

Дескриптивный анализ данных в R

2.7. Цель работы:

Научиться получать элементарные характеристики наборов данных. Усвоить способы импорта данных. Научиться трактовать некоторые элементарные способы графического представления наборов данных. Выполнить дескриптивный анализ данных.

2.8. Получение первичных элементарных характеристик о наборах данных

Векторизованные вычисления

Несмотря на то, что R похож на многие современные скриптовые языки программирования, например, такие, как Perl и Python, в нём есть много своеобразного. Одна из интересных и очень полезных особенностей R — это, так называемые, векторизованные вычисления. Использовать их очень просто. Допустим, мы хотим перевести вес из килограммов в граммы:

```
> w<-c(69,68,93,87,59,82,72)
> w
[1] 69 68 93 87 59 82 72
```

```
w*1000
[1] 69000 68000 93000 87000 59000 82000 72000
```

Конечно, циклы в R тоже будут работать (но не рекомендуются):

```
for(i in seq_along(w)){w[i]<-w[i]*1000}
```

Векторизованы и операции между векторами и матрицами. Помимо простых арифметических операций, векторизованы и более сложные преобразования. Есть целое семейство функций, которые специализируются на векторизованных вычислениях: *apply()*, *by()*, *lapply()*, *sapply()*, *tapply()* и другие, но о них чуть позже, а сейчас разберем примеры работы с векторными операциями.

Пример 2.2.1:

Программный код для создания таблицы данных:

1. Создадим таблицу данных о диабетиках:

```
> patientID<-c(1, 2, 3, 4)
> age<-c(25, 34, 28, 52)
> diabetes<-c("Type1", "Type2", "Type1", "Type1")
> status<-c("Poor", "Improved", "Excellent", "Poor")
> patientdata<-data.frame(patientID, age, diabetes, status)
> patientdata
  patientID age diabetes    status
1         1  25   Type1     Poor
2         2  34   Type2  Improved
3         3  28   Type1  Excellent
4         4  52   Type1     Poor
```

2. Извлекаем информацию только о возрасте:

```
> patientdata[1:2]
  patientID age
1         1  25
2         2  34
3         3  28
4         4  52
```

3. То же самое можно получить, используя номера колонок:

```
> patientdata[c("diabetes", "status")]
  diabetes status
1   Type1   Poor
2   Type2 Improved
3   Type1 Excellent
4   Type1   Poor
```

```
> patientdata$age #Обозначает переменную age (возраст) в таблице данных patientdata
[1] 25 34 28 52
```

Знак \$ используется, чтобы обозначить определенную переменную в таблице данных.

4. Создадим сводную таблицу типов диабета в зависимости от состояния больного, вы можете использовать следующий программный код:

```
> table(patientdata$diabetes, patientdata$status)
      Excellent Improved   Poor
Type1         1         0      2
Type2         0         1      0
```

Поскольку добавление *patientdata\$* перед названием каждой переменной может быстро надоесть, существуют команды для быстрого вызова переменной. Для упрощения программного кода можно использовать функции *attach()* или *detach()* или *with()*.

Attach, detach и with

Функция *attach()* добавляет указанную таблицу данных к пути поиска R. Когда указывается имя переменной, программа ищет эту переменную в таблицах данных, включенных в траекторию поиска. В качестве примера можно взять стандартную таблицу данных *patientdata*, используя следующий программный код, чтобы узнать основные статистики ее полей

Пример 2.2.1 (продолжение):

5. Узнаем основные статистики характеристик набора *patientdata*:

```
summary(patientdata)
  patientID      age      diabetes      status
Min.   :1.00  Min.   :25.00  Type1:3    Excellent:1
1st Qu.:1.75  1st Qu.:27.25  Type2:1    Improved :1
Median :2.50  Median :31.00                Poor      :2
Mean   :2.50  Mean   :33.75
3rd Qu.:3.25  3rd Qu.:38.50
Max.   :3.00  Max.   :52.00
```

функция *summary()* обрабатывает переменные по-разному: для непрерывной переменной *age* вычислены минимум (minimum, Min.), максимум (maximum, Max.), среднее (Mean) и квантили (first and third quartiles: 1st Qu., 3rd Qu.), а для категориальных переменных *diabetes* и *status* подсчитана частота встречаемости каждого значения.

6. Те же показатели можно получить по отдельности:

```
mean(patientdata$age)
[1] 33.75
```

Или так:

```
> attach(patientdata) # если эта переменная не является глобальной
```

```
> summary(age)
  Min. 1st Qu. Median   Mean 3rd Qu.   Max.
25.00 27.25 31.00 33.75 38.50 52.00
> detach(patientdata)
```

Важно не забыть сделать в конце `detach()`, потому что велика опасность запутаться в том, что Вы присоединили, а что - нет. Кроме того, если присоединённые переменные были как-то модифицированы, на самой таблице это не скажется. Это же можно сделать чуть по-другому:

```
with(patientdata, mean(age))  
[1] 33.75
```

Ограничение функции ***with()*** заключается в том, что она не действует за пределами фигурных скобок. Рассмотрим следующий пример (возьмем уже знакомый нам набор данных ***mtcars***, чтобы избежать глобальных переменных):

```
> with(mtcars, {stats<-summary(mpg); stats})  
  Min. 1st Qu.  Median    Mean 3rd Qu.    Max.  
 10.40  15.42   19.20   20.09  22.80   33.90  
>  
> stats  
Error: object 'stats' not found
```

Если требуется создать объекты, которые будут существовать вне конструкции `with()`, используйте специальный символ присвоения `<<-` вместо обычного (`<-`). Этот прием позволит сохранить созданный объект в рабочем пространстве вне конструкции `with()`. Это можно показать на примере такого программного кода:

```
> with(mtcars, {nokeepstats<-summary(mpg); keepstats<<-summary(mpg)})  
> nokeepstats  
Error: object 'nokeepstats' not found  
> keepstats  
  Min. 1st Qu.  Median    Mean 3rd Qu.    Max.  
 10.40  15.42   19.20   20.09  22.80   33.90
```

7. Оператор конвейера

Используйте оператор конвейера (`%>%`) библиотеки `tidyverse` для объединения нескольких функций в «конвейер» функций без промежуточных переменных, результат вычисления становится первым параметром для следующей функции:

```
library(tidyverse)  
data(mpg)  
mpg %>%  
  filter(cty > 21) %>%  
  head(3) %>%  
  print()  
#> # Tibble: 3 x 11  
#> manufacturer model displ year cyl trans drv cty hwy fl class  
#> <chr> <chr> <dbl> <int> <int> <chr> <chr> <int> <int> <chr> <chr>  
#> 1 chevrolet malibu 2.4 2008 4 auto~ f 22 30 r mids~  
#> 2 honda civic 1.6 1999 4 manu~ f 28 33 r subc~  
#> 3 honda civic 1.6 1999 4 auto~ f 24 32 r subc~
```

