Лабораторная работа №2. Предварительный анализ данных и модель регрессии

С.А.Суязова (Аксюк), sa_aksyuk@guu.ru
17 сен, 2021

Table of Contents

Ключевые слова: R[1], r-project, RStudio

Примеры выполнены R версии 4.1.1, «Kick Things».

Версия RStudio: 1.4.1717.

Все ссылки действительны на 8 февраля 2021 г.

---

[1] R Core Team (2015). R: A language and environment for statistical computing. R Foundation for Statistical Computing, Vienna, Austria. URL https://www.R-project.org/

Репозиторий с материалами к курсу: github.com/aksyuk/R_data_glimpse

Файл с макетом кода для этой практики: `.Labs/lab-02_before.R`

Лабораторная работа №2: Предварительный анализ данных и модель регрессии

Философия опрятных (tidy) данных

Если вы уже работали со статистикой от сбора до презентации результатов анализа, то наверняка заметили, что 80% времени уходит на подготовку данных. Понятие “опрятных данных” было введено с целью упорядочить шаги по такой подготовке, сократить время, затрачиваемое на них, и облегчить дальнейший анализ. Этот процесс в западной литературе называется “data tidying”, по-русски получается длиннее: “структурирование и приведение в порядок наборов данных” [2]. Требования к опрятным данным очень похожи на требования третьей нормальной формы в базах данных:

1. Каждая переменная формирует столбец.
2. Каждое наблюдение формирует строку.
3. Каждый тип единицы наблюдения формирует таблицу.

К счастью, в R есть удобные средства структурирования данных – они сгруппированы в коллекцию пакетов *tidyverse*. Пакеты из этой коллекции построены по принципу единой грамматики, в которой таблицы являются подлежащими, а действия над ними – глаголами. Мы рассмотрим средства пакета `dplyr`: функции непосредственной манипуляции данными. Но для начала остановимся на важной детали – правилах именования переменных.

Формально в R действуют несколько ограничений на имена:

- имя должно начинаться с буквы;
- имя должно содержать только буквы, цифры, подчёркивания и точки;
- имя не должно совпадать с именами системных констант (`pi`);
- крайне нежелательно совпадение имён пользовательских объектов с именами функций пакетов R.

Но это ещё не всё. Хороший код не просто формально и стилистически верен, но и понятен. Поэтому содержательные имена – хороший тон не только для переменных, но и для столбцов таблиц. Удачно подобранные имена ускоряют написание кода, однако слишком длинные могут сделать его нечитаемым. Так или иначе, выбираете вы содержательные имена или рискуете запутаться в переменных `my.val`, `my.val.1` и прочих случайных сочетаниях символов, главное – действовать последовательно. Остановитесь на стиле написания имён, который нравится именно вам:

---

[2] Олег Замощин [@turegum](https://habr.com/users/turegum/). Data tidying: Подготовка наборов данных для анализа на конкретных примерах / habr.com, 24 января 2015. URL: https://habr.com/ru/post/248741/

```
пишите_все_имена_в_змеином_регистре
илиИспользуйтеВерблюжийРегистр
далее.имена.в.основном.написаны.через.точки
Только.не_используйтеВсеВперемешку
```

И помните, что люди из Google уже написали руководства по стилю программирования на всех языках программирования, в том числе и на R [3].

Графические системы в R

R предоставляет пользователю широкие графические возможности. Если говорить о визуализации количественных данных, то на данный момент в R существуют три основных графических системы, которые принципиально отличаются друг от друга [4]:

1. **Встроенная система «base»** [5]. Построена по принципу конструктора: различные функции вызываются последовательно и дополняют график. При этом обязательно использование одной базовой функции, которая определяет тип графика и данные и создаёт область графика. Такими основными функциями являются, например, `plot()`, `boxplot()`, `curve()`.

2. **Система «trellis»**, реализованная в пакете **«lattice»** [6]. Была создана специально для изображения кросс-секционных данных и направлена в основном на создание графиков по категориям значений. В отличие от «base», подавляющее большинство графиков «lattice» строится вызовом одной функции, и после создания графика его сложно изменить, например, добавив подписи осей, и нельзя дополнить новой порцией точек или кривых. С другой стороны, графические параметры в «lattice» настроены заранее и оптимизированы для представления нескольких графиков на одной панели.

