Федеральное государственное образовательное бюджетное учреждение
высшего профессионального образования
«ФИНАНСОВЫЙ УНИВЕРСИТЕТ
ПРИ ПРАВИТЕЛЬСТВЕ РОССИЙСКОЙ ФЕДЕРАЦИИ»
(Финансовый университет)

Департамент Математики

Дисциплина «Программирование в среде R»

П.Б. Лукьянов

МЕТОДИЧЕСКИЕ УКАЗАНИЯ ПО ВЫПОЛНЕНИЮ
КОНТРОЛЬНОЙ РАБОТЫ № 1

Расчет показателей эффективности торговой сети

Часть 2. Генерация данных

Для студентов, обучающихся по направлению подготовки
«Прикладная информатика»
(программа подготовки бакалавра)

Москва 2021

Данные методические указания последовательно освещают вопросы генерации исходных данных на языке R с использованием функций, созданных Пользователем.

Часть материала, описывающая управляющие параметры функций write.table() и read.table() приводится повторно, для удобства изложения. Изложение ведется от простого к сложному. Весь код, представленный ниже, нужно набрать в RStudio и проверить его работу на практике.

Пусть функция **generate.supply()** должна выполнять генерацию поставок некоторого товара в магазин. Функция должна создавать текстовый файл с двумя столбцами: день месяца (или недели, декады) и значение товара, переданное в магазин, в единицах измерения товара. Количество поставляемого товара в разные дни может отличаться, поэтому при генерации значений поставок должен использоваться генератор случайных чисел.

У функции generate.supply() должны быть следующие параметры:

- Имя файла без расширения. По умолчанию должен создаваться файл с расширением.txt (в последующем расширение файла будет изменено)
- Место размещения файла (может указываться абсолютный или относительный путь к файлу)
- Минимальное значение поставки за день
- Максимальное значение поставки за день
- Количество дней, для которых создаются данные по поставкам

Для каждого параметра нужно задать значение по умолчанию, чтобы функцию можно было вызывать без указания параметров.

Один из возможных вариантов кода функции и варианты ее вызова представлены на рис. 1.

На рис. 2 представлены файлы с данными, полученные при разных значениях входных параметров.

```r
generate.supply <-
  function(way = '',
           file.name = 'Поставка',
           days = 7,
           min = 100,
           max = 140) {

    # в среде R создадим таблицу из двух столбцов длиной days
    tabl <-
      data.frame(1:days, as.integer(runif(
        n = days, min = min, max = max
      )))

    # запишем эту таблицу в файл с расширением txt
    write.table(
      x = tabl,
      file = paste0(way, file.name, '.txt'),
      col.names = FALSE,
      row.names = FALSE
    )
  }

generate.supply()
generate.supply(file.name = 'Магнит', days = 30)
generate.supply(file.name = 'Дикси', days = 20, min=1300, max=2000)
```

Рис. 1. Код и вызовы функции generate.supply()

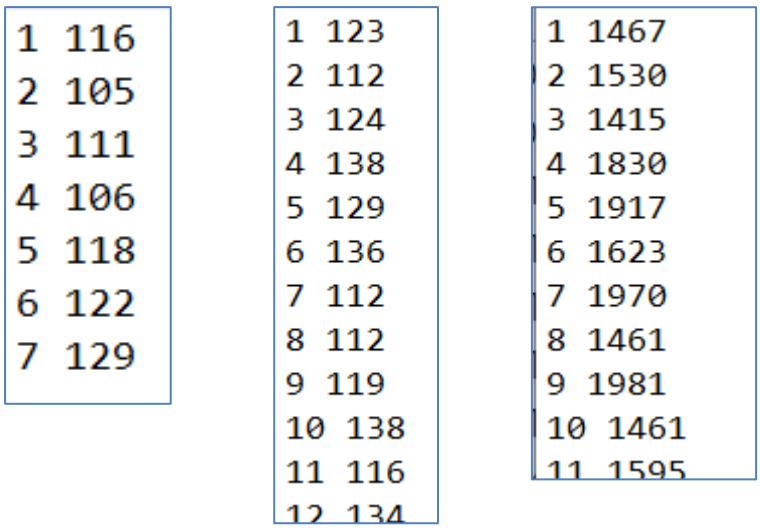

Рис. 2. Файлы с данными. Работа функции generate.supply()

