Основы программирования в R

Загрузка данных и их описание

Алла Тамбовцева, НИУ ВШЭ

Содержание

Загрузка файла и кодировка

Загрузим данные из файла `firtree.csv`, в котором хранятся результаты вымышленного опроса посетителей ёлочного базара, и сохраним их в переменную `tree`.

Показатели в файле:

- `gender` – пол респондента;
- `ftype` – тип хвойного дерева, которое оценивал респондент;
- `height` – высота хвойного дерева в сантиметрах;
- `expenses` – сумма (в рублях), которую респондент готов отдать за хвойное дерево;
- `score` – балл, на который респондент оценил вид хвойного дерева (1 – очень плохо, 5 – отлично);
- `wish` – ответ на вопрос «Хотели бы, чтобы Вам подарили такое хвойное дерево?» (да, нет).

Так как в файле есть текст на кириллице и создавался он на Mac OS/Linux, необходимо добавить аргумент `encoding` со значением кодировки `"UTF-8"`, иначе на Windows вместо русских букв в тексте будут крокозябры:

```
tree <- read.csv("D:/Downloads/firtree.csv",
                 encoding = "UTF-8")
```

```
tree <- read.csv("/Users/allat/Desktop/firtree.csv")
```

Похожая проблема может возникнуть, если наоборот, файл с кириллицей создавался на Windows, а загружается в R на Mac/Linux. Тогда нужно будет выставить кодировку `"Windows-1251"`.

Если добавление кодировки в `encoding` не решает проблему (текст не отображается в читаемом виде), нужно перед работой с файлом запустить следующую строчку кода:

```
Sys.setlocale("LC_CTYPE", "ru_RU.UTF-8")
```

Этот код сохранит настройки языка и кодировки, и файлы на русском языке будут благополучно открываться.

Посмотрим на датафрейм — функция `View()` открывает датафрейм в отдельной вкладке:

```
View(tree)
```

Внимание: первая буква у `View()` заглавная!

Теперь запросим первые строки датафрейма:

```
head(tree)
```

```
##   X gender         ftype height score expenses wish
## 1 1 female пихта Нобилис    190     3     1051   да
## 2 2   male пихта Нобилис    174     3     2378  нет
## 3 3 female    сосна Крым    248     4      655   да
## 4 4 female    сосна Крым    191     1     2934   да
## 5 5 female    сосна Крым    147     3     1198  нет
## 6 6   male    сосна Крым     91     3     2139   да
```

По умолчанию функция `head()` выдает первые 6 строк, но это можно изменить:

```
# первые 8 строк
head(tree, 8)
```

```
##   X gender            ftype height score expenses wish
## 1 1 female    пихта Нобилис    190     3     1051   да
## 2 2   male    пихта Нобилис    174     3     2378  нет
## 3 3 female       сосна Крым    248     4      655   да
## 4 4 female       сосна Крым    191     1     2934   да
## 5 5 female       сосна Крым    147     3     1198  нет
## 6 6   male       сосна Крым     91     3     2139   да
## 7 7   male ель обыкновенная    151     5      702   да
## 8 8 female ель обыкновенная     94     2     2707  нет
```

Аналогичным образом можно вывести последние строки датафрейма:

```
View(tail(tree))
```

Здесь функцию `tail()` мы заключили в `View()`, чтобы строки выводились в удобном формате — не в консоль, а в отдельном окне.

Техническое описание данных

Для начала запросим размерность датафрейма: число строк и число столбцов.

```
# 1200 строк и 7 столбцов
dim(tree)
```

```
## [1] 1200    7
```

Функция `dim()` возвращает вектор из двух элементов, причем на первом месте всегда идёт число строк, на втором — число столбцов. Если нам нужно только число строк или только число столбцов, можно выбрать нужный элемент по индексу, а можно поступить проще — воспользоваться готовыми функциями.

Функция `ncol()` возвращает число столбцов, а функция `nrow()` — число строк.

```
ncol(tree)
```

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

```
nrow(tree)
```

```
## [1] 1200
```

Если мы хотим получить техническое описание датафрейма — сколько в нём строк и столбцов, какого типа эти столбцы, можно воспользоваться функцией `str()`. Эта функция (`str` от *structure*) возвращает структуру любого объекта, не только датафрейма, поэтому, если не совсем ясно, какой объект выдала какая-нибудь функция из неизвестной библиотеки, можно смело её использовать.

Посмотрим на структуру датафрейма `tree`:

```
str(tree)
```

```
## 'data.frame':    1200 obs. of  7 variables:
##  $ X       : int  1 2 3 4 5 6 7 8 9 10 ...
##  $ gender  : Factor w/ 2 levels "female","male": 1 2 1 1 1 2 2 1 1 2 ...
##  $ ftype   : Factor w/ 4 levels "ель обыкновенная",..: 2 2 4 4 4 4 1 1 1 3 ...
##  $ height  : int  190 174 248 191 147 91 151 94 138 221 ...
##  $ score   : int  3 3 4 1 3 3 5 2 5 4 ...
##  $ expenses: int  1051 2378 655 2934 1198 2139 702 2707 713 1521 ...
##  $ wish    : Factor w/ 3 levels "","да","нет": 2 3 2 2 3 2 2 3 3 3 ...
```

Со столбцами `X`, `height`, `score` и `expenses` всё понятно, это обычные целочисленные столбцы типа *integer*. С остальными столбцами интереснее — они имеют тип `factor`. `Levels` здесь — это уникальные значения в векторе.

Тип `factor` используется в тех случаях, когда нечисловые, качественные, значения кодируются числами. Другими словами, когда числа «ненастоящие», когда с ними нельзя работать как с числами в математике. Например, если вместо значений `"female"` и `"male"` в столбце `gender` мы будем ставить 0 и 1, мы всё равно не сможем говорить, что 1 здесь больше 0, это какие-то наши условные обозначения, результат договоренности. Или, например, если мы будем кодировать любимый цвет респондента числами от 1 до 4 (красный, жёлтый, зелёный, синий), мы не сможем сравнивать эти числа и утверждать, что 4 в два раза больше 2, потому что это то же самое, что сравнивать слова «жёлтый» и «синий». Считать среднее значение по такому набору чисел тоже неправильно, даже если технически мы можем все числа сложить и поделить на их количество, потому что результат будет неинтерпретируемым. Ведь непонятно, что такое средний цвет, равный, к примеру, 2.5.

Особого внимания заслуживает столбец `wish`. Помимо очевидных значений `"да"` и `"нет"` здесь есть значение . На самом деле это пустые ячейки, которые считались в R таким образом. Чтобы они нам не мешали, давайте ещё раз загрузим файл, добавив опцию `na.strings = ` , которая принудит R считать такие ячейки за полноценные пропущенные значения `NA`:

```
tree <- read.csv("D:/Downloads/firtree.csv",
                 encoding = "UTF-8", na.strings = "")
```

Посмотрим на структуру обновленного датафрейма:

```
str(tree)
```

```
## 'data.frame':    1200 obs. of  7 variables:
##  $ X       : int  1 2 3 4 5 6 7 8 9 10 ...
##  $ gender  : Factor w/ 2 levels "female","male": 1 2 1 1 1 2 2 1 1 2 ...
##  $ ftype   : Factor w/ 4 levels "ель обыкновенная",..: 2 2 4 4 4 4 1 1 1 3 ...
##  $ height  : int  190 174 248 191 147 91 151 94 138 221 ...
##  $ score   : int  3 3 4 1 3 3 5 2 5 4 ...
##  $ expenses: int  1051 2378 655 2934 1198 2139 702 2707 713 1521 ...
##  $ wish    : Factor w/ 2 levels "да","нет": 1 2 1 1 2 1 1 2 2 2 ...
```

Всё исправилось!

Поиск пропущенных значений

Теперь мы точно знаем, что в некоторых столбцах есть пропущенные значения (NA's). Попробуем их посчитать. Для начала воспользуемся функцией `complete.cases()`, которая вернёт нам вектор из значений `TRUE` и `FALSE`, где `TRUE` означает, что строка в таблице не содержит пропущенные значения (*case* — это строка, то есть одно наблюдение). Выведем первые несколько значений вектора:

```
head(complete.cases(tree))
```

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

Теперь, чтобы посчитать число полностью заполненных строк, нам достаточно посчитать число `TRUE`. Сделать это очень просто: R воспринимает значения `TRUE` как 1, а `FALSE` — как 0, поэтому можно просто суммировать все значения в векторе выше:

```
sum(complete.cases(tree))
```

```
## [1] 1198
```

Но нам нужен противоположный набор значений, ведь мы хотим посчитать число строк с пропущенными значениями! Поэтому к `complete.cases()` нужно добавить отрицание. Отрицание в программировании обычно задаётся с помощью восклицательного знака. Поставим его перед функцией и получим «перевёрнутый» вектор, где `TRUE` и `FALSE` поменялись местами.

```
sum(!complete.cases(tree))
```

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

Получается, в датафрейме `tree` у нас есть две строки, в которых есть хотя бы одно пропущенное значение.

Важно: в R есть ещё одна функция для поиска пропущенных значений — `is.na()`:

```
sum(is.na(tree))
```

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

В нашем случае результаты с `complete.cases()` и `is.na()` совпадают, но так будет не всегда. Функция `complete.cases()` проверяет заполненность строк, а функция `is.na()` — заполненность ячеек. Допустим, у нас есть маленький датафрейм такого вида:

```
test <- cbind.data.frame(a = c(NA, 2, 3),
                         b = c(NA, NA, 1))
test
```

```
##    a  b
## 1 NA NA
## 2  2 NA
## 3  3  1
```

В нём две строки, содержащие хотя бы один `NA`, но всего пропущенных значений три. Сравним результаты:

```
sum(!complete.cases(test))
```

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

```
sum(is.na(test))
```

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

Содержательное описание данных

Выведем описательные статистики по всему датафрейму `tree` с помощью функции `summary()`:

```
summary(tree)
```

```
##        X             gender                  ftype         height
##  Min.   :   1.0   female:612   ель обыкновенная:258   Min.   : 70.0
##  1st Qu.: 300.8   male  :588   пихта Нобилис   :326   1st Qu.:115.0
##  Median : 600.5                sосна датская   :289   Median :157.0
##  Mean   : 600.5                sосна Крым      :327   Mean   :159.1
##  3rd Qu.: 900.2                                       3rd Qu.:203.2
##  Max.   :1200.0                                       Max.   :250.0
##      score          expenses        wish
##  Min.   :1.000   Min.   : 302.0   да  :611
##  1st Qu.:2.000   1st Qu.: 904.8   нет :587
##  Median :3.000   Median :1630.5   NA's:  2
##  Mean   :3.005   Mean   :1629.0
##  3rd Qu.:4.000   3rd Qu.:2300.0
##  Max.   :5.000   Max.   :2999.0
```

Для количественных показателей функция возвращает минимальное и максимальное значения (`Min.` и `Max`), среднее арифметическое и медиану (`Mean` и `Median`), а также нижний и верхний квантили (`1st Qu.` и `3rd Qu.`). Так, для столбца `height` получаем:

- высота 50% деревьев в данных не превышает значение 157 см;
- высота 25% деревьев в данных не превышает значение 115 см;
- высота 75% деревьев в данных не превышает значение 203.2 см.

Для качественных показателей функция возвращает частоты — сколько раз то или иное значение встречается в столбце. Количество пропущенных значений тоже учитывается.

Паттерны пропущенных значений

Для дальнейшей работы с пропущенными значениями нам понадобятся дополнительные библиотеки. Установим их.

```
install.packages("VIM")
install.packages("mice")
```

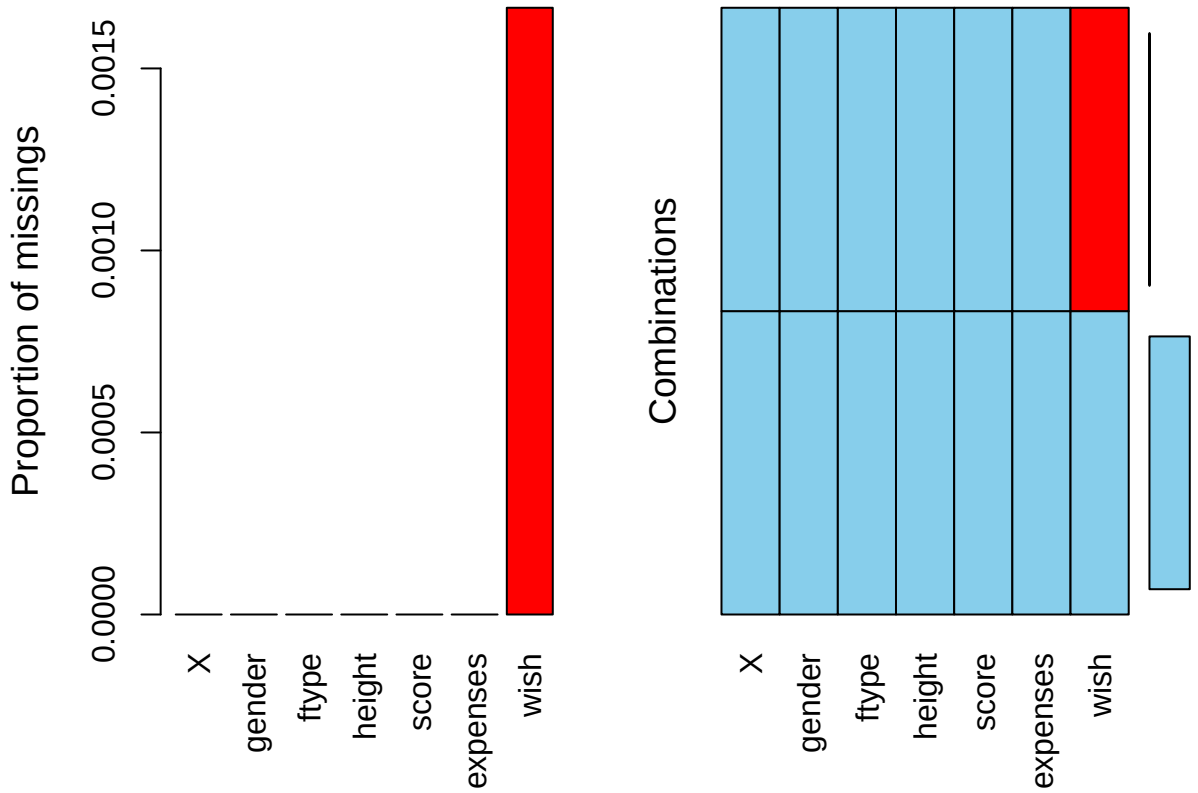
Обратимся к ним:

```
library(VIM)
library(mice)
```

Выведем графики, которые покажут, в каких переменных пропущенных значений больше всего и как выглядит таблица с пропущенными значениями (паттерны пропущенных значений).

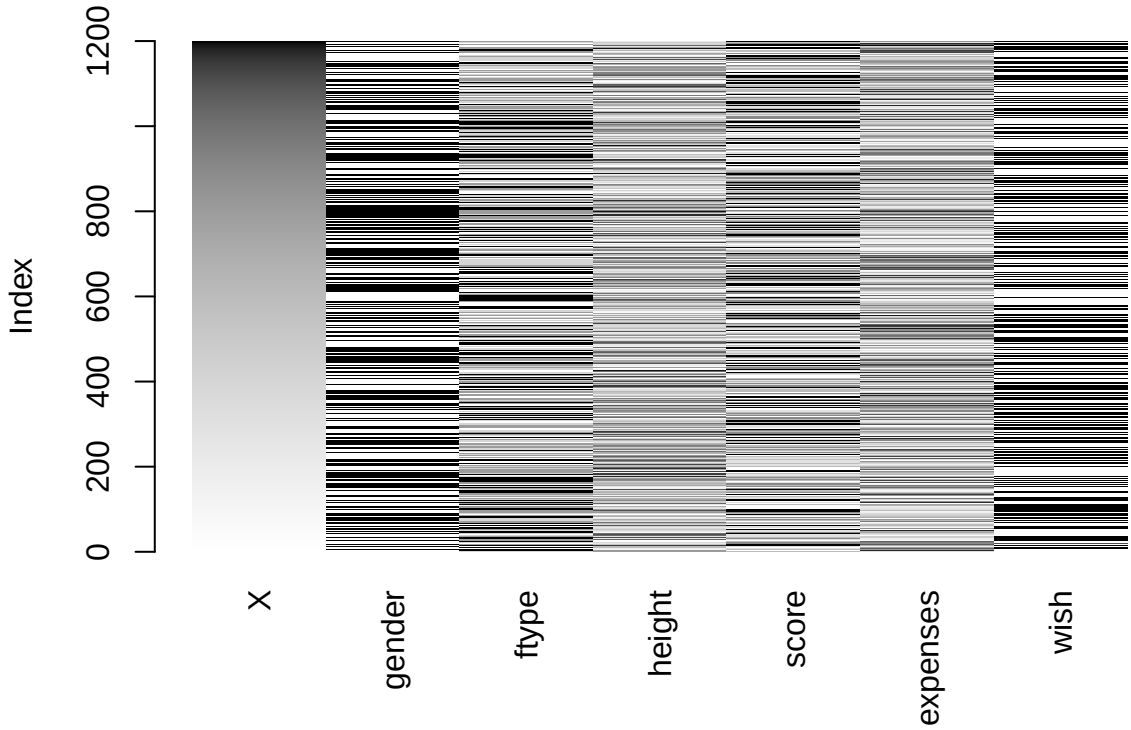
На графике слева показано, с какой частотой встречаются пропущенные значения в той или иной переменной. На графике справа показано, в каких комбинациях эти пропущенные значения встречаются.

```
# aggr - из библиотеки VIM
aggr(tree)
```



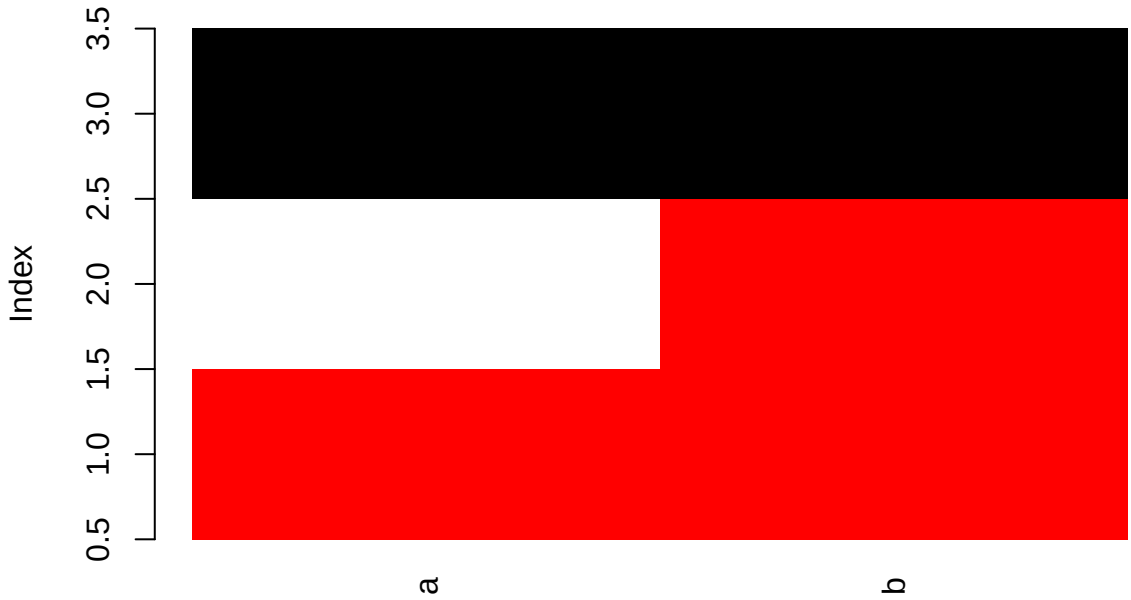
Следующий график отвечает за заполненность наблюдений (красным цветом отмечены пропущенные значения, остальное — заполненные значения, чем темнее цвет, тем больше значение). По вертикальной оси — номер строки в датафрейме, id наблюдения.

```
# matrixplot - из библиотеки mice
matrixplot(tree)
```



Так как в датафрейме `tree` всего две строчки с пропущенными значениями, и они не рядом, на графике их почти не видно. Но если пропусков много, этот график их покажет, сразу станет видно красные «дыры» на фоне серых и черных полосок. Для примера можем посмотреть на тот же график для `test`:

```
matrixplot(test)
```



Датафрейм маленький, и по графику сразу видно, что ячеек с пропущенными значениями много, если сравнивать с общим числом ячеек в датафрейме.

Описание качественных данных

Если нас интересует отдельный столбец датафрейма, его можно выбрать через `$`:

```
head(tree$wish) # первые несколько значений
```

```
## [1] да  нет да  да  нет да
## Levels: да нет
```

Выбрать, а дальше описывать отдельно. Если показатель качественный (текстовый или факторный), для него логично определить уникальные значения:

```
unique(tree$wish)
```

```
## [1] да   нет  <NA>
## Levels: да нет
```

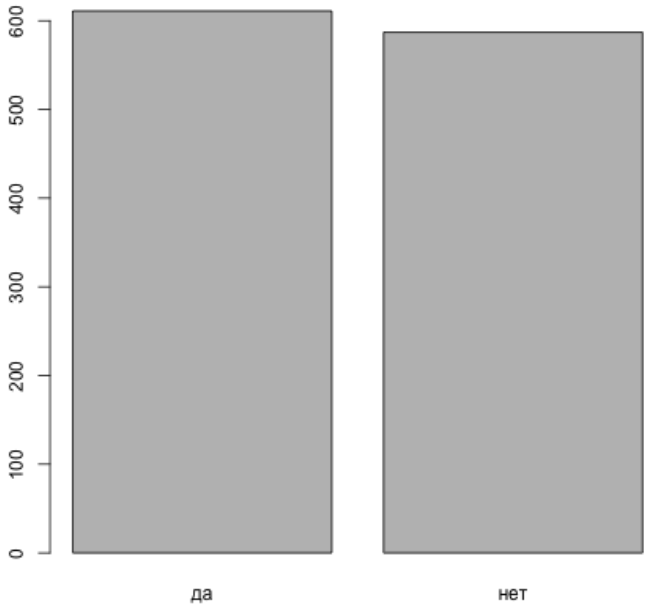
И соответствующие им частоты:

```
table(tree$wish)
```

```
##
##  да нет
## 611 587
```

Потом эту таблицу частот можно поместить внутрь функции `barplot()` и построить столбиковую диаграмму:

```
barplot(table(tree$wish))
```



Можем добавить цвета:

```
barplot(table(tree$wish),
        col = c("hotpink", "pink"))
```

График далёк от идеального: подписей нет, вертикальная ось коротковата... Но настройкой графиков мы будем заниматься позже, пока просто смотрим, что возможность быстро построить график есть.

Описание количественных данных

Уже знакомую нам функцию `summary()` мы можем применить и к отдельному столбцу (и к вектору вне датафрейма тоже):

```
summary(tree$expenses)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   302.0   904.8  1630.5  1629.0  2300.0  2999.0
```

Здесь уже всё знакомо.

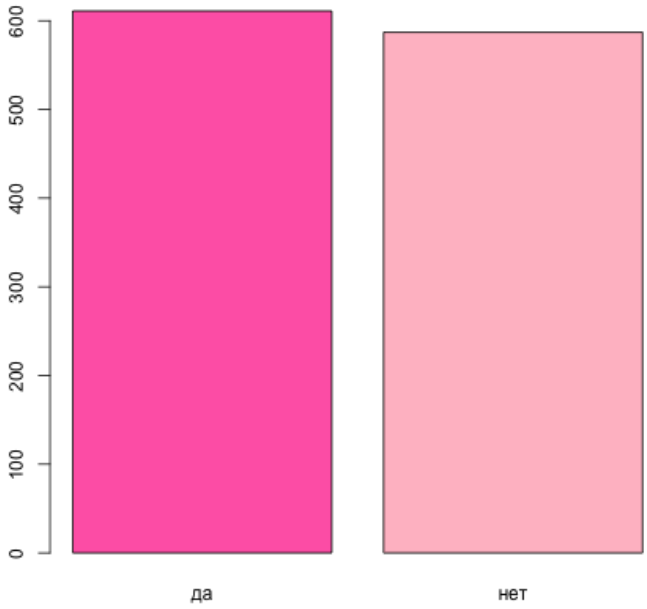
Теперь посмотрим на более подробную выдачу R с описательными статистиками. Чтобы это сделать, нам понадобится библиотека `psych`, которая содержит набор функций, часто используемых в психометрических исследованиях. Установим её:

```
install.packages("psych")
```

Обратимся к библиотеке через `library()`:

```
library(psych)
```

Теперь запросим описательные статистики для столбца `expenses` с помощью функции `describe()`:




```
describe(tree$expenses)
```

```
##    vars    n    mean     sd median trimmed     mad min  max range skew kurtosis
## X1    1 1200 1629.05 792.82 1630.5 1622.58 1028.18 302 2999  2697 0.06    -1.24
##       se
## X1 22.89
```

Что есть что?

- `vars`: число описываемых переменных (не путать с `var` для дисперсии);
- `n`: число наблюдений;
- `mean`: среднее арифметическое, выборочное среднее;
- `sd`: стандартное отклонение;
- `median`: медиана;
- `trimmed`: усечённое среднее, среднее по цензурированной выборке (см. ниже);
- `mad`: медианное значение абсолютного отклонения от медианы (нам не понадобится);
- `min`, `max`: минимальное и максимальное значение;
- `range`: размах;
- `skew`: коэффициент асимметрии или скошенности (см.нижк);
- `kurtosis`: коэффициент эксцесса (см. ниже);
- `se`: стандартная ошибка среднего;

Подробнее про некоторые статистики.

Усечённое среднее, среднее по цензурированной выборке

Считается так: выборка упорядочивается по возрастанию, из неё убирается 5% наблюдений слева и справа (наименьшие и наибольшие), потом по такой усечённой или цензурированной выборке считается обычное среднее арифметическое.

Наравне с медианой считается более устойчивой оценкой среднего, так как после усечения выборки

такой показатель уже несильно зависит от слишком больших или слишном маленьких (нетипичных) значений в выборке. То есть, при наличии нетипичных наблюдений в выборке (выбросов) такое среднее более адекватно отражает реальность, чем обычное среднее арифметическое.

Коэффициент асимметрии

Показатель принимает значения примерно от -3 до 3. Значение 0 соответствует симметричному распределению (например, нормальному, вспомните график плотности, симметричный относительно математического ожидания). Значения меньше 0 соответствуют распределению, которое скошено влево (длинный хвост «слева»), значения больше 0 соответствуют распределению, которое скошено вправо (длинный «хвост» справа).

В нашем случае распределение почти симметричное, коэффициент близок к нулю, но при это оно немного скошено вправо, поэтому значение больше 0.

Коэффициент эксцесса

Показатель принимает значения примерно от -3 до 3 и отвечает за выраженность пика распределения. Чем больше значение коэффициента, тем более выраженный пик. Стандартное нормальное распределение имеет коэффициент эксцесса равный 0. Отрицательные значения коэффициента соответствуют более «плоским» и «гладким» распределениям, у которых пик не такой заметный. Посмотрите на картинку здесь и сравните.

В нашем случае распределение несильно отличается от нормального, поэтому коэффициент близок к нулю.

Библиотека `psych` удобна тем, что она содержит функцию `describeBy()`, которая позволяет выводить описательные статистики по группам. Нет необходимости отфильтровывать нужные строки и сохранять их в отдельные датасеты, можно просто указать группирующую переменную. Например, сравним, сколько на хвойные деревья могут тратить мужчины и женщины:

```
describeBy(tree$expenses, tree$gender)
```

```
##
##  Descriptive statistics by group
## group: female
##    vars   n    mean     sd median trimmed     mad min  max range skew kurtosis
## X1    1 612 1640.38 785.23 1652.5  1635.9 1012.62 302 2999  2697 0.03    -1.25
##       se
## X1 31.74
## ------------------------------------------------------------
## group: male
##    vars   n    mean     sd median trimmed     mad min  max range skew kurtosis
## X1    1 588 1617.25 801.15   1597 1608.73 1052.65 302 2992  2690 0.09    -1.24
##       se
## X1 33.04
```

Очень удобно!

Если нас интересует только определённая характеристика столбца, можем воспользоваться базовыми, уже знакомыми нам, функциями.

```
min(tree$expenses)  # минимум
```

```
## [1] 302
```

```
max(tree$expenses)  # максимум
```

```
## [1] 2999
```

```r
mean(tree$expenses)  # среднее
```

```
## [1] 1629.045
```

```r
median(tree$expenses)  # медиана
```

```
## [1] 1630.5
```

```r
var(tree$expenses)  # дисперсия
```

```
## [1] 628562.6
```

```r
sd(tree$expenses)  # стандартное отклонение
```

```
## [1] 792.8194
```

Однако у всех этих функций есть одна особенность — они возвращают `NA`, если в столбце или векторе есть хотя бы одно пропущенное значение. Попробуем посчитать среднее для вектора с `NA`:

```r
mean(c(7, 5, NA, 9))
```

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

Нет ответа, плюс, получили предупреждение о наличие `NA`. Чтобы этого избежать, можно добавить опцию `na.rm = TRUE`, сокращение от *NA remove*:

```r
mean(c(7, 5, NA, 9), na.rm = TRUE)
```

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

Пропущенные значения не удаляются из самого вектора, но не учитываются при вычислении среднего. То же будет актуально и для других характеристик (минимум, медиана и прочие).

Напоследок построим гистограмму:

```r
hist(tree$expenses)
```

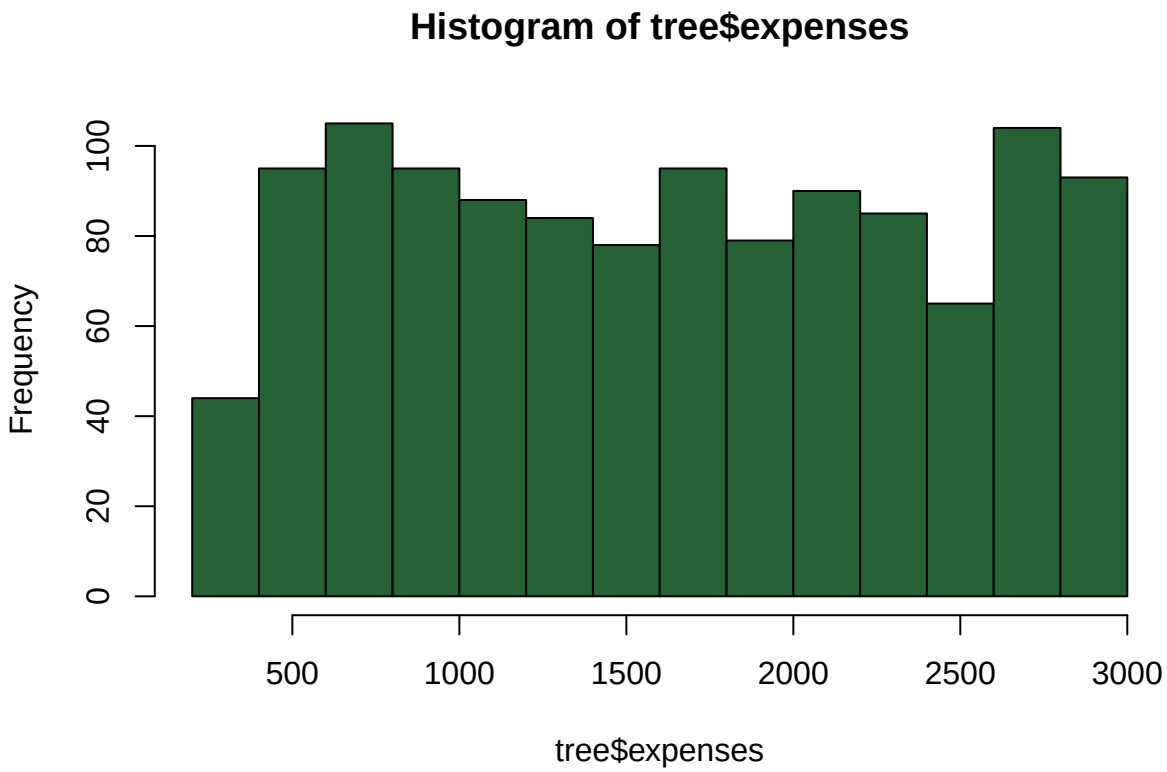
Добавим цвет:

```
hist(tree$expenses, col = "hotpink")
```

Выбор цветов в R богатый, список всех цветов с примерами можно посмотреть здесь. При желании можно вводить не название цвета, а его код в формате RGB или HEX. Пример с цветом в формате

HEX (*hexadecimal*):

```
hist(tree$expenses, col = "#266136")
```



Про форматы цветов можно посмотреть здесь.

Настройку графиков и наведение красоты мы обсудим позже.