Использование конвейера намного чище и проще для чтения, нежели использование промежуточных временных переменных (результат будет тот же):

```
temp1 <- filter(mpg, cty > 21)  
temp2 <- head(temp1, 3)  
print(temp2)
```

8. Применение функции к каждой строке таблицы данных

Рассмотрим пример с действительными числами и применим функцию `mean` к каждому столбцу матрицы:

```
mat <- matrix(c(1, 3, 2, 5, 4, 6), 2, 3);  
colnames(mat) <- c("t1", "t2", "t3");  
mat  
t1 t2 t3  
[1,] 1 2 4  
[2,] 3 5 6
```

```
apply(mat, 2, mean) # Вычисляем среднее значение каждого столбца.
```

```
t1 t2 t3
```

```
2.0 3.5 5.0
```

В базовой версии R функция `apply()` предназначена для обработки матрицы или таблицы данных. Второй аргумент этой функции определяет направление:

- 1 – обработка строка за строкой;
- 2 – обработка столбец за столбцом.

В качестве функции может выступать любая стандартная или пользовательская функция.

Для полноты картины следует отметить, что задачи вычисления среднего и суммы по строкам настолько часто встречаются, что в R для этого предусмотрены специальные функции `rowSums`, `rowMeans`, `colSums` и `colMeans`.

Также функцию `apply()` можно применять для многомерных массивов:

```
> arr <- array(rnorm(2 * 2 * 10), c(2, 2, 10))
> apply(arr, c(1,2), mean)
```

Последний вызов можно заменить на более удобный для чтения вариант:

```
> rowMeans(arr, dim = 2)
```

9. Определим стандартное отклонение, дисперсию (ее квадрат) и, так называемый, межквартильный размах:

```
sd(age);var(age);IQR(age)
```

```
[1] 12.09339
```

```
[1] 146.25
```

```
[1] 11.25
```

10. Расширим данные по пациентам:

В наших данных по диабету всего 4 строки. Их достаточно просмотреть глазами, чтобы всё понять. А как понять за разумный промежуток времени, есть ли какие-то выдающиеся числа, в данных тысячного размера? Для этого есть графические функции. Самая простая — это, так называемый, **ящик-с-усами**, или **боксплот**. Для начала добавим к нашим данным ещё тысячу гипотетических пациентов с возрастом, случайно взятым из межквартильного размаха исходных данных:

```
> new.1000<-round(sample((median(age)-IQR(age)):
+   +(median(age)+IQR(age)),1000,replace=TRUE))
> age<-c(age,new.1000)
> patientdata<-data.frame(patientID, age, diabetes, status)
> view(patientdata)
> plot(patientdata$age, patientdata$status)
> plot(patientdata$status, patientdata$age)
```

В результате этих действий наша таблица дополнится 1000 пациентов, возрастом от 25 до 52 лет (цифры, изначально присутствовавшие в таблице). Теперь с помощью команды

```
> fix(patientdata)
```

добавьте произвольным образом несколько пациентов старше 52 лет – это нам понадобится позже.

Этот пример интересен ещё и тем, что в нём представлена техника получения случайных значений. Функция `sample()` способна выбирать случайным образом данные из выборки. В данном случае мы использовали `replace=TRUE` поскольку нам нужно было выбрать много чисел из гораздо меньшей выборки. Если писать на R имитацию карточных игр (а такие программы уже

написаны), то надо использовать `replace=FALSE`. потому что из колоды нельзя достать опять ту же самую карту. Кстати говоря, из того, что значения случайные, следует, что результаты последующих вычислений могут отличаться, если их воспроизвести ещё раз.

11. Проанализируем наши данные с помощью графика `boxplot`:

```
> boxplot(patientdata$age)
```

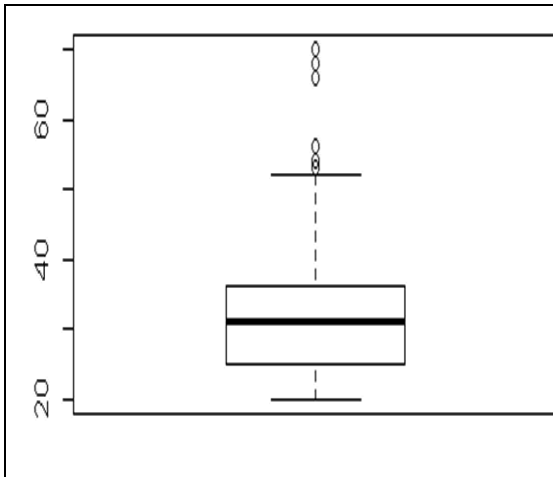


Рис 2_2.1. boxplot или ящик-с-усами.

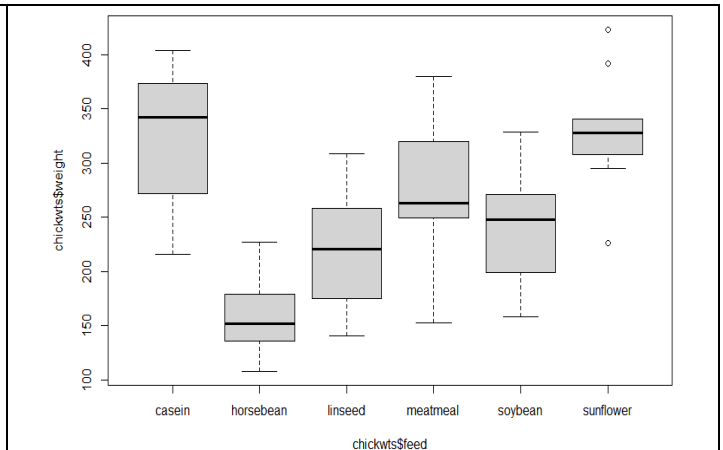


Рис. 2_2.2. Действие команды `boxplot` на встроенный объект `chickwts`

Как видно из рис. 2_2.1, несколько наших "старейшин" представлены высоко расположенными точками. Сам бокс, то есть главный прямоугольник, ограничен сверху и снизу квантилями, так что высота прямоугольника — это IQR. Это можно проверить:

```
summary(patientdata$age)
```

Min.	1st Qu.	Median	Mean	3rd Qu.	Max.
20.00	25.00	31.00	31.11	36.00	70.00

```
IQR(patientdata$age)
```

```
[1] 11
```

```
IQR= 3rd Qu.- 1st Qu.
```

Так называемые «усы» по умолчанию обозначают точки, удаленные на полтора IQR. Линия посередине прямоугольника — это, как легко догадаться, медиана. Точки, лежащие вне усов, рассматриваются как выбросы, и поэтому рисуются отдельно.

Боксплоты были специально придуманы известным статистиком Дж. Тьюки (John \V. Tükey - интересно, что именно Джон Тьюки первый в 1958г. применил слово `software` по отношению к программному обеспечению.) для того, чтобы быстро, эффективно и устойчиво отражать основные характеристики выборки. R использует оригинальные боксплоты Тьюки, а кроме того, может рисовать несколько боксплотов сразу, то есть эта команда векторизована (см рис. 2_2.2.):

```
> boxplot(chickwts$weight ~ chickwts$feed)
```

```
> library(help = "datasets")#справка по датасетам
```

На рис. 2_2.2. ящик-с-усами для примера стандартного датасета `chickwts`, который содержит числовую переменную `weight`, определяющую вес цыпленка и `feed` – фактор, определяющий тип корма. Боксплот (рис 2_2.2.) помогает нам понять, каким кормом лучше кормить птицу для получения максимального прироста по весу. Ниже показан фрагмент этого набора данных.

```
head(chickwts)
```

	weight	feed
1	179	horsebean
2	160	horsebean
3	136	horsebean
...		
71	332	casein

Интересно, что, если назначить коробчатую диаграмму переменной, можно вернуть список с различными компонентами. Создайте коробчатую диаграмму с *chickwts* набором данных и сохраните ее в переменной:

```
> res<-boxplot(chickwts$weight ~ chickwts$feed)
> res
$stats
      [,1] [,2] [,3] [,4] [,5] [,6]
[1,] 216.0 108.0 141.0 153.0 158 295.0
[2,] 271.5 136.0 175.0 249.5 199 307.5
[3,] 342.0 151.5 221.0 263.0 248 328.0
[4,] 373.5 179.0 258.5 320.0 271 340.5
[5,] 404.0 227.0 309.0 380.0 329 341.0

$n
[1] 12 10 12 11 14 12

$conf
      [,1] [,2] [,3] [,4] [,5] [,6]
[1,] 295.4771 130.0155 182.9151 229.4147 217.5964 312.9485
[2,] 388.5229 172.9845 259.0849 296.5853 278.4036 343.0515

$out
[1] 423 392 226

$group
[1] 6 6 6

$names
[1] "casein" "horsebean" "linseed" "meatmeal" "soybean" "sunflower"
```

Вывод будет содержать шесть элементов, описанных ниже:

- **статистика**: каждый столбец представляет нижний ус, первый квартиль, медиану, третий квартиль и верхний ус каждой группы.
- **n**: количество наблюдений каждой группы.
- **conf**: каждый столбец представляет нижний и верхний крайние значения доверительного интервала медианы.
- **out**: общее количество выбросов.
- **группа**: общее количество групп.
- **имена**: имена каждой группы.

Стоит упомянуть, что вы можете создать коробчатую диаграмму из только что созданной переменной (*res*) с помощью *bxp* функции.

```
>bxp(res)
```

Одним из ограничений коробчатых диаграмм является то, что они **не предназначены для обнаружения мультимодальности**. По этой причине также рекомендуется строить коробчатую диаграмму в сочетании с гистограммой или линией плотности.

По умолчанию **коробчатые диаграммы будут построены с указанием порядка факторов в данных**. Однако вы можете переупорядочить или отсортировать коробчатую диаграмму в R, переупорядочив данные по любой метрике, такой как медиана или среднее значение, с помощью *reorder* функции.

```
par(mfrow = c(1, 2))
# Lower to higher
medians <- reorder(chickwts$feed, chickwts$weight, median)
# medians <- with(chickwts, reorder(feed, weight, median)) # Equivalent
boxplot(chickwts$weight ~ medians, las = 2, xlab = "", ylab = "")
```

```
# Higher to lower
medians <- reorder(chickwts$feed, -chickwts$weight, median)
# medians <- with(chickwts, reorder(feed, -weight, median)) # Equivalent

boxplot(chickwts$weight ~ medians, las = 2, xlab = "", ylab = "")

par(mfrow = c(1, 1))
```

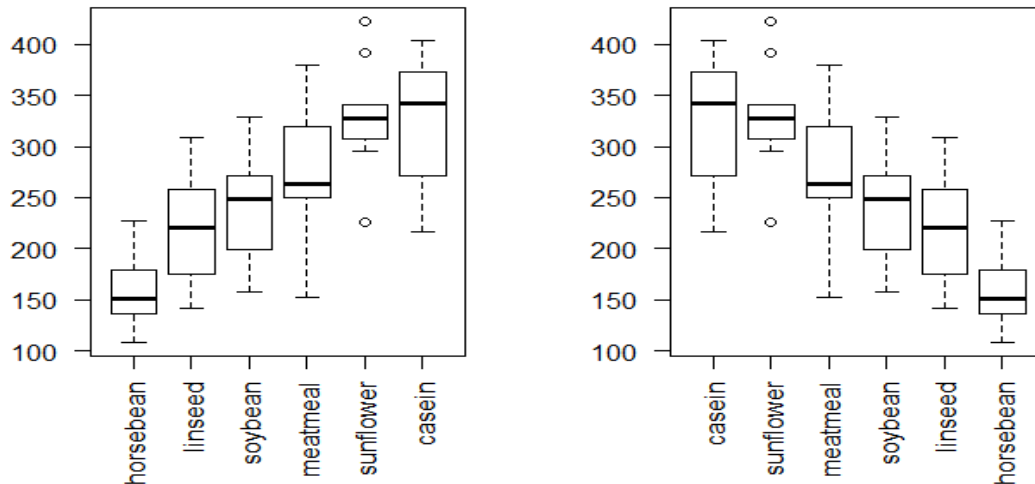


Рис. 2_2.3. Упорядочивание элементов boxplot из chickwts по медиане

Боксплоты можно разукрасить:

```
boxplot(chickwts$weight ~ medians, las = 2, xlab = "", ylab = "", col = rainbow(6))
```