3. **Система «ggplot»** (пакет **«ggplot2»** [7]). Реализует грамматику графиков, предложенную Леландом Уилкинсоном. График собирается как «предложение» из нескольких функций, первая из которых задаёт исходные данные («существительное»), вторая – их представление («глагол»), остальные добавляют на график дополнительные слои («прилагательные»). Как и в «lattice», система автоматически управляет графическими параметрами. Также функции «ggplot2»

---

[3] Руководство по стилю программирования на R. URL: https://drive.google.com/open?id=1zS-jyjokHEL_IjCPaKPPbrJU89VHzzae

[4] Roger D. Peng. Материалы курса «Exploratory Data Analysis» Университета Джонса Хопкинса на портале coursera.org, доступные в репозитории на github.com: https://github.com/rdpeng/courses/tree/master/04_ExploratoryAnalysis

[5] R Core Team (2015). R: A language and environment for statistical computing. R Foundation for Statistical Computing, Vienna, Austria. URL: https://www.R-project.org/

[6] Sarkar, Deepayan (2008) Lattice: Multivariate Data Visualization with R. Springer, New York. ISBN 978-0-387-75968-5

[7] H. Wickham. ggplot2: Elegant Graphics for Data Analysis. Springer-Verlag New York, 2009

могут трансформировать данные (например, шкалы показателей) и изменять их под различные координатные системы.

Функции трёх графических систем не сочетаются друг с другом. Поэтому прежде чем строить график, стоит понять, какие принципы изображения необходимы в данном случае:

1. На начальной, разведочной, стадии анализа, для представления данных, не сгруппированных по категориям, подходит «base».

2. Система «lattice» поможет представить данные с переменными-факторами, представив графики по категориям на отдельных панелях.

3. Система «ggplot2» построит более сложные визуализации, например, с пересчётом в другие координаты; нанесёт данные на географическую карту; сохранит график со всеми настройками в виде отдельного объекта в рабочем пространстве R.

В R существует много готовых функций на базе перечисленных пакетов, которые служат для создания специальных графиков. Некоторые из них:

- мозаичные графики и графики ассоциаций для визуализации категориальных данных: пакет «vcd» [8];
- самоорганизующиеся карты Кохонена: пакет «kohonen» [9];
- «тепловые карты» для визуализации схожести объектов и результатов кластерного анализа: пакет «gplots» [10];
- интерактивные трёхмерные графики: пакет «rgl» [11];
- фазовые плоскости одно- и двумерных автономных систем дифференциальных уравнений: пакет «phaseR» [12].

Далее мы рассмотрим работу с графическими системами «lattice» и «ggplot2».

---

[8] Блог Р.Кабакова Quick-R: Visualizing Categorical Data. URL: http://www.statmethods.net/advgraphs/mosaic.html

[9] Self-Organising Maps for Customer Segmentation using R / R-bloggers.com. URL: http://www.r-bloggers.com/self-organising-maps-for-customer-segmentation-using-r/

[10] Sebastian Raschka. A short tutorial for decent heat maps in R. URL: http://sebastianraschka.com/Articles/heatmaps_in_r.html

[11] Garrett Grolemund. Quick list of useful R packages / support.rstudio.com. URL: https://support.rstudio.com/hc/en-us/articles/201057987-Quick-list-of-useful-R-packages

[12] Markus Gesmann. Phase plane analysis in R / magesblog.com. URL: http://www.magesblog.com/2014/11/phase-plane-analysis-in-r.html

Предварительный анализ и визуализация данных по импорту за 2019 год

Загрузим пакеты, которые понадобятся нам в этой лабораторной.

```
library('dplyr')            # функции манипуляции данными
#>
#> Присоединяю пакет: 'dplyr'
#> Следующие объекты скрыты от 'package:stats':
#>
#>     filter, lag
#> Следующие объекты скрыты от 'package:base':
#>
#>     intersect, setdiff, setequal, union
library('data.table')       # объект "таблица данных"
#>
#> Присоединяю пакет: 'data.table'
#> Следующие объекты скрыты от 'package:dplyr':
#>
#>     between, first, last
library('WDI')              # загрузка данных из базы Всемирного банка
library('ggplot2')          # графическая система ggplot2
library('lattice')          # графическая система lattice
library('GGally')           # матричные графики разброса
#> Registered S3 method overwritten by 'GGally':
#>   method from
#>   +.gg   ggplot2
```

Пример №1. Используем данные импорта в Уругвай в 2019 году по коду 86: товары, связанные с железнодорожным транспортом. Рассчитаем описательные статистики и построим несколько графиков, чтобы посмотреть на данные в разрезе периодов времени (месяцы 2019 года) и группы стран географического региона “Латинская Америка и Карибский бассейн”, к которой относится Уругвай.

Загрузка и очистка данных

Код ниже загружает данные из БД международной торговли https://comtrade.un.org/. Чтобы определить код страны, можно обратиться к справочнику по адресу https://comtrade.un.org/Data/cache/partnerAreas.json (ищем “Uruguay”).

```

# создаём директорию для данных, если она ещё не существует:
data.dir <- 'data'
if (!file.exists(data.dir)) dir.create(data.dir)

# создаём файл с логом загрузок, если он ещё не существует:
log.filename <- 'data/download.log'
if (!file.exists(log.filename)) file.create(log.filename)

# Загрузить из базы данных международной торговли статистику импорта за 2019
#  год (данные ежемесячные) по стране и кодам, указанным в варианте.
# функция, реализующая API (источник: UN COMTRADE)
```

```
source("https://raw.githubusercontent.com/aksyuk/R-
data/master/API/comtrade_API.R")

# имя файла для сохранения
dest.file <- './data/comtrade_import-2019_raw.csv'
if (!file.exists(dest.file)) {
    # загрузить и сохранить файл
    s1 <- get.Comtrade(r = 'all', p = '858',
                       ps = as.character(2019), freq = "M",
                       rg = '1', cc = '86',
                       fmt = 'csv')
    write.csv(s1$data, fileName, row.names = F)
    # сделать запись в лог
    write(paste('Файл', dest.file, 'загружен', Sys.time()),
          file = log.filename, append = T)
}
```

Чтобы не превысить ограничение на загрузку по API базы данных, имеет смысл скачать статистику один раз, записать её на диск, а при повторном запуске скрипта загружать данные из сохранённого файла.

```
# читаем загруженные данные
DF.01 <- read.csv(dest.file, stringsAsFactors = F)
```

Посмотрим на размерность, структуру фрейма данных и выведем имена столбцов.

```
dim(DF.01)
#> [1] 22 35
str(DF.01)
#> 'data.frame':    22 obs. of  35 variables:
#>  $ Classification        : chr  "HS" "HS" "HS" "HS" ...
#>  $ Year                  : int  2019 2019 2019 2019 2019 2019 2019 2019 2019
2019 ...
#>  $ Period                : int  201901 201901 201903 201907 201912 201910
201903 201912 201907 201901 ...
#>  $ Period.Desc.          : chr  "January 2019" "January 2019" "March 2019"
"July 2019" ...
#>  $ Aggregate.Level       : int  2 2 2 2 2 2 2 2 2 2 ...
#>  $ Is.Leaf.Code          : int  0 0 0 0 0 0 0 0 0 0 ...
#>  $ Trade.Flow.Code       : int  1 1 1 1 1 1 1 1 1 1 ...
#>  $ Trade.Flow            : chr  "Imports" "Imports" "Imports" "Imports" ...
#>  $ Reporter.Code         : int  56 97 97 97 97 152 276 276 528 600 ...
#>  $ Reporter              : chr  "Belgium" "EU-28" "EU-28" "EU-28" ...
#>  $ Reporter.ISO          : logi  NA NA NA NA NA NA ...
#>  $ Partner.Code          : int  858 858 858 858 858 858 858 858 858 858 ...
#>  $ Partner               : chr  "Uruguay" "Uruguay" "Uruguay" "Uruguay" ...
#>  $ Partner.ISO           : logi  NA NA NA NA NA NA ...
#>  $ X2nd.Partner.Code     : logi  NA NA NA NA NA NA ...
#>  $ X2nd.Partner          : logi  NA NA NA NA NA NA ...
#>  $ X2nd.Partner.ISO      : logi  NA NA NA NA NA NA ...
#>  $ Customs.Proc..Code    : logi  NA NA NA NA NA NA ...
#>  $ Customs               : logi  NA NA NA NA NA NA ...
#>  $ Mode.of.Transport.Code: logi  NA NA NA NA NA NA ...
```

```
#>  $ Mode.of.Transport     : logi  NA NA NA NA NA NA ...
#>  $ Commodity.Code        : int  86 86 86 86 86 86 86 86 86 86 ...
#>  $ Commodity             : chr  "Railway, tramway locomotives, rolling-stock
and parts thereof; railway or tramway track fixtures and fittings a"|
__truncated__ "Railway, tramway locomotives, rolling-stock and parts thereof;
railway or tramway track fixtures and fittings a"| __truncated__ "Railway, tramway
locomotives, rolling-stock and parts thereof; railway or tramway track fixtures
and fittings a"| __truncated__ "Railway, tramway locomotives, rolling-stock and
parts thereof; railway or tramway track fixtures and fittings a"| __truncated__
...
#>  $ Qty.Unit.Code         : logi  NA NA NA NA NA NA ...
#>  $ Qty.Unit              : logi  NA NA NA NA NA NA ...
#>  $ Qty                   : logi  NA NA NA NA NA NA ...
#>  $ Alt.Qty.Unit.Code     : logi  NA NA NA NA NA NA ...
#>  $ Alt.Qty.Unit          : logi  NA NA NA NA NA NA ...
#>  $ Alt.Qty               : logi  NA NA NA NA NA NA ...
#>  $ Netweight..kg.        : int  0 0 0 0 0 NA 0 0 0 0 ...
#>  $ Gross.weight..kg.     : logi  NA NA NA NA NA NA ...
#>  $ Trade.Value..US..     : int  1578 1578 6387 891 6444 50434 6387 6444 891
6242 ...
#>  $ CIF.Trade.Value..US.. : logi  NA NA NA NA NA NA ...
#>  $ FOB.Trade.Value..US.. : logi  NA NA NA NA NA NA ...
#>  $ Flag                  : int  0 0 0 0 0 0 0 0 0 0 ...
colnames(DF.01)
#>  [1] "Classification"         "Year"                   "Period"
#>  [4] "Period.Desc."           "Aggregate.Level"        "Is.Leaf.Code"
#>  [7] "Trade.Flow.Code"        "Trade.Flow"             "Reporter.Code"
#> [10] "Reporter"               "Reporter.ISO"           "Partner.Code"
#> [13] "Partner"                "Partner.ISO"            "X2nd.Partner.Code"
#> [16] "X2nd.Partner"           "X2nd.Partner.ISO"       "Customs.Proc..Code"
#> [19] "Customs"                "Mode.of.Transport.Code" "Mode.of.Transport"
#> [22] "Commodity.Code"         "Commodity"              "Qty.Unit.Code"
#> [25] "Qty.Unit"               "Qty"                    "Alt.Qty.Unit.Code"
#> [28] "Alt.Qty.Unit"           "Alt.Qty"                "Netweight..kg."
#> [31] "Gross.weight..kg."      "Trade.Value..US.."      "CIF.Trade.Value..US.."
#> [34] "FOB.Trade.Value..US.."  "Flag"
```

Как видно, в именах столбцов много точек: так R заменяет символы, которые не могут входить в имена переменных (в нашем случае, пробелы и символ доллара). Очистим заголовки фрейма от повторяющихся точек, а также от точек в конце имени столбца, используюя функцию `gsub()`, которая относится к семейству функций для поиска и замены символов в строках.

В R для поиска и замены подстрок в символьных векторах служат функции:

- `grep(<что_ищем>', <где_ищем>')` – функция просматривает символьный вектор и возвращает номера тех элементов, в которых встречается подстрока .
- `grep(<что_ищем>', <где_ищем>', value = T)` – возвращает значения (аргумент value = TRUE) элементов, в которых встречается подстрока.
- `grepl(<что_ищем>', <где_ищем>')` – функция делает то же, что и `grep()`, однако возвращает логический вектор, элементы которого равны `TRUE`, если в

соответствующем элементе символьного вектора встречается подстрока, и FALSE в противном случае.

- sub(<*что_ищем*>', <*на_что_заменяем*>', <*где_ищем*>') – функция замены первого вхождения подстроки на другую последовательность символов. Если второй аргумент пустой, подстрока будет удалена.

- gsub(<*что_ищем*>', <*на_что_заменяем*>', <*где_ищем*>') – то же, что и sub(), но ищет и заменяет все вхождения подстроки в исходной строке [13].

```
# копируем имена в символьный вектор, чтобы ничего не испортить
nms <- colnames(DF.01)
# заменить серии из двух и более точек на одну
nms <- gsub('[.]+', '.', nms)
# убрать все хвостовые точки
nms <- gsub('[.]+$', '', nms)
# заменить US на USD
nms <- gsub('Trade.Value.US', 'Trade.Value.USD', nms)
# проверяем, что всё получилось, и заменяем имена столбцов
colnames(DF.01) <- nms
# результат обработки имён столбцов
colnames(DF.01)
#>  [1] "Classification"         "Year"                   "Period"
#>  [4] "Period.Desc"            "Aggregate.Level"        "Is.Leaf.Code"
#>  [7] "Trade.Flow.Code"        "Trade.Flow"             "Reporter.Code"
#> [10] "Reporter"               "Reporter.ISO"           "Partner.Code"
#> [13] "Partner"                "Partner.ISO"            "X2nd.Partner.Code"
#> [16] "X2nd.Partner"           "X2nd.Partner.ISO"       "Customs.Proc.Code"
#> [19] "Customs"                "Mode.of.Transport.Code" "Mode.of.Transport"
#> [22] "Commodity.Code"         "Commodity"              "Qty.Unit.Code"
#> [25] "Qty.Unit"               "Qty"                    "Alt.Qty.Unit.Code"
#> [28] "Alt.Qty.Unit"           "Alt.Qty"                "Netweight.kg"
#> [31] "Gross.weight.kg"        "Trade.Value.USD"        "CIF.Trade.Value.USD"
#> [34] "FOB.Trade.Value.USD"    "Flag"
```

Чтобы привести таблицу с данными к аккуратному виду, нам также понадобится убрать полностью пустые столбцы.

```
# делаем подсчёт пропусков по каждому столбцу
na.num <- sapply(DF.01, function(x) sum(is.na(x)))
na.num
#>         Classification                   Year                 Period
#>                      0                      0                      0
#>            Period.Desc        Aggregate.Level           Is.Leaf.Code
#>                      0                      0                      0
#>        Trade.Flow.Code             Trade.Flow          Reporter.Code
#>                      0                      0                      0
```

[13] Jeffrey Leek. Материалы курса “Getting and Cleaning Data” Университета Джонса Хопкинса на портале coursera.org, доступные в репозитории на github.com: https://github.com/jtleek/modules/tree/master/03_GettingData

```
#>               Reporter           Reporter.ISO           Partner.Code
#>                      0                     22                      0
#>                Partner            Partner.ISO      X2nd.Partner.Code
#>                      0                     22                     22
#>           X2nd.Partner       X2nd.Partner.ISO      Customs.Proc.Code
#>                     22                     22                     22
#>                Customs Mode.of.Transport.Code      Mode.of.Transport
#>                     22                     22                     22
#>         Commodity.Code              Commodity          Qty.Unit.Code
#>                      0                      0                     22
#>               Qty.Unit                    Qty      Alt.Qty.Unit.Code
#>                     22                     22                     22
#>           Alt.Qty.Unit                Alt.Qty           Netweight.kg
#>                     22                     22                      1
#>        Gross.weight.kg        Trade.Value.USD    CIF.Trade.Value.USD
#>                     22                      0                     22
#>    FOB.Trade.Value.USD                   Flag
#>                     22                      0

# в каких столбцах все наблюдения пропущены?
col.remove <- na.num == nrow(DF.01)
names(col.remove)[col.remove == T]
#>  [1] "Reporter.ISO"           "Partner.ISO"            "X2nd.Partner.Code"
#>  [4] "X2nd.Partner"           "X2nd.Partner.ISO"       "Customs.Proc.Code"
#>  [7] "Customs"                "Mode.of.Transport.Code" "Mode.of.Transport"
#> [10] "Qty.Unit.Code"          "Qty.Unit"               "Qty"
#> [13] "Alt.Qty.Unit.Code"      "Alt.Qty.Unit"           "Alt.Qty"
#> [16] "Gross.weight.kg"        "CIF.Trade.Value.USD"    "FOB.Trade.Value.USD"

# уберём эти столбцы из таблицы
DF.01 <- DF.01[, !col.remove]
dim(DF.01)
#> [1] 22 17
```

Для удобства манипулирования данными, создадим из нашего фрейма `DF.01` объект `DT.import` типа “таблица данных” (`data.table`).

```
DT.import <- data.table(DF.01)
```

У объектов типа `data.table` есть несколько преимуществ перед обычными фреймами:

1. Компактное отображение в консоли. Если число строк таблицы велико, будут автоматически показаны только часть верхних и нижних строк.

2. Больше возможностей у оператора `[]`. В квадратных скобках можно делать более лаконичное присваивание столбцам таблицы, кроме того, чтобы обратиться к столбцу, не нужно заново писать имя таблицы и символ `$`.

3. Специальные выражения для оператора квадратных скобок `[]`: например, можно подсчитать количество строк с помощью `.N`, найти индекс строк по условию с помощью `.I`, обратиться к подвыборке из таблицы с помощью `.SD`.

Подробнее о работе с объектами `data.table` и функциями манипулирования данными из пакета `dplyr` см. в разделе “Дополнительно” данного руководства.

Воспользуемся функцией `select()` для выбора только нужных столбцов таблицы.

```
# столбцы таблицы
colnames(DT.import)
#>  [1] "Classification"  "Year"            "Period"          "Period.Desc"
#>  [5] "Aggregate.Level" "Is.Leaf.Code"    "Trade.Flow.Code" "Trade.Flow"
#>  [9] "Reporter.Code"   "Reporter"        "Partner.Code"    "Partner"
#> [13] "Commodity.Code"  "Commodity"       "Netweight.kg"    "Trade.Value.USD"
#> [17] "Flag"
# оставляем только нужные столбцы
DT.import <- select(DT.import, Year, Period, Trade.Flow, Reporter, Partner,
                    Commodity.Code, Trade.Value.USD)
```

В этой таблице нет групп стран по регионам и уровню дохода. Присоединим их из справочника из пакета `WDI`.

```
# список стран из справочника БД Всемирного банка
DT.country <- data.table(WDI_data$country)
DT.country <- select(DT.country, iso2c, country, region, income)
```

Нам понадобятся категории для стран-поставщиков (столбец Reporter), поэтому изменим названия нужных столбцов, добавив к ним `'Reporter.'`.

```
# добавляем 'Reporter.' в названия столбцов с характеристиками стран,
#  чтобы отличать продавцов от покупателя
colnames(DT.country) <- paste0('Reporter.', colnames(DT.country))
# добавляем столбцы с группами стран-продавцов
DT.import <- merge(DT.import, DT.country,
                   by.x = 'Reporter', by.y = 'Reporter.country')
colnames(DT.import)
#>  [1] "Reporter"        "Year"            "Period"          "Trade.Flow"
#>  [5] "Partner"         "Commodity.Code"  "Trade.Value.USD" "Reporter.iso2c"
#>  [9] "Reporter.region" "Reporter.income"
dim(DT.import)
#> [1] 18 10
```

Наконец, последний этап очистки данных: перекодируем названия периодов времени (месяцев 2019 года), чтобы они не воспