1. НАЧАЛО РАБОТЫ В СТАТИСТИЧЕСКОЙ СРЕДЕ R

R – это свободная программная среда для статистической обработки. Одновременно это и язык программирования. Хорошая альтернатива SPSS, STATISTICA и другим статистическим пакетам, проигрывающая им, пожалуй, только по интерфейсу (если сравнивать с RConsole) и многократно превосходящая по функциональности. Достигается это благодаря большому количеству специализированных пакетов (библиотек 4000), которые можно скачивать и подключать по мере необходимости. Благодаря открытому коду библиотеки постоянно пополняются, и ученые-статистики дарят миру не только обоснование новых методов в своих публикациях, но и готовый инструментарий, позволяющий пользователю применить эти методы на своем массиве данных.

Но здесь кроется и подвох. Отнюдь не любой пользователь может правильно воспользоваться все тем богатством методов и алгоритмов, реализованных на R. Для этого он должен быть подкованным в статистике, да и не просто подкованным, а неплохо разбираться в этих методах (знать условия их применимости, интерпретацию результатов). Если в стандартных статистических пакетах можно найти в основном известные из базового курса прикладной статистики методы, которые успешно применяются уже почти сотню лет (а потому их результаты не вызывают сомнений), то в R при использовании современных методов обработки требуется основательно изучить используемый метод, а также функцию, которая его реализует. Благо все пакеты снабжены справочной информацией, однако ее полнота остается на совести автора пакета (как, впрочем, и соответствие описания алгоритмов их реализации). Но и здесь есть положительный момент: снижается возможность бездумного применения статистических методов, исследователь обращает больше внимания на ограничения применимости инструментария, на различные варианты реализации того или иного анализа.

Стоит заметить, что на русском языке не так много литературы, посвященной языку R. Но все же есть источники, объясняющие особенности работы в R [1, 2], а также разбирающие современные методы статистического анализа с использованием программной среды R [2–5]. На английском языке литературы гораздо больше. Можно, например, рекомендовать пособие для начинающих [6].

Скачать R можно с сайта www.r-project.org. Здесь рассматривается работа в самом простом приложении RConsole, поэтому все материалы пригодны и для использования в специальных программах типа RStudio, в которых дополнительным преимуществом выступает интерфейс (контекстные подсказки, список объектов и др.).

1.1. ТИПЫ ДАННЫХ

Первое, в чем стоит разобраться, это какие типы данных (переменных) есть в R. Для этого нужно вспомнить, что наряду с обычной для анализа числовой информацией (расходы, прибыль) мы часто имеем дело с разного рода качественными признаками (пол, профессия) и с чем-то средним, т. е. признаками, имеющими градации изменения качества (удовлетворенность, уровень образования). Все переменные (вне зависимости от их шкалы измерения), при желании, могут быть закодированы, т. е. тому или иному уровню может быть поставлено в соответствие число («мужской» – 1, «женский» – 2). Но это не значит, что мы можем с ними обращаться, как с числами (например, рассчитать средний пол). Поэтому лучше вводить (или считывать) в R данные в том виде, в котором они изначально представлены. Но поскольку при вводе удобнее обращаться с числами, то рекомендуется при большом числе качественных признаков использовать для ввода данных SPSS. При этом заранее каждому ответу респондента нужно присвоить код и внести его в программу при описании переменных.

Стоит иметь в виду, что одна переменная (столбец данных) должна содержать значения (наблюдения) одной и той же природы. Это означает, что в одной переменной не могут быть перемешаны качественные и количественные составляющие, иными словами, стоит сразу определиться со шкалой измерения каждого признака. Казалось бы, что это очевидно, но иногда возникают проблемы. Например, если вопрос касается среднего заработка семьи в денежных единицах (числовой характер), часто возможен отказ от ответа, т. е. вариант «затрудняюсь ответить». Очевидно, доход у респондента все-таки имеется,

поэтому просто приравнять к нулю такие значения нельзя. В этом случае имеет смысл поставить NA, что означает not available (пропущенное значение). Это значение не является текстовым, поэтому вектор останется числовым. То же самое касается и тире или тильд (и просто пустых ячеек), которые часто ставятся в статистических таблицах на месте отсутствующих данных. Они должны быть в обязательном порядке заменены на NA.