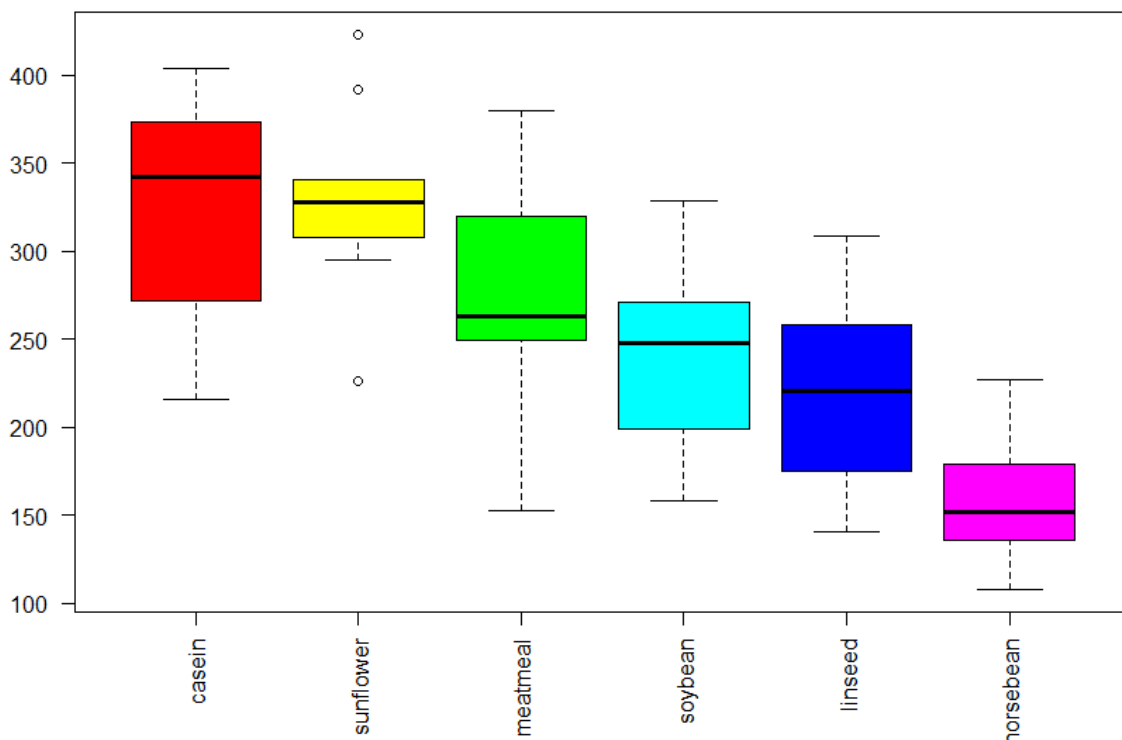


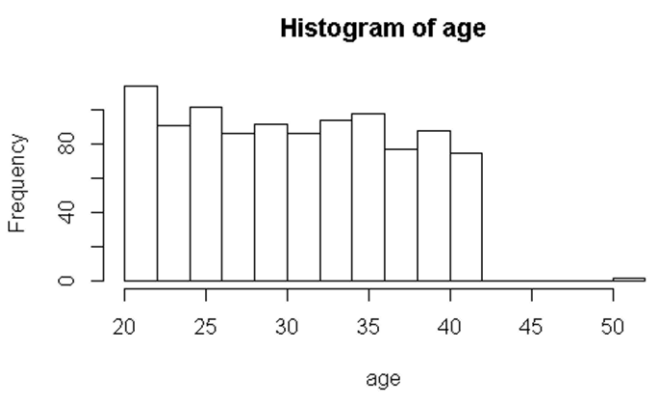
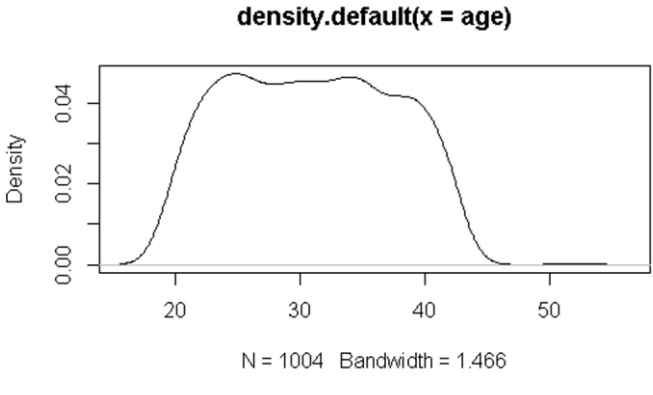
Рис. 2_2.4. Упорядочивание элементов boxplot из chickwts и их раскраска

12. Давайте построим гистограмму, которая отображает частоту появления значений выборки

Этот способ графического изображения выборки тоже очень популярен (см. рис. 2_2.4.). Это гистограммы, то есть столбики, высота которых соответствует частоте встречаемости данных, попавших в определенный диапазон:

`hist(age,breaks=20)`

По умолчанию команда **hist()** разбивает переменную на 10 интервалов, но их количество можно указать и вручную, как в предложенном примере.

 <p>Histogram of age</p> <p>Frequency</p> <p>age</p>	 <p>density.default(x = age)</p> <p>Density</p> <p>N = 1004 Bandwidth = 1.466</p>
<p><code>hist(age,breaks=20)</code></p>	<p><code>plot(density(age))</code> либо <code>plot(density(age,adjust=2)); rug(age)</code> – так будет еще более сглажено (зависит от adjust)</p>
<p>Рис 2_2.5. Гистограмма</p>	<p>Рис. 2_2.6. Сглаженная гистограмма (результат действий команд density)</p>

Численным аналогом гистограммы является функция `cut()`. При помощи этой функции можно выяснить, сколько данных, какого типа у нас имеется:

`table(cut(age,breaks=20))`

(20,21.6]	(21.6,23.2]	(23.2,23.8]	(23.8,26.4]	(26.4,28]	(28,29.6]
70	94	41	102	86	45
(29.6,31.2]	(31.2,32.8]	(32.8,33.4]	(33.4,36]	(36,37.6]	(37.6,39.2]
94	39	94	98	32	92
(39.2,40.8]	(40.8,42.4]	(42.4,44]	(44,45.6]	(45.6,47.2]	(47.2,48.8]
41	75	0	0	0	0
(48.8,50.4]	(50.4,52]				
0	1				

13. Еще раз о функции `ordered`:

При работе с векторами, которые представлены порядковыми данными, для функции `factor()` нужно добавлять параметр `ordered=TRUE` (вернемся к датасету `patientdata`). Примененная к вектору

`status <- c("Poor", "Improved", "Excellent", "Poor")`

команда `status <- factor(status, ordered=TRUE)` преобразует этот вектор в вид (3, 2, 1, 3) и установит внутреннее соответствие как 1=Excellent, 2=Improved, 3=Poor. Во время любой обработки этого вектора он будет воспринят как порядковая переменная с применением соответствующих статистических методов.

По умолчанию уровни фактора присваиваются значениям вектора в алфавитном порядке. Это сработало для фактора `status`, поскольку порядок "Excellent", "Improved", "Poor" имеет смысл. Если бы вместо "Poor" стояло "Ailing" (чахнувший), то возникло бы затруднение, поскольку тогда порядок был бы такой: "Ailing", "Excellent", "Improved".