Функция generate.supply() должна создавать различные наборы исходных данных для последующего использования их при моделировании экономической деятельности магазина. Имея множество вариантов данных по поставкам и продажам, можно проиграть различные сценарии принятия управленческих решений и оценить их экономический эффект.

В функции generate.supply() для записи таблицы в текстовый файл используется функция write.table(). При вызове функции нужно задать некоторые параметры, управляющие записью (см. табл. 1).

Таблица 1. Основные параметры функции write.table()

Имя	**Значение**
x	Переменная языка R, содержащая таблицу данных
file	Строка с именем файла. В строке может быть или только имя файла, или же имя с абсолютным или относительным путем к нему. Если указано только имя файла, поиск файла будет выполняться в текущей (рабочей) папке. Узнать, в какой папке мы находимся, можно с помощью функции getwd(). Для перехода в заданную папку используется функция setwd(). В строке можно указывать URL-ссылку на файл, если он размещен в сети. Если имя файла не задано (file=’’), таблица будет выведена в консоль. По умолчанию file=’’.
col.names	По умолчанию col.names=TRUE. Это означает, что в файл будет записана первая строчка с именами столбцов. В обратном случае (col.names=FALSE) имена записаны не будут.
row.names	По умолчанию row.names=TRUE. Это означает, что в таблицу будут записан первый столбец с именами строк. Если предполагается дальнейшее использование файла в программе Excel или других табличных редакторах, то наличие в файле имен строк приведет смещению заголовков столбцов, поэтому нужно задавать row.names=FALSE.

sep	Разделитель значений в строках таблицы. По умолчанию sep = ‘ ‘.
dec	Параметр определяет, какой символ используется для отделения целой части от десятичной. По умолчанию dec=”.”. В русскоязычной версии Excel десятичным разделителем является запятая, поэтому для открытия таблицы в Excel использовать dec = ‘,’.

Параметры по умолчанию

Обратите внимание на возможность вызова generate.supply() без параметров. Вызов функции без задания значений параметрам называется вызовом функции «по умолчанию». При таком вызове функция считает, что параметры все равно переданы, но они установлены в конкретное значение «по умолчанию», прописанное в определении функции, и исходя из этого значения, функция работает определенным образом.

В общем случае у функции может быть множество параметров, и у каждого параметра могут быть свои значения по умолчанию. Программист при вызове функции, как правило, задает значения лишь нескольким параметрам, оставляя значения других параметров по умолчанию.

Пока файлы с данными не слишком информативны – не хватает заголовков столбцов с указанием дней и названия товара. Поэтому добавим в функцию еще один параметр – название товара и создадим заголовки (см. рис. 3). Обратите внимание, для создания заголовков столбцов используется функция colnames(), которой передается номер столбца из таблицы.

```r
generate.supply2 <-
  function(way = '',
           file.name = 'Поставка',
           days = 7,
           min = 100,
           max = 140,
           goods) {

    # в среде R создадим таблицу из двух столбцов длиной days
    # для первого столбца создаем заголовок
    tabl <-
      data.frame('День' = 1:days, as.integer(runif(
        n = days, min = min, max = max
      )))

    # используем функцию colnames() для создания заголовка столбца
    # разные способы обращения к столбцу, оба правильные
    colnames(x=tabl)[2] = goods
    colnames(x=tabl[2]) = goods

    # запишем эту таблицу в файл с расширением txt
    write.table(
      x = tabl,
      file = paste0(way, file.name, '.txt'),
      col.names = TRUE,
      row.names = FALSE
    )
    return(tabl)
  }

z <- generate.supply2(goods = 'Апельсины, ящики')
z
```

Рис. 3. Код и вызов функции generate.supply2()

Ключевое слово return

В функции generate.supply() не использовался return(), но для последующего анализа в R нам будет нужна полученная таблица, поэтому в функцию generate.supply2() добавим return(tabl). Отредактируйте код функции и вызовите ее, оцените результат.