Есть и более деликатные вопросы, касающиеся шкалы измерения. Например, при опросе об объемах потребления алкоголя (количественная переменная) могут быть и качественные варианты ответа: «затрудняюсь ответить (не могу точно сказать)», «не потребляю алкоголя в принципе». В первом случае, конечно, ответ необходимо кодировать как NA, во втором, казалось бы, стоит поставить ноль. Однако тогда эта ситуация не будет отличаться от нулевого потребления алкоголя в текущий период без принципиального отказа от алкоголя. Вместе с тем для исследования спроса на алкоголь эта разница может быть принципиальной. Возможно, идеальным выходом было бы включение дополнительной логической (бинарной) переменной с ответом на вопрос «Отказываетесь ли вы принципиально от алкоголя?». Бинарная (логическая) переменная принимает только два значения: 0 (FALSE) и 1 (TRUE). Теперь понятно, что кодировка данных должна быть тесно увязана с целями анализа, чтобы уже при вводе не потерять важной информации.

Еще одно попутное замечание: регистр важен! Иными словами, например, true и TRUE – это разные вещи, первое воспринимается как название объекта, второе – как значение логической переменной. Все текстовые значения нужно брать в кавычки, т. е. «true» – это значение текстовой переменной.

Теперь разберем подробнее, как создавать и редактировать переменные в R. Приводимый далее код можно полностью скопировать в командную строку. Команды будут выполняться по строкам. Знак «#» означает комментарий (команда после «#» не выполняется). Стоит отметить, что если строка не закончена (т. е. выражения переносятся на следующую строку знаком абзаца), то в R команда считывается со следующих строк (пока команда не закончится, например, не закроется скобка). Если необходимо в одну строку ввести несколько команд, то между ними ставится точка с запятой. В конце строки можно точку с запятой не ставить. Стрелками вверх и вниз можно перебирать введенные ранее команды.

Команды в R состоят из выражений и/или присвоений. Выражения (при отсутствии присвоений) выполняются вместе с выводом результатов на экран и теряются. Если необходимо их хранить в памяти, то используется присвоение выражения какому-либо объекту, название которого вводится символами. Присвоение осуществляется с помощью эквивалентных знаков «=» или «<-». При этом запись *x*<-*y* означает, что *x* будут присвоены значения *y*, *x*->*y* наоборот. Следовательно, сначала выполняются все операции справа от знака равенства и с начала стрелки. При присвоении результат операции не выводится автоматически на экран, для вывода нужно заключать запись в скобки (*x*<-*y*).

Выражения, как правило, содержат функции, сначала вводится название функции, затем в скобках через запятые перечисляются ее аргументы. Например, функция *c*(*x*,*y*,…) объединяет значения *x*,*y*,… в вектор, причем шкала измерения определяется автоматически по введенным значениям. Если значения разнородные (числовые и текстовые), то все значения преобразуются в текстовые. Логические значения FALSE и TRUE эквивалентны 0 и 1 при операциях с числами и переводятся в текстовые в текстовых векторах:

```
# числовой вектор, десятичным разделителем является точка
numeric_vector<-c(2.3,-5.6,7.3,1,6.1)
#упорядоченные последовательности целых чисел (оператор ":" имеет приоритет над арифметическими действиями)
integer_vector<-1:10; 100:81;-3:3; -(3:5); 0.2*0:5
# преобразование числового вектора в целочисленный путем отбрасывания дробной части
num_as_int <- as.integer (numeric_vector)
# чтобы просмотреть значения вектора, просто вводим его название
numeric_vector; num_as_int
# текстовый вектор
character_vector<-c(1,2,"незнаю")
# логический вектор
logical_vector<-c(TRUE,FALSE,FALSE)
# чтобы проверить, является ли вектор числовым, текстовым, логическим
is.numeric (numeric_vector); is.character (character_vector); is.logical(logical_vector)
# чтобы узнать структуру объекта, вводим str(x)
```

```
str(numeric_vector); str(character_vector); str(logical_vector)
# преобразование типа векторов
num_as_chr<-as.character(numeric_vector);
chr_as_num<-as.numeric(character_vector)
num_as_logi<-as.logical(c(1,0,0))
```