Сходная проблема возникла бы, если бы нам был нужен такой порядок: "Poor", "Improved", "Excellent". Для упорядоченных факторов редко подходит алфавитный порядок уровней, предлагающийся по умолчанию.

Установку по умолчанию можно изменить при помощи параметра `levels`. Например,

`status <- factor(status, order=TRUE, levels=c("Poor", "Improved", "Excellent"))`

присвоит уровни значениям вектора следующим образом: 1=Poor, 2=Improved, 3=Excellent. Проверьте, что все присвоенные уровни соответствуют реальным значениям данных.

2.9. Объединение наборов данных

Если ваши данные существуют в виде разрозненных фрагментов, их нужно объединить прежде, чем двигаться дальше. В этом разделе рассказано, как добавлять столбцы (переменные) и строки (наблюдения) к таблице данных.

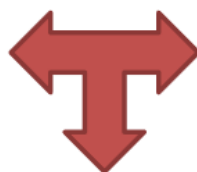
2.9.1. Добавление столбцов

mydata1

	country	year	y	y_bin	x1	x2	x3
1	A	2000	1343	1	0.28	-1.11	0.28
2	A	2001	-1900	0	0.32	-0.95	0.49
3	A	2002	-11	0	0.36	-0.79	0.7
4	A	2003	2646	1	0.25	-0.89	-0.09
5	B	2000	-5935	0	-0.08	1.43	0.02
6	B	2001	-712	0	0.11	1.65	0.26
7	B	2002	-1933	0	0.35	1.59	-0.23
8	B	2003	3073	1	0.73	1.69	0.26
9	C	2000	-1292	0	1.31	-1.29	0.2
10	C	2001	-3416	0	1.18	-1.34	0.28
11	C	2002	-356	0	1.26	-1.26	0.37
12	C	2003	1225	1	1.42	-1.31	-0.38

mydata2

	country	year	x4	x5	x6
1	A	2000	10	1	9
2	A	2001	7	1	9
3	A	2002	7	9	4
4	A	2003	1	2	3
5	B	2000	0	5	6
6	B	2001	5	8	5
7	B	2002	9	4	5
8	B	2003	1	5	1
9	C	2000	4	5	4
10	C	2001	6	9	6
11	C	2002	6	5	3
12	C	2003	7	3	3



```
mydata <- merge(mydata1, mydata2, by=c("country","year"))
```

	country	year	y	y_bin	x1	x2	x3	x4	x5	x6
1	A	2000	1343	1	0.28	-1.11	0.28	10	1	9
2	A	2001	-1900	0	0.32	-0.95	0.49	7	1	9
3	A	2002	-11	0	0.36	-0.79	0.7	7	9	4
4	A	2003	2646	1	0.25	-0.89	-0.09	1	2	3
5	B	2000	-5935	0	-0.08	1.43	0.02	0	5	6
6	B	2001	-712	0	0.11	1.65	0.26	5	8	5
7	B	2002	-1933	0	0.35	1.59	-0.23	9	4	5
8	B	2003	3073	1	0.73	1.69	0.26	1	5	1
9	C	2000	-1292	0	1.31	-1.29	0.2	4	5	4
10	C	2001	-3416	0	1.18	-1.34	0.28	6	9	6
11	C	2002	-356	0	1.26	-1.26	0.37	6	5	3
12	C	2003	1225	1	1.42	-1.31	-0.38	7	3	3

Для слияния двух таблиц нужно использовать функцию `merge()`. В большинстве случаев две таблицы объединяются по значениям одной или нескольких ключевых переменных. К примеру, команда

```
total <- merge(dataframeA, dataframeB, by="ID")
```

сводит таблицы данных `dataframeA` и `dataframeB` по значениям переменной `ID`. Аналогично команда

```
total <- merge(dataframeA, dataframeB, by=c("ID","Country"))
```

сводит две таблицы данных по значениям переменных `ID` и `Country`. Подобное сведение таблиц данных в горизонтальном направлении часто используется для добавления переменных к таблице.

Примечание. Если вы хотите просто объединить две матрицы или таблицы данных в горизонтальном направлении и вам не нужно указывать значения переменной, по которым произойдет объединение, можете использовать функцию `cbind()`:

```
total <- cbind(A, B)
```

Эта команда объединяет объекты `A` и `B` в горизонтальном направлении. Для того чтобы функция работала правильно, каждый объект должен иметь одинаковое число строк, расположенных в одинаковом порядке.

```
> m1 <- matrix(1:10, ncol = 2)
> colnames(m1) <- letters[1:2]
```

```
> m2 <- matrix(11:20, ncol = 2)
> colnames(m2) <- letters[3:4]
> M <- cbind(m1, m2); M
```

	a	b	c	d
[1,]	1	6	11	16
[2,]	2	7	12	17
[3,]	3	8	13	18
[4,]	4	9	14	19
[5,]	5	10	15	20

2.9.2. Добавление строк

Для объединения двух таблиц данных в вертикальном направлении используйте функцию `rbind()`:

```
total <- rbind(dataframeA, dataframeB)
```

Объединяемые таблицы должны содержать одинаковые атрибуты (имена столбцов), однако им не обязательно быть расположенными в одной и той же последовательности. Если в таблице `dataframeA` есть атрибуты, которых нет в таблице `dataframeB`, тогда перед объединением этих таблиц нужно сделать одно из двух:

- удалить лишние атрибуты из таблицы `dataframeA`;
- создать дополнительные атрибуты в таблице `dataframeB` и присвоить им значения `NA` (пропущенные).

Слияние таблиц в вертикальном направлении обычно используется для добавления наблюдений в таблицу данных.

2.10. Разделение наборов данных на составляющие

В R имеются обширные возможности для извлечения отдельных частей объектов. Эти возможности можно использовать для выбора и исключения переменных и/или наблюдений. В приведенных ниже разделах обсуждаются несколько способов выбора или удаления переменных и наблюдений.

Для примера составим такой набор данных:

```
manager <- c(1, 2, 3, 4, 5)
> date <- c("10/24/08", "10/28/08", "10/1/08", "10/12/08", "5/1/09")
> country <- c("US", "US", "UK", "UK", "UK")
> gender <- c("M", "F", "F", "M", "F")
> age <- c(32, 45, 25, 39, 99)
> q1 <- c(5, 3, 3, 3, 2)
> q2 <- c(4, 5, 5, 3, 2)
> q3 <- c(5, 2, 5, 4, 1)
> q4 <- c(5, 5, 5, NA, 2)
> q5 <- c(5, 5, 2, NA, 1)
> leadership <- data.frame(manager, date, country, gender, age, +
q1, q2, q3, q4, q5, stringsAsFactors=FALSE)
leadership
```

	manager	date	country	gender	age	q1	q2	q3	q4	q5
1	1	10/24/08	US	M	32	5	4	5	5	5
2	2	10/28/08	US	F	45	3	5	2	5	5
3	3	10/01/08	UK	F	25	3	5	5	5	2
4	4	10/12/08	UK	M	39	3	3	4	NA	NA
5	5	05/01/09	UK	F	99	2	2	1	2	1

Суть задачи в следующем:

Как мужчины и женщины различаются по стилю руководства организациями?

- различаются ли мужчины и женщины на руководящих должностях по степени лояльности к вышестоящему начальству?
- зависит ли это от страны, или выявленные гендерные различия носят универсальный характер?

Один из способов ответить на эти вопросы – взять начальников из разных стран и ранжировать подчиненных им менеджеров по степени лояльности, используя вопросы вроде этого: этот менеджер спрашивает мое мнение перед принятием кадровых решений.

- 1 – абсолютно не согласен;
- 2 – не согласен;
- 3 – бывает по-разному;
- 4 – согласен;
- 5 – полностью согласен.

В результате можно получить данные вроде тех, что представлены в табл. leadership
Каждая строка – это оценка, которую дал менеджеру его или ее начальник.

Manager (менеджер)	Date (дата)	Country (страна)	Gender (пол)	Age (возраст)	q1	q2	q3	q4	q5
1	10/24/08	US	M	32	5	4	5	5	5
2	10/28/08	US	F	45	3	5	2	5	5
3	10/01/08	US	F	25	3	5	5	5	2
4	10/12/08	US	M	39	3	3	4		
5	05/01/09	US	F	99	2	2	1	2	1

2.10.1. Выбор переменных

Часто бывает так, что новый набор данных создается из небольшого числа переменных, выбранных из большего набора данных. К примеру, команда

```
newdata <- leadership[,c(6:10)]
> newdata
  q1 q2 q3 q4 q5
1  5  4  5  5  5
2  3  5  2  5  5
3  3  5  5  5  2
4  3  3  4 NA NA
5  2  2  1  2  1
```

позволяет выбрать переменные q1, q2, q3, q4 и q5 из таблицы данных leadership и сохранить их в таблице данных newdata. Не указывая номера строк (.), мы по умолчанию выбираем все строки.

При помощи команд

```
> myvars <- c("q1", "q2", "q3", "q4", "q5")
> newdata <- leadership[myvars]
```

можно выбрать те же самые переменные. В этом случае имена переменных (в кавычках) используются для обозначения подлежащих извлечению переменных.

Наконец, для выполнения той же задачи вы могли бы использовать команды

```
myvars <- paste("q", 1:5, sep="")
newdata <- leadership[myvars]
```

2.10.2. Исключение переменных

Существует много причин для того, чтобы исключить переменные. Например, в том случае, если переменная содержит несколько пропущенных значений, вам может понадобиться удалить ее до начала анализа данных. Давайте рассмотрим несколько способов удаления переменных.

Переменные q3 и q4 можно удалить при помощи следующих команд:

```
> myvars <- names(leadership) %in% c("q3", "q4")
> newdata <- leadership[!myvars]
```

Для того чтобы понять, как это работает, рассмотрим команды по частям:

1. names(leadership) создает текстовый вектор, содержащий названия переменных: c("managerID", "testDate", "country", "gender", "age", "q1", "q2", "q3", "q4", "q5").
2. myvars <- names(leadership) %in% c("q3", "q4") возвращает логический вектор со значениями TRUE для каждого элемента вектора names(leadership), который

соответствует q3 или q4, и со значениями FALSE в противном случае: c(FALSE, FALSE, FALSE, FALSE, FALSE, FALSE, FALSE, TRUE, TRUE, FALSE).

3. Оператор «не» (!) изменяет логические значения на противоположные: c(TRUE, TRUE, TRUE, TRUE, TRUE, TRUE, FALSE, FALSE, TRUE).
4. leadership[c(TRUE, TRUE, TRUE, TRUE, TRUE, TRUE, TRUE, FALSE, FALSE, TRUE)] выбирает столбцы, для которых значения логического вектора равны TRUE, так что q3 и q4 исключаются.

Зная, что q3 и q4 – это восьмая и девятая переменные, вы можете удалить их при помощи команды

```
newdata <- leadership[c(-8,-9)]
```

Это работает, поскольку минус перед номером столбца означает, что этот столбец должен быть удален.

Наконец, эти же два столбца можно удалить при помощи команды

```
leadership$q3 <- leadership$q4 <- NULL
```

Теперь вы назначаете эти столбцы неопределенными (NULL). Обратите внимание, что NULL – это не то же самое, что NA (отсутствующие значения).

Удаление переменных – действие, противоположное отбору переменных. Выбор между этими действиями зависит от того, какое из них легче осуществить. Если нужно удалить много переменных, может быть, проще выбрать остающиеся, и наоборот.

2.10.3. Выбор наблюдений

Выбор или удаление наблюдений (строк) — это в большинстве случаев залог успешной подготовки данных и их анализа. Несколько примеров содержатся в приведенном ниже программном коде.

Программный код 3.6. Выбор наблюдений

```
newdata <- leadership[which(gender=="M" & age > 30),]  
> detach(leadership)  
> newdata <- leadership[1:3,]  
> newdata <- leadership[which(leadership$gender=="M" & leadership$age > 30),]  
> newdata <- leadership[which(gender=="M" & age > 30),]
```

В каждом из этих примеров приведены номера строк, а место для номеров столбцов оставлено пустым (то есть выбраны все столбцы). В первом примере в наборе **newdata** выбираем строки с первой по третью (первые три наблюдения).

Во втором примере выбираем всех мужчин старше 30 лет. Давайте рассмотрим эту строку программного кода по частям, чтобы понять его.

- Логическое выражение leadership\$gender=="M" создает вектор c(TRUE, FALSE, FALSE, TRUE, FALSE).
- Логическое выражение leadership\$age > 30 создает вектор c(TRUE, TRUE, FALSE, TRUE, TRUE).
- Логическое выражение C(TRUE, FALSE, FALSE, TRUE, FALSE) & C(TRUE, TRUE, FALSE, TRUE, TRUE) создает вектор c(TRUE, FALSE, FALSE, TRUE, FALSE).
- Функция which () возвращает номера элементов вектора, которые представлены значением TRUE. Таким образом, выражение which (c (TRUE, FALSE, FALSE, TRUE, FALSE)) возвращает вектор C (1, 4).
- Команда leadership [c (1, 4),] выбирает из таблицы данных первое и четвертое наблюдения, которые удовлетворяют нашим критериям (мужчины старше 30 лет).

В третьем примере использована функция attach(), чтобы нам не нужно было писать перед каждым именем переменной название таблицы данных.

При анализе данных вы можете захотеть ограничиться наблюдениями, сделанными в период между 1 января и 31 декабря 2009 года. Как это можно осуществить? Вот одно из возможных решений:

```
leadership$date <- as.Date(leadership$date, "%m/%d/%y")  
startdate <- as.Date("2009-01-01")  
enddate <- as.Date("2009-10-31")
```

```
newdata <- leadership[which(leadership$date >= startdate &
leadership$date <= enddate),]
```

Преобразуйте даты, которые исходно воспринимались программой как текстовые значения, в формат даты (мм/дд/гг). Затем назначьте начальную и конечную даты временного отрезка. Поскольку по умолчанию функция `as.Date()` уже создает даты в формате гггг/мм/дд, вам не нужно указывать формат отдельно. Наконец, выберите наблюдения, которые удовлетворяют заданному критерию, как мы делали в предыдущем примере.

2.10.4. Функция `subset()`

Примеры, приведенные в предыдущих двух разделах, важны, поскольку они помогают понять, как R интерпретирует логические векторы и операторы сравнения. Понимание принципа работы этих примеров поможет понять общие принципы действия программного кода в R. Теперь, после того как вы освоили сложные способы, посмотрим, как сделать это проще.

Функция `subset()` – возможно, самый простой способ выбора переменных и наблюдений. Вот два примера:

```
newdata <- subset(leadership, age >= 35 | age < 24, select=c(q1, q2, q3, q4))
newdata <- subset(leadership, gender=="М" & age > 25, select=gender:q4)
```

В первом примере вы выбираете все строки, в которых значение переменной `age` больше или равно 35 или меньше 24, оставляя переменные с `q1` по `q3`. Во втором примере вы отбираете всех мужчин старше 25 лет, оставляя переменные с `gender` по `g4` (`gender`, `g4` и все столбцы, находящиеся между ними). Вы уже видели оператор «двоеточие» в выражениях типа от:до. Здесь этот оператор позволяет оставить все переменные в таблице данных, начиная с переменной **от** и заканчивая переменной **до**.

2.11. Способы загрузки (импорта) данных

В R реализованы разные способы импорта данных. Исчерпывающее руководство по импорту данных в R доступно в Интернете по адресу: <http://cran.r-project.org/doc/manuals/R-data.pdf>.

Как видно на рис. 2_2.7, в R можно вводить данные с клавиатуры, импортировать из текстовых файлов, из Microsoft Excel и Access, из распространенных статистических программ, специализированных форматов, а также из разных систем управления базами данных. Поскольку никогда нельзя угадать, откуда вы получите данные, мы рассмотрим здесь все источники по порядку. Вам нужно прочесть только про те, которые вы собираетесь использовать.

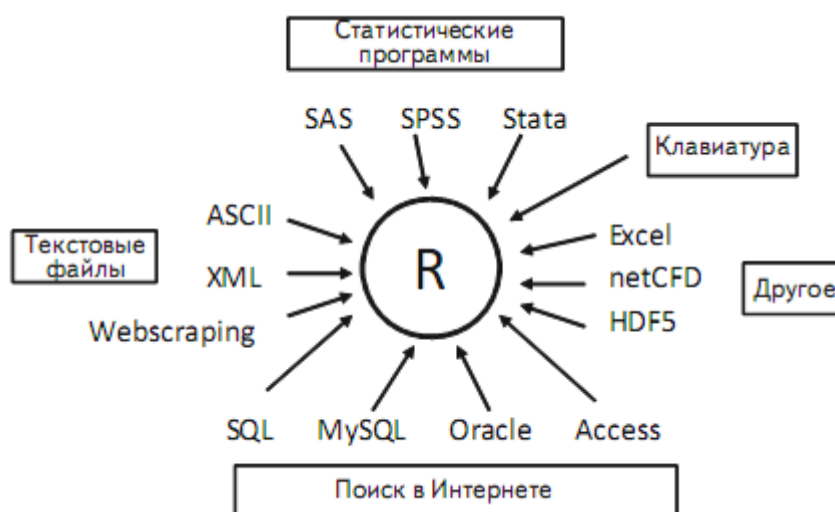


Рис. 2_2.7. Источники, из которых можно импортировать данные в R

2.11.1. Редактирование

Введите команду: `patientdata=edit(patientdata)` и отредактируйте таблицу при помощи встроенного текстового редактора под Windows. Сокращенная версия команды `mydata <- edit(mydata)` – это просто `fix(mydata)`.

Этот метод хорошо работает для небольших объемов данных.

2.11.2. Импорт данных из текстового файла с разделителями

Импорт данных из текстовых файлов с разделителями возможен при помощи команд `read.table()` и `read.csv()`, функции, которая сохраняет данные в виде таблицы. Вот пример использования функции:

```
mydataframe <- read.table(file, header="logical", sep=" ", row.names="names")
carSpeeds <- read.csv(file = 'data/car-speeds.csv')
```

где *file* – это ASCII файл с разделителями, *header* – это логическое значение, определяющее, содержит ли первая строка названия переменных (TRUE – да, FALSE – нет), *sep* указывает, каким символом разделены элементы данных, а *row.names* – необязательный параметр, для указания столбца (столбцов), в котором содержатся названия строк.

Например, программный код

```
grades <- read.table("studentgrades.csv", header=TRUE, sep=",",
row.names="STUDENTID")
```

позволяет прочесть файл с разделителями-запятыми, который называется *studentgrades.csv*, из текущей рабочей директории и сохранить его в виде таблицы данных с названием *grades*. В этом файле названия переменных содержались в первой строке, а названия строк – в столбце с названием *STUDENTID*.

Обратите внимание на то, что использование параметра *sep* позволяет импортировать файлы с любыми символами в качестве разделителей. Файлы с символом табуляции в качестве разделителя можно импортировать, указав `sep="\t"`. По умолчанию используется `sep=" "`, обозначающий один или несколько пробелов, символов табуляции, символов новой строки или возврата каретки.

По умолчанию текстовые переменные преобразуются в факторы. Это не всегда уместно (например, для переменной, которая содержит комментарии респондента). Такое преобразование можно заблокировать разными способами. Добавление параметра `stringsAsFactors=FALSE` не позволит преобразовывать в факторы все текстовые переменные. В качестве альтернативы можно использовать параметр `colClasses` для того, чтобы указать формат (например, логический, числовой, текстовый, фактор) каждого столбца. У функции `read.table()` есть много дополнительных параметров, при помощи которых можно контролировать импорт данных. Подробнее об этом можно прочесть, выполнив команду `help(read.table)`.

В R также реализовано несколько алгоритмов получения данных из Интернета. Вместо имени файла можно использовать функции `file()`, `gzfile()`, `bzfile()`, `xzfile()`, `unz()` и `url()`.