Ключевое слово **return** используется в описании функции, чтобы явным образом определить результат, который возвратит функция при ее вызове. При достижении **return** работа функции завершается и происходит возврат в основную программу, в точку вызова функции.

Если в основной программе результат вызова функции присваивался переменной, то в переменной будет значение, указанное в круглых скобках после return в теле функции. Если **return** не использовать, то результатом работы функции будет результат выполнения последнего оператора в теле функции.

Генерация произвольного количества товаров

Следующее улучшение в работе функции заключается в требовании, чтобы функция могла обрабатывать произвольное количество товаров. Для этого в параметр goods будем передавать вектор с названиями, а в самой функции обрабатывать каждый элемент вектора goods (рис. 4).

Получившаяся таблица представлена на рис. 5. Для каждого товара сейчас используется один и тот же диапазон значений, что в общем случае не правильно.

Улучшение функции должно быть следующим: генератор случайных чисел должен работать в своем диапазоне для каждого товара. В параметры функции нужно передавать минимумы и максимумы для каждого товара.

Изменим функцию generate.supply4(), выполним это требование, и вызов функции станет следующим (см. рис. 6):

```r
generate.supply3 <-
  function(way = '',
           file.name = 'Поставка',
           days = 7,
           min = 100,
           max = 140,
           goods = 'Молоко, уп.') {
    # создадим таблицу из одного столбца длиной days
    tabl <- data.frame('День' = 1:days)

    # добавляем столбцы под каждый товар
     for (i in 1:length(goods)) {
       tabl[i + 1] <- sample(x = min:max, size = days)
       colnames(x = tabl)[i + 1] = goods[i]
     }

    # запишем эту таблицу в файл с расширением txt
    write.table(
      x = tabl,
      file = paste0(way, file.name, '.txt'),
      col.names = TRUE,
      row.names = FALSE
    )
    return(tabl)
  }

z <-generate.supply3(
    goods = c('Апельсины, ящики', 'Капуста, кг', 'Орехи, уп.'),
    file.name = 'Магнит, поставка март'
  )
z
```

Рис. 4. Код и вызов функции generate.supply3()

День	Апельсины, ящики	Капуста, кг	Орехи, уп.
1	119	102	133
2	107	116	135
3	138	109	139
4	130	108	118
5	109	114	122
6	132	124	113
7	115	134	104

Рис. 5. Таблица с данными по поставкам нескольких товаров

```
z <-generate.supply4(
  goods = c('Сахар, пакеты','Капуста, кг','Орехи, уп.'),
  file.name = 'Магнит, поставка апрель',
  way = 'c:\\',
  days = 10,
  min = c(30, 210, 500),
  max = c(36, 240, 600)
)

z
```

Рис. 6. Вызов функции с заданием границ по каждому товару

Результат вызова функции показан на рис. 7 – у каждого товара свой диапазон значений поставок, в соответствии с заданными значениями min и max.

День	Сахар, пакеты	Капуста, кг	Орехи, уп.
1	34	223	507
2	33	218	573
3	30	237	507
4	31	215	571
5	31	212	524
6	30	227	521
7	32	223	516
8	33	215	579
9	34	239	565
10	34	229	546

Рис. 7. У каждого товара теперь свой диапазон значений

Использование списков

Подготовка данных для функции generate.supply4() вызывает определенные сложности: нужно отдельно формировать вектор названий товаров, отдельно готовить вектора для минимальных и максимальных значений.

Если в дальнейшем характеристики товара будут дополняться ценой, сроком хранения, артикулом и т.д., согласование многочисленных векторов между собой станет сложной задачей.

