x) & x < 0])** - уберёт NA и выдаст значения, которые меньше нуля.

**rep(x:y, times = z)** - повторит z раз последовательность от x до y. **rep(x:y, each = z)** - повторит подряд z раз каждое число от x до y. **rep(x:y, length = z)** - повторит от x до y до длины вектора равной z.

**sort(вектор x)** - расположит значения по порядку.

**unique(вектор x)** - удалит повторяющиеся значения из вектора.

**runif(x)** - напишет x раз случайные числа от 0 до 1.

system.time({...}) - внутри какая либо функция или ещё что-нибудь такое. Покажет, сколько времени заняло выполнение этой функции.

**all.equal(x, y)** - покажет, равны ли x с y. Если есть различия, напишет, во сколько в среднем будет разница. Если сравниваются буквы, то напишет, в каком количестве различия.

**sample(x:y, replace = T)** - генератор случайных чисел от x до y. replace означает, что могут быть повторы. У **sample(x, prob =** (можно выставить различные значения для каждого отдельного значения x. Но нужно указать именно для каждого)).

**x[rep(c(T, F))] -** Пропустит все значения с чётными индексами

**all(условие (например x > 0))** - если в векторе все значения больше нуля, выдаст TRUE, если не все, то FALSE.

**any(условие (например x > 0))** - если в векторе есть хоть одно значение больше нуля, то вернёт TRUE.

**attr(x, 'название столбца атрибутов')** <- 'Название атрибута' - создаст у вектора x атрибуты с названием.

**attributes(x) - покажет, какие атрибуты есть у x. attributes(x) <- NULL** - уберёт атрибуты.

**factorial(x) - вернёт факториал x. Это значит, что если x равен 5, то factorial(5) = 1 \* 2 \* 3 \* 4 \* 5. В алгебре факториал обозначается знаком !.**

**t(матрица)** - перевернёт матрицу, ряды станут колонками, а колонки рядами.

**is.null(x$y)** - проверяет, существует ли переменная с таким именем.

**paste(x, collapse = 'какой либо знак')** - склеит все элементы x в одну строку, отделяться один от другого будет знаком, указанным в кавычках. **paste(c('углекислый', 'веселящий'), 'газ', collapse = ', а также ')** - выдаст 'углекислый гаp, а

также веселящий газ.

**paste(c('углекислый', 'веселящий'), 'газ') -** выдаст 'углекислый газ', 'веселящий газ'.

**paste0('текст', число)** - склеивает аргументы. Например, если есть матрица, colnames(matrix) <- paste0('column', 1:5) - назовёт каждую колонку и прономерует её.

**strsplit('текст', 'разбивающий знак', fixed = T)** - разобьёт текст на разные части по разбивающиему знаку. Если ходим разбить на отдельные символы strsplit('текст', NULL)

**outer(vector, vector2, paste, sep = '...')** - возвращает все возможные комбинации векторов.

В R есть три похожих ключевых слова, NA, NaN и NULL. Они различаются по смысловой нагрузке.

NA -- это пропущенное значение ("not available"). Например, респондент не ответил на все вопросы предложенной анкеты, или данные с метеостанции за определённый период потерялись из-за сбоя оборудования. NA в этом случае обозначает,

что эти данные существуют и имеют смысл, но их не удалось узнать.

NaN -- "not-a-number" -- результат недопустимой арифметической операции, например 0/0 или Inf - Inf.

NULL -- отсутствие объекта, "пустота". Применяется в тех случаях, когда объект действительно не существует, не может иметь осмысленного значения.

Для проверки значений есть три функции, is.na, is.nan и is.null, соответственно.

**merge(df, df2, by = 'x')** - если в двух датафреймах есть одинаковая переменная 'x', два этих датафрейма объединятся в один, результаты будут указаны только для общих значений икса.

Если переменную назвали именем вызова какой либо функции, после этого, чтоб вызвать функцию, а не переменную, нужно воспользоваться конструкцией stats::название функции.

if(!require(installr)) { - обновит R.

install.packages("installr");

require(installr)

}

updateR()

anyNA(...) - проверит, есть ли значения NA.

Если нам нужно написать что- либо, заключённое в кавычки, нужно воспользоваться знаком \\. Например "Операция \"Ы\""

**nchar('текст')** - покажет количество символов в тексте. Пробел считается символом. В пакете stringr есть аналог str\_length(...)

cat('текст') - напечатает 'как есть'. В print(...) не так.

**formatC('число')** - округлит значения до ближайших значимых и уменьшит количество знаков после запятой.

**Если создавать вектор c('слово1', 'слово2', 'слово3'), то их значимость будет выставлена в том же порядке. Будет считаться, что слово3 больше, чем слово2, а слово2 больше, чем слово1.**

**ordered(...) - покажет уровни по порядку. Теперь можно будет сделать сравнение и выдать результаты, которые больше или меньше какого либо показателя. Например temp <- c('cold', 'norm', 'hot')**

**ft <- ordered(sample(temp, 14, replace = T), temp)**

**ft[ft >= 'norm'] - выдаст только нормальную и тёплую температуру. Или sort(ft) - от холодной к горячей.**

**cut(численный вектор, интервал)** - получает на вход численный вектор и разбивает на интервалы.

?options(...) - набор активных настроек сессии R, отвечающих за подсчёт и вывод результатов вычислений. Важные параметры options: digits - количество знаков при печати чисел, error - поведение при ошибке, width - длина строки при печати

векторов и матриц. По умолчанию, все строковые переменные становятся факторами, отменить такое поведение можно вызовом options(stringsAsFactors = F). В новых версиях по умолчанию строки уже не становятся факторами.

**sign(число)** - вернёт -1, если число меньше нуля, 0, если число равно нулю и 1, если число больше нуля.

**rbinom(x, y, z)** - Эта функция генерирует необходимое количество случайных значений заданной вероятности из заданной выборки. Найдёт x раз из y чисел с вероятностью z. z должно быть от нуля до единицы. Результаты будут от нуля до значения

y. Чем меньше z, тем ближе значения будут к 0, чем больше z, тем ближе значения будут к y. При z = 1 все значения будут выдаваться равным y.

**replicate(x, y)** - потворит x раз действие y.

**outer(x, y, функция (например paste или paste0)** - перебор всех возможных комбинаций аргументов.

**function(x)** <- Vectorize(функция, 'аргумент функции, который мы хотим векторизовать') - если функция изначально не векторизована, а мы хотим её векторизовать.

**expand.grid(вектора, вектор)** - создаст датафрейм из всех комбинаций.

**transform(df, название переменной = действие с переменной (например x = -x - значения x станут отрицательными)) -** позволяет изменять переменные или создавать новые. transform(df, new = x + 5) - создание новой переменной.

**levels(factor)** - покажет уровни факторной переменной. **levels(factor) <- levels(c(x, y...)) -** изменит факторы на x, y и т.д.

**relevel()** – изменит порядок уровней.

**В основах R** есть написанная нами функция **combin\_count**, которая позволяет решать задачи по **комбинаторике.**

**log(x, y) –** выдаст логарифм числа **x** по основанию **y. log(8, 2) = 3.**

**kmeans(df, 1)** – посчитает k-средние для кластеризации.

<https://stackoverflow.com/questions/25959385/how-to-draw-the-plot-of-within-cluster-sum-of-squares-for-a-cluster> - ссылки, как рисовать графики кластеров.  
<https://uc-r.github.io/kmeans_clustering>  
<https://towardsdatascience.com/the-complete-guide-to-clustering-analysis-10fe13712787>

**prcomp(df, center = T) - Principal Components Analysis** – анализ главных компонентов. **summary(prcomp(…)) – cumulative proportion** – покажет какой **процент** предсказывается/объясняется таким количеством главных компонент.

data(swiss)

fit <- prcomp(swiss, center = T)

plot(fit, type = "l")

summary(fit)

biplot(fit)

**factanal(df, factors = 2, rotation = "varimax") -** факторный анализ

**dist(df)** – расстояние для построения иерархической модели разбивки на кластеры. (расчет матрицы расстояний)

**hclust(dist(df))** - иерархическая кластеризация

**cutree(hclust(dist(df)), число)** - номер кластера для каждого наблюдения, число указывает, на сколько кластеров разбиваем.