Функция `file()` позволяет получать доступ к файлам, буферу обмена и к стандартным потокам ввода-вывода. Функции `gzfile()`, `bzfile()`, `xzfile()` и `unz()` дают возможность доступа к сжатым файлам. Функция `url()` предоставляет доступ к файлам в Интернете через полный URL-адрес, который включает `http://`, `ftp://` или `file://`. Для HTTP или FTP можно назначить программы-посредники (проxy). Полные URL-адреса (заключенные в прямые кавычки) можно использовать также вместо названий файлов. Более подробную информацию можно получить, введя команду `help(file)`.

2.11.3. Импорт данных из Excel

Лучший способ прочесть файл в формате Excel – это сохранить его в формате текстового файла с разделителями и импортировать в R, как это описано выше. Под Windows для доступа к файлам Excel также можно использовать пакет RODBC. В первой строке электронной таблицы

должны содержаться названия переменных (столбцов). Прежде всего скачайте и установите пакет RODBC.

```
install.packages("RODBC")
```

Теперь вы можете использовать следующий программный код для импорта данных:

```
library(RODBC)
channel <- odbcConnectExcel("myfile.xls")
mydataframe <- sqlFetch(channel, "mysheet")
odbcClose(channel)
```

Здесь myfile.xls – это файл Excel, mysheet – это название нужного листа из рабочей книги Excel, channel – это вспомогательный объект RODBC, созданный функцией odbcConnectExcel(), и mydataframe – это получившаяся таблица данных. Этот пакет можно также использовать для импорта данных из Microsoft Access. Подробности изложены в файле справки: help(RODBC).

В Excel 2007 используются файлы формата XLSX, которые фактически представляют собой сжатый набор XML-файлов. Для импорта электронных таблиц в этом формате можно использовать пакет xlsx. Убедитесь, что перед первым использованием вы скачали и установили этот пакет. Функция read.xlsx() осуществляет импорт нужного листа XLSX-файла в таблицу данных. Проще всего использовать эту функцию по такой схеме:

```
read.xlsx(file, n),
```

где file – это путь к файлу книги Excel 2007, а n – число листов, которые нужно импортировать. Например, под Windows программный код

```
library(xlsx)
workbook <- "c:/myworkbook.xlsx"
mydataframe <- read.xlsx(workbook, 1)
```

импортирует первый лист книги myworkbook.xlsx, хранящейся на диске C:, и сохраняет его в виде таблицы данных mydataframe. Пакет xlsx может не только импортировать листы XLSX-файлов. Он также может создавать файлы этого формата и управлять ими. Программистам, перед которыми стоит задача разработать интерфейс для обмена данными между R и Excel, следует обратить внимание на этот относительно новый пакет.

2.11.4. Другие возможности импорта данных

Кроме перечисленных возможностей есть и другие:

- Извлечение данных из веб-страниц;
- Импорт данных из XML-файлов
- Импорт данных из
 - SPSS
 - SAS
 - Stata
- Импорт данных из netCDF:
 - Данные в формате netCDF (network Common Data Form – сетевая общедоступная форма данных) создаются в программе Unidata с открытым программным кодом.
- Импорт данных из HDF5
 - HDF5 (Hierarchical Data Format – иерархический формат данных) – это программная технология, которая позволяет управлять чрезвычайно большими и сложными наборами данных
- Импорт данных из систем управления базами данных
 - R может взаимодействовать с самыми разными системами управления базами данных (database management systems, DBMS), включая Microsoft SQL Server, Microsoft Access, MySQL, Oracle, PostgreSQL, DB2, Sybase, Teradata и SQLite. Некоторые пакеты предоставляют доступ через оригинальные драйверы баз данных, тогда как другие обеспечивают доступ к данным через ODBC (Open Database Connectivity interface – открытый интерфейс взаимодействия с базами данных) или JDBC (Java Database Connectivity – Java-интерфейс взаимодействия с базами данных). Использование R для доступа к данным, хранящимся во внешних DBMS, может быть эффективным способом работы с большими наборами данных.

2.12. Полезные функции для работы с объектами

Таблица 3.1. Обзор полезных функций для работы с объектами:

Функция	Описание
<code>length(объект)</code>	Число элементов/компонентов объекта
<code>dim(объект)</code>	Число измерений объекта
<code>str(объект)</code>	Структура объекта
<code>class(объект)</code>	Класс или тип объекта
<code>mode(объект)</code>	Способ хранения (вид) объекта
<code>names(объект)</code>	Названия частей объекта
<code>c(объект, объект, ...)</code>	Объединяет объекты в вектор
<code>cbind(объект, объект, ...)</code>	Объединяет объекты в виде столбцов
<code>rbind(объект, объект, ...)</code>	Объединяет объекты в виде строк
<code>объект</code>	Выводит на экран весь объект
<code>head(объект)</code>	Выводит на экран первую часть объекта
<code>tail(объект)</code>	Выводит на экран последнюю часть объекта
<code>ls()</code>	Выводит на экран список имеющихся объектов
<code>rm(объект, объект, ...)</code>	Удаляет один или более объектов. Команда <code>rm(list = ls())</code> удалит почти все объекты из рабочего пространства
<code>новый_объект <- edit(объект)</code>	Редактирует объект и сохраняет результат в виде нового объекта
<code>fix(объект)</code>	Редактирует объект

Мы уже обсудили большинство из приведенных функций. Команды `head()` и `tail()` используются для быстрого просмотра больших наборов данных. Например, `head(patientdata)` выводит на экран первые шесть строк таблицы данных, тогда как функция `tail(patientdata)` выводит последние шесть.

2.13. Задания к лабораторной работе

Изучив, текст лабораторной работы, ознакомиться с простыми способами анализа данных с помощью функций и графиков, опробовать приведенные способы загрузки данных.

1. Выполнить учебный импорт любых таблиц данных из csv-файла и xls-таблицы.
 2. Выполнить дескриптивный анализ данных из ЛР №2_1 (знать, что входит в дескриптивный анализ).
 3. Выполнить сортировку наборов данных по выбранному признаку.
 4. Сформировать отдельные наборы данных по одинаковому признаку (например, составить `subdataset`, из студентов, отдавших предпочтение по шкале > 0.7 определенной книге), вывести результат, выполнить подсчет размерностей новых таблиц, снова выполнить их анализ – гистограмма, боксплот, срединные меры (см п.2)
 5. Результаты пояснить, сделать выводы (каждый график должен иметь подрисуночную подпись – Рис.1. Название графика. И пояснения к графику).
- Составить отчет с проведенным анализом данных. (отчет по ЛР 1-2 необходимо предоставить в случае, если вы отстаеете от графика сдачи Лабораторных работ).