Поэтому реализуем следующий подход: весь набор данных по одному товару поместим в структуру list. Все подготовленные таким образом характеристики товаров объединим в одну переменную, и эту переменную будем передавать как параметр в функцию (рис. 8).

```
h <-list(
    list(name = 'Молоко, уп.', min = 600, max = 800, price = 65),
    list(name = 'Кефир, уп.', min = 200, max = 300, price = 76),
    list(name = 'Хлеб, шт.', min = 30, max = 50, price = 33),
    list(name = 'Вода, бут.', min = 400, max = 430, price = 22),
    list(name = 'Соль, пачка', min = 18, max = 22, price = 3),
    list(name = 'Гречка, пачка', min = 50, max = 60, price = 80),
    list(name = 'Торт, шт.', min = 20, max = 30, price = 1500)
)

z <- generate.supply5(goods = h,
                      way = 'd:\\test\\',
                      days = 20)
```

Рис. 8. Правильная группировка характеристик товаров

Как изменится обработка такой сложной переменной внутри функции? На рис. 9 представлен код, реализующий обработку списков, переданных как параметр.

```
# создадим таблицу из одного столбца длиной days
tabl <- data.frame('День' = 1:days)

# добавляем столбцы под каждый товар
for (i in 1:length(goods)) {
  tabl[i + 1] <-
    sample(x = goods[[i]]$min:goods[[i]]$max,
           size = days,
           replace = TRUE)
  colnames(x = tabl)[i + 1] = goods[[i]]$name
}
```

Рис. 9. Обращение к элементам списка

Именование элементов структур данных

Обратите внимание, как происходит обращение к элементам структуры list. В данном примере для доступа к элементам списка используются двойные квадратные скобки и знак доллара $. В языке R реализовано два механизма для доступа к элементу структуры данных: через индексацию, в этом случае используются квадратные скобки, и через задание метки (имени) некоторому полю. Рассмотрим второй способ более подробно. Создадим два вектора:

```
w1 <- c(12, 35, 2071)
w2 <- c(speed = 12, count = 35, res = 2071, -9.4, 88.003)
```

Традиционный способ получить значение, например второго элемента вектора – написать w1[2] или w2[2].

Но при создании вектора w2 некоторым элементам была присвоена метка: первому элементу – speed, второму – count и т.д. Использование имен или меток дает дополнительное удобство, так как теперь для доступа к элементу не обязательно знать его порядковый номер в векторе, гораздо проще обратиться к имени элемента, чтобы получить его значение:

```
w2[‘speed’] – для первого элемента
w2[‘count’] – для второго элемента и т.д.
```

В результате различные формулы с использованием имен элементов вектора будут выглядеть, например, так:

```
q <- w2[‘speed’] * w2[‘count’] / 12 + 7 ….
```

Такой же способ применяется для именования элементов в списках:

```
r <- list(name = 'Хлеб', price = 42, amount = 27, firm = 'Пекарня № 2')
```

Но при попытке использовать значения из элементов списка получим ошибку:

```
rub <- r['price'] * r['amount']
```

Дело в том, что любой элемент в списке считается списком, а не числовым значением. Поэтому для доступа к значению элемента требуется

дополнительная операция по извлечению его из структуры list и преобразованию к вектору:

```
rub <- unlist(r['price']) * unlist(r['amount']) или
rub <- unlist(r[2]) * unlist(r[3])
```

Каждый раз выполнять unlist(…) не очень удобно, поэтому в R придумана конструкция вида ИмяСтруктуры$ИмяЭлемента:

```
r$price, r$amount.
```

В этом случае формула будет выглядеть так:

```
rub <- r$price * r$amount
```

причем метку можно заключать в кавычки:

```
rub <- r$’price’ * r$amount
```

Если бы при реализации функции (см. рис. 9) мы бы отказались от использования конструкции $имяЭлемента, код был бы более сложным:

```
tabl[i + 1] <-
  sample( x = unlist(goods[[i]][2]):unlist(goods[[i]][3]),
    size = days,
    replace = TRUE )
```

У внимательного читателя возникает следующий вопрос: почему с использованием индексации для доступа к элементу списка используются двойные квадратные скобки?