Наряду с векторами для представления однотипных данных в многомерном пространстве используются матрицы (двумерное пространство) и многомерные массивы. Создаются они следующим образом:

```
M<-matrix(1:10,nrow=5,ncol=2,byrow=TRUE)
Arr<-array(1:8,c(2,2,2))
```

Если необходимо в одном объекте представить данные разных типов, т. е. числовые, текстовые и логические, то используются списки. Их удобно создавать как результат выполнения пользовательских функций, поскольку функция возвращает только один объект. Именно в виде списков представлены результаты большинства встроенных функций в R:

```
List = list (M, Arr)
```

1.2. ОПЕРАЦИИ И ФУНКЦИИ

Самым необычным в R (к чему нужно привыкнуть в процессе работы) является то, что операции над векторами выполняются покомпонентно. Например, если вектор возводится в квадрат, то в результате получаем новый вектор той же размерности (с тем же числом элементов), в котором каждое значение равно квадрату соответствующего значения исходного вектора. Аналогично при умножении вектора на вектор соответствующие значения векторов перемножаются. Если векторы разной размерности, то вектор меньшей размерности будет циклически продолжен до вектора большей размерности. В частности, если один из векторов является константой (а это в R тоже вектор!), то просто все значения большего вектора преобразуются на эту константу.

Если размерность большего вектора не кратна размерности меньшего, то при применении совместных операций над ними появится предупреждение. К таким предупреждениям нужно быть внимательным, поскольку, скорее всего, вы вовсе не хотели перемножать векто-

ры разной размерности, но где-то возникла ошибка в процессе ввода данных или (что чаще всего) при фильтрации данных (извлечении данных, удовлетворяющих заданному условию).

Предупреждение о том, что в результате вычислений получено значение NaN (Not a Number), возникает в результате операций по извлечению корня из отрицательного числа с помощью функции sqrt(-abs(x)). Однако (-abs(x))^0.5 просто возвращает NaN без предупреждений. Если нужно получить результат в комплексных числах, то в выражении требуется указать комплексную часть числа, например, sqrt(-1+0i).

При использовании операторов арифметических действий для осуществления операции над матрицами, так же, как над векторами, происходит покомпонентное вычисление оператора. Иными словами, при перемножении матриц одинаковой размерности мы получим матрицу той же размерности, каждый элемент которой равен произведению соответствующих элементов исходных матриц. Для матричного перемножения в стандартном его понимании используется оператор «% * %»:

```
x<-1:10; y<-c(2,6,4,8,1,0,9,5,7,-3)
# размерность вектора (число элементов)
length(y)
# сумма и произведение квадратов элементов вектора x
sum(x^2); prod(x^2)
# покомпонентное деление векторов, при делении на 0 результат вычислений Inf (Infinity)
x/y
# перемножение векторов разной размерности
x*(1:2); x*c(y,y); 20*x
```

В табл. 1–3 представлены основные функции, которые понадобятся для решения задач, и приведено их краткое описание.

Таблица 1

Основные математические функции языка R

Название	Описание
choose	Число сочетаний
combn	Все возможные комбинации, содержится в пакете {combinat}
diff	Первые разности элементов вектора

Окончание табл. 1

Название	Описание
length	Число элементов, которые содержит объект
max	Максимальное значение
outer	Внешнее произведение
sum	Сумма элементов, входящих в объект
table	Вектор с числом повторений и названиями уникальных элементов
prod	Произведение элементов, входящих в объект
runif	Случайное равномерно распределенное число
abs	Модуль
cumsum	Накопленная (кумулятивная) сумма элементов вектора

Таблица 2

Основные функции языка R для работы с массивами

Название	Описание
cbind	Объединение векторов и массивов по столбцам
rbind	Объединение векторов и массивов по строкам
names	Название элементов вектора
rep	Повторение заданных элементов заданное число раз
which	Возвращает индексы элементов логического вектора (массива), равных TRUE
%in%	Используется для соотнесения элементов двух векторов. Возвращает TRUE на тех позициях, где элементы совпадают, и FALSE на остальных позициях
apply	Применяет функцию к каждой строке (или столбцу) двумерного массива.
mapply	Применяет функцию к каждому компоненту списка
replicate	Выполняет функцию заданное число раз
sample	Извлекает заданное число элементов из объекта случайных образом. По умолчанию (replace = FALSE) выборка без возвращения. Если поменять значение аргумента (replace = TRUE), будет производиться выборка с возвращением
tapply	Применяет функцию к элементам первого аргумента, соответствующим каждому уникальному элементу второго аргумента

Таблица 3

Основные графические функции языка R

Название	Описание
plot	Рисует график, по умолчанию – точечный. Если установить type = 'h', то получим график с вертикальными линиями
polygon	В существующем графическом окне изображается полигон
abline	Добавление линии на график
text	Добавление текста на график
curve	Изображение графика заданной функции

1.3. ЛОГИЧЕСКИЕ И УСЛОВНЫЕ ОПЕРАТОРЫ

Для сравнения значений векторов используются условия, заданные с помощью логических операторов «<», «<=», «>», «>=». При этом проверка точного равенства значений осуществляется с помощью оператора двойного равенства «==». Не стоит путать с «=», которое означает знак присвоения! Проверить различие значений можно с помощью оператора «!=» (не равно). Для объединения (оператор ИЛИ) двух логических выражений L1 и L2 используется вертикальная черта L1|L2, для пересечения (оператор И) – знак амперсанда L1&L2. Восклицательный знак означает операцию логического отрицания, т. е. !L1 возвращает условие противоположное L1.

Результатом выполнения логического выражения является логический вектор. В этом легко убедиться, если выполнить код

```
x = 1:10; y = c(2,6,4,8,1,0,9,5,7,-3)
L1 = x>y
```

Условие со стандартным оператором if записывается следующим образом:

```
if (x[1] == 2) print(1) else print(0)
```

Если нужно выполнить несколько действий, то их берут в фигурные скобки и перечисляют через точку с запятой. Выражение else можно опустить, если в нем нет необходимости.

Условие в if всегда должно быть логическим вектором с одним элементом. Если это будет вектор с большим числом элементов, опе-

рация все равно будет выполняться, но будет использоваться только первый элемент вектора. Предупреждение об этом появится на экране.

Для того чтобы выполнять некоторые действия по условию сразу на все элементы вектора или массива, используется функция ifelse. Оформляется она несколько иначе. Например, если выполнить код

```
ifelse (x > 7, y, x);
```

то все элементы в векторе *x*, которые больше 7, заменятся на соответствующие элементы вектора *y*. Поясним, как это происходит.

Условие $x > 7$ формирует логический вектор

```
FALSE FALSE FALSE FALSE FALSE FALSE FALSE TRUE TRUE TRUE
```

Далее на всех позициях с TRUE помещаются соответствующие элементы вектора *y*. В нашем случае это позиции 8, 9, 10. На всех позициях с FALSE помещаются соответствующие элементы вектора *x*. Это позиции с первой по седьмую. И возвращается этот результат.

В ifelse можно задавать и более сложные действия. Использовать его гораздо лучше, чем писать цикл с if. Проблемы с циклами мы обсудим ниже.

1.4. ИЗВЛЕЧЕНИЕ ДАННЫХ

Для отбора данных в соответствии с заданным условием используется индексный вектор, заключаемый в квадратные скобки и ставящийся после объекта, из которого необходимо извлечь данные.

Индексный вектор может быть логическим, тогда в результате выражение возвращает только значения, соответствующие TRUE или NA (если логический вектор содержит NA). В этом случае индексный вектор должен быть той же размерности, что и вектор данных. Если он меньшей размерности, то циклически повторяется до достижения размерности вектора данных. Иногда это удобно использовать (например, для извлечения нечетных чисел из регулярной последовательности). Но как правило, такая ситуация возникает вследствие ошибки при задании логического выражения, когда переменная фильтрации (допустим, пол) взята для какого-то другого подмножества респондентов меньшей размерности. Очевидно, что циклическое повторение индексного вектора не приводит нас к нужному результату, а извлекает ка-

кие-то лишние данные. Причем никаких сообщений об ошибке не появляется. Поэтому очень важно внимательно отслеживать правильность указанных логических выражений.

Кроме того, индексный вектор может быть представлен в виде натуральных чисел, которые будут проинтерпретированы как номера элементов, поэтому в результате извлекутся только те значения, которые соответствуют указанным номерам. Если номер будет превышать размерность вектора, то на его месте будет возвращаться NA.

Индексный вектор может содержать только отрицательные целые значения, тогда это означает, что элементы с соответствующим номером будут исключены. Недопустимо использовать как положительные, так и отрицательные значения в индексном векторе, т. е. одновременно включать и исключать элементы. В случае такой необходимости можно использовать извлечение два раза, т. е. *x*[L1][L2], но нужно иметь в виду, что второй раз значения будут извлекаться уже из вектора *x*[L1] другой размерности, поэтому нумерация элементов может измениться.

Если элементы вектора (а также строки или столбцы матрицы) имеют названия (имена могут быть заданы с помощью функций names, colnames, rownames), то для извлечения данных индексный вектор можно построить на основе этих названий.

```
    x[L1]
    x[1:3]
    # все нечетные числа из последовательности натуральных чисел
до 10
    (1:10) [c(TRUE,FALSE)]
    # удаляем первый и последний элементы
    x[-c(1,length(x))]; x[-1][-length(x)]
    # включаем первые 5 наблюдений и исключаем первый
    x[1:5][-1]; x[2:5]
    # создаем вектор и присваиваем имена каждому элементу
    Vector<-1:10; names(Vector)<-c (LETTERS[1:5], 1:2)
    Vector[c("A", "B")]
    # для сравнения то, что мы получаем с помощью логических условий
    Vector[names(Vector)=="A"|names(Vector)=="B"];
Vector[!(names(Vector)=="A"|names(Vector)=="B")]
```

1.5. ЦИКЛЫ

В R доступны стандартные операторы циклов while и for. Используется следующий синтаксис:

```
while (cond) expr
```

Условие выполнения тела цикла cond, как и условие в if, всегда должно быть логическим вектором с одним элементом. При cond = = TRUE цикл выполняется, как только cond = FALSE, происходит выход из цикла. Тело цикла expr заключается в фигурные скобки, если содержит несколько выражений. Эти выражения перечисляются через точку с запятой.

В условии cond не должно быть объектов, которых не существует в рабочем пространстве. Перед тем как начать while, нужно задать все переменные условия.

Стоит иметь в виду, что внутри цикла переменная цикла не меняется, т. е. если никак не меняется cond, а в начале выполнения цикла cond = TRUE, цикл будет бесконечно выполняться. Поэтому не нужно забывать, например, прибавлять счетчик. Цикл для суммирования натуральных чисел до *k* включительно будет выглядеть так:

```
k=100
i=1; ss=0
while (i <= k) {ss=ss+i; i = i+1}
```

Оператор for выполняет действия для всех элементов последовательности. Используется следующий синтаксис:

```
for (i in seq) expr
```

Здесь *i* – это переменная цикла, seq – последовательность элементов, которые должна принимать переменная цикла, обычно используются регулярные последовательности, создаваемые оператором «:». Выражение expr (как и ранее) – тело цикла, т. е. выполняемые действия.

Стоит отметить ряд особенностей. Во-первых, при выполнении for создается переменная цикла, т. е. ее не нужно определять заранее. Во-вторых, на каждой итерации *i* принимает соответствующее значение из последовательности seq, поэтому переопределение *i* внутри цикла не влияет на число итераций. И здесь не нужно внутри for добавлять счетчик, как это делается в while.

Тот же самый цикл для суммирования натуральных чисел до k включительно с использованием for будет выглядеть более компактно:

```
ss=0
for (i in 1:100) ss=ss+i
```

Цикл можно прервать оператором break, закончить текущую итерацию и перейти к следующей – оператором next.

Следует отметить, что циклы в R работают медленно. Вообще во многих случаях лучше использовать векторизованные функции, чем прибегать к циклам. Это будет компактнее, и код будет лучше читаться.

Например, то же самое суммирование чисел можно выполнить с помощью функции sum (табл. 1) следующим образом:

```
ss = sum (1:100)
```

На малых размерностях разница в скорости выполнения кода, конечно, не будет ощутима. Но когда нужно выполнить одинаковые действия для миллионов и более элементов, то векторизованные функции незаменимы. Наиболее часто используемые из них apply, tapply и всякого рода подобные функции. Они позволяют применить одну и ту же функцию к строкам (столбцам) массива или к элементам, соответствующим уникальным значениям другого объекта. Тем самым они заменяют цикл по номеру строки (столбца) или по последовательности уникальных элементов объекта.

В apply, tapply и такого рода функциях можно использовать стандартные функции. Но чаще возникает необходимость в каких-то нестандартных действиях, поэтому без создания пользовательских функций не обойтись.

1.6. СОЗДАНИЕ ПОЛЬЗОВАТЕЛЬСКИХ ФУНКЦИЙ

Функции представляют собой объекты типа function. Единственное их отличие от других объектов, например от вектора, в том, что для их вызова используются круглые скобки. Вот с помощью такого синтаксиса создается функция:

```
fun = function (arg1, arg2) expr
```

Название функции fun, т. е. если она создана (выполнен код), то можно обращаться к функции по названию fun(1,2) с указанием в скобках значений ее аргументов; arg1, arg2 – это имена аргументов; над ними, как правило, выполняются действия в теле функции expr.

Заметим, что функция не выполняется до ее вызова, т. е., в принципе, в ней может быть записано все, что угодно, только без синтаксических ошибок. И она будет иметь право на существование. Ошибка (в нашем случае, не существует объекта expr) будет возникать только при вызове функции.

Можно также указывать значения аргументов по умолчанию. Делается это так:

```
fun = function (arg1 = 1, arg2 = 2) expr
```

Это будет означать, что если, например, не указано значение аргумента arg1, то он принимает значение 1.

При вызове функции значения аргументов указываются:

– в порядке следования, например fun(1,2);

– при нарушении порядка следования – по именам, например fun (arg2 = 1, arg1 = 2).

Часто порядок следования нарушается, когда мы не присваиваем значения аргументам, имеющим умолчания.

Приведем пример функции, выполняющей подсчет количества элементов числового вектора, кратных 7. Для проверки кратности можно использовать оператор «%%», возвращающий остаток от деления. Если остаток от деления равен 0, то число кратно 7. Значит, условие будет выглядеть как

```
x %% 7 == 0
```

Далее мы можем просто просуммировать значения логического вектора. Поскольку TRUE будет заменен на 1, а FALSE на 0, то при суммировании мы получим количество раз, когда выполнилось условие, т. е. чисел, кратных 7:

```
sum (x  %% 7 == 0)
```

Тогда функция будет иметь вид

```
div7 = function (x) sum (x  %% 7 == 0)
```

Эту функцию можно использовать внутри apply, чтобы подсчитать, сколько элементов каждого столбца (или строки) массива кратны 7. Создадим массив для проверки, например, с помощью функции внешнего произведения outer:

```
m = outer(1:10,1:10,"+")
```

Далее применим к нему функцию:

```
r = apply (m, 1, div7)
```

Можно было указать функцию непосредственно внутри apply, чтобы не создавать много лишних функций, особенно если они нужны только для однократного использования. Делается это так:

```
r = apply (m, 1, function (x) sum (x  %% 7 == 0))
```

Итак, ограничимся этими основными сведениями по работе в среде R. Дополнительную информацию о работе функций (аргументы, их порядок, значения по умолчанию и описание) можно получить, если вызвать справку по функции, например, таким способом:

```
?rep
```

В справке также приведены и примеры применения функции, что позволяет лучше разобраться в ее работе.