На самом деле можно использовать как одинарные квадратные скобки, так и двойные. Различие в том, что в первом случае (одинарные скобки) возвращаемый объект будет новым списком с одним элементом, во втором случае (двойные скобки) будет возвращен тот тип объекта, по которому и сформирован первый элемент списка.

Таким образом, если использовать одинарные скобки для списка goods, нужно выполнить операцию unlist() еще один раз. Первый раз мы использовали unlist(), чтобы из внутреннего списка

```
list(name = 'Молоко, уп.', min = 600, max = 800, price = 65)
```

получить нужный элемент. Сейчас, при использовании двух вызовов unlist(), один unlist() позволит перейти от конструкции «список списков» к внутреннему списку, а второй unlist() преобразует внутренний список к вектору, элементы которого нам и нужны:

```
for (i in 1:length(goods)) {
  tabl[i + 1] <-
     sample(x = unlist(unlist(goods[i])[2]):unlist(unlist(goods[i])[3]),
     size = days,
     replace = TRUE
    )
  colnames(x = tabl)[i + 1] = unlist(goods[i])[1]
}
```

Осталось рассмотреть последний способ, при котором происходит обращение к элементам списка по их имени, как это делается в векторах, через квадратные скобки с именем элемента в кавычках:

```
for (i in 1:length(goods)) {
  tabl[i + 1] <-
    sample(x = unlist(unlist(goods[i])['min']):unlist(unlist(goods[i])['max']),
     size = days,
     replace = TRUE
    )
  colnames(x = tabl)[i + 1] = unlist(goods[i])[1]
}
```

В чем плюсы и минусы каждого способа? Очевидно, что самая простая форма доступа к элементам списка представлена на рис. 9, но эта конструкция будет работать, если у списка, передаваемого как параметр в функцию, будут те же самые имена у элементов, как и в коде самой функции.

Если при подготовке исходных данных имя пропущено или записано неверно, код на рис. 9 и последняя версия с конструкцией unlist(unlist(goods[i])['max']) будут давать ошибку.

Наоборот, код, в котором используется только индексация, более универсальный: данные, подготовленные с ошибками в именах или вообще без указания имен, будут обработаны корректно, но написание кода через индексацию более сложное.

Генерация данных по поставкам и продажам

Следующий шаг в развитии функции генерации данных заключается в том, чтобы в зависимости от переданного параметра формировать или файл поставок товаров, или файл продаж.

До объявления функции разместим несколько глобальных констант: две константы, задающие тип формируемых данных SUPPLY <- 1 для файла поставок и SALE <- 2 для файла продаж; зададим две константы для определения типа файла: FILE_SUPPLY <- ‘in’ для поставок и FILE_SALE <- ‘out’ для продаж.

Добавим в описание функции параметр dataType, значение по умолчанию SUPPLY. Перед формированием файла нужно добавить код определения типа файла (см. рис. 10):

```
ext <- switch(dataType, FILE_SUPPLY, FILE_SALE)

# запишем таблицу в файл с нужным расширением
write.table(
  x = tabl,
  file = paste0(way, file.name, '.',ext),
  col.names = TRUE,
  row.names = FALSE
)
return(tabl)
```

Рис. 10. Определение файла нужного типа

Итак, теперь имеется функция, которая для произвольного числа товаров создает файлы, содержащие наборы исходных данных для произвольного количества дней как по поставкам товаров, так и по продажам. Вместе с тем, готовить данные с помощью этой функции рано: при создании файлов по поставкам и продажам с одинаковыми границами минимумов и максимумов возможны ситуации, когда в некоторые дни продажи товара будут превышать поставку, что в реальных производственных условиях невозможно.

Вариантов решения проблемы два: готовить разные наборы минимумов и максимумов для файлов поставок и продаж, что не очень корректно, и второй вариант – в функцию добавить проверку данных на превышение продаж над поставкой. Если данные по продажам data.out превышают поставку data.in, то для data.out нужно брать значение data.in:

data.out <- ifelse(data.out > data.in, data.in, data.out)

Таким образом, в функции должна быть реализована следующая логика: если (dataType == SALE), функция в заданной папке ищет файл поставок с таким же именем; если файл найден, считывает данные в data.in, после генерации данных продаж data.out функция выполняет сравнение.

Если файл поставок не найден, функция должна создавать два набора данных data.in и data.out, в случае необходимости корректировать data.out и в результате записывать в файл поставок data.in, а в файл продаж – переменную data.out.

Для считывания таблиц из текстовых файлов предназначена функция **read.table()**. При вызове функции нужно определить некоторые параметры, управляющие считыванием данных из файла (см. табл. 2).

Примеры. Выполним считывание данных из таблиц:

data.in <- read.table(‘март.in’, header = TRUE)

data.out <- read.table(‘d:\\Дикси\\март.out’, header = TRUE)

Какого типа переменные data.in, fata.out?

Таблица 2. Основные параметры функции read.table()

Имя	**Значение**
file	Строка с именем файла. В строке может быть или только имя файла, или же имя с абсолютным или относительным путем к нему. Если указано только имя файла, поиск файла будет выполняться в текущей (рабочей) папке. Узнать, в какой папке мы находимся, можно с помощью функции getwd(). Для перехода в заданную папку используется функция setwd(). В строке с именем можно указывать URL-ссылку на файл, если он размещен в сети.
header	В файле в первой строке могут быть заголовки (имена столбцов). В этом случае строку нужно считать так, чтобы имена столбцов в файле стали именами столбцов таблицы R. Если заголовки есть, нужно выставить header в TRUE, иначе в FALSE. Значение по умолчанию FALSE.
row.names	В файле данных может быть столбец, в котором содержатся имена строк таблицы. Чтобы при считывании данных скопировать имена строк из файла в имена строк таблицы R нужно указать номер этого столбца, например, row.names = 1. Имена строк должны быть уникальными.
sep	от separator – разделитель. Параметр определяет, как отделяются друг от друга значения в строках. По умолчанию считается, что разделитель – пробел или табуляция. Если разделитель – точка с запятой, пишут sep = “;”
dec	Параметр определяет, какой символ используется для отделения целой части от десятичной. По умолчанию dec=”.”
nrows	Целое число, которое задает, сколько строк должно быть считано из таблицы

skip	Целое число, которое указывает, сколько строк с начала таблицы нужно пропустить перед началом считывания данных
encoding	Строка, задающая кодировку, в соответствии с которой будут преобразовываться коды символов из считываемого файла в символы таблицы.

В случае, если строковые поля или заголовки скопированной таблицы в R отображаются не корректно, при считывании таблицы нужно принудительно задать кодировку, в соответствии с которой будет обрабатываться строковые поля копируемой таблицы. В функции read.table() нужно задать значение параметра encoding, соответствующее той кодировке, в которой был сохранен файл с данными. Рекомендуется использовать кодировку UTF-8.

Если заголовки столбцов нуждаются в корректировке, существует несколько способов заменить имя столбца. Используются функции names() или colnames():

```
colnames(data.in)[1] <- "День недели"   или
names(data.in)[1] <- "День недели"
```

При таком способе изменения имени столбца необходимо явно задавать его номер, но номер в большой таблице может быть не известен. Рекомендуется использовать более сложную конструкцию:

```
names(data.in)[names(data.in) == "День"] <- "День недели" или
colnames(data.in)[which(names(data.in)=="День")]<-"День недели"
или
colnames(data.in)[which(colnames(data.in) == "День")]<-"День недели"
```

Логика в этих примерах следующая: функции names(data.in) или colnames(data.in) считывают все имена столбцов data.in в массив. По заданному условию на соответствие имени столбца нужному значению мы находим нужный столбец и присваиваем ему новое имя.

Проверка существования папок и файлов

Что произойдет, если функция read.table() не найдет файл? Функция напишет сообщение об ошибке, и работа программы завершится (см. рис. 11). Что произойдет, если будет попытка записи файла в папку, которой не существует? Также будет выведено сообщение об ошибке (рис. 11).

```
Ошибка в file(file, "rt") :не могу открыть соединение
Вдобавок: Предупреждение:
В file(file, "rt") :
```

```
Ошибка в file(file, ifelse(append, "a", "w")) :
  не могу открыть соединение
Вдобавок: Предупреждение:
В file(file, ifelse(append, "a", "w")) :
```

Рис. 11. Ошибки при обращении к несуществующим файлам и папкам

Чтобы Пользователь не видел этих ошибок, чтобы программа продолжала устойчиво работать, в функции генерации данных нужно выполнить проверки: существует ли папка, путь к которой передан в параметре way? И вторая проверка: существует ли файл с именем

paste0(way, file.name, '.', FILE_SUPPLY) ?

В том случае, если файла или папки не существует, функция должна создать файл или папку, и что важно, никаких сообщений и предупреждений об ошибках не должно выводиться.

В языке R есть функции dir.exists() и file.exists(), с помощью которых можно проверить существование папок и файлов. Функции возвращают TRUE, если папка или файл существуют и FALSE в обратном случае.

Поэтому в самом начале функции генерации данных добавим код на проверку существования папки (рис. 12), а в случае (dataType == SALE) проверим существование файла (рис. 13).

```
#================= проверим, существует ли папка
if (way != ''){
  isFoundDir <- dir.exists(way)
  if (isFoundDir == FALSE){
    dir.create(path = way, showWarnings = FALSE)
    isFoundDir <- dir.exists(way)
    if (isFoundDir == FALSE){
      print('Папка назначения не существует, создать ее нельзя')
      return(NULL)
    }
  }
}
```

Рис. 12. Код проверки существования папки и ее создания

```
# если формируем файл продаж, считываем или генерим поставку
if (dataType == SALE) {
  file.in <- paste0(way, file.name, '.', FILE_SUPPLY)
  isFoundFile <- file.exists(file.in)
  if (isFoundFile) {
    data.in <- read.table(file = file.in,
                          header = TRUE,
                          encoding = 'UTF-8')
  } else {
    data.in <- data.frame('День' = 1:days)
    for (i in 1:length(goods)) {
      data.in[i + 1] <-
        sample(
          x = goods[[i]]$min:goods[[i]]$max,
          size = days,
          replace = TRUE
        )
      colnames(x = data.in)[i + 1] = goods[[i]]$name
    }
    write.table(
      x = data.in,
      file = file.in,
      col.names = TRUE,
      row.names = FALSE,
      fileEncoding = 'UTF-8'
    )
  }
}
```

Рис. 13. Код проверки существования файла и его создания

Для случая (dataType == SALE) в функции генерации данных есть две таблицы data.in и data.out, выполним проверку на превышение продаж над поставкой (рис. 14).

```
# если это продажа, сравниваем data.out и data.in
if (dataType == SALE) {
  for (i in 1:length(goods)) {
    data.out[, i + 1] <-
      ifelse(data.out[,i+1] > data.in[,i+1],
             data.in[,i+1],
             data.out[,i+1])
  }
}
```

Рис. 14. Переопределение данных по продажам

На основании изучения изложенного материала задача написания собственной функции генерации данных не должна вызывать затруднений.

Решение следующей задачи – считывания и обработки данных по нескольким товарам также должно быть основано на использовании списков.

Для помещения таблиц всех магазинов в список можно использовать следующий подход: объявить переменные типа list() для хранения данных по поставкам и продажам и затем в цикле размещать в эти переменные считанные таблицы по каждому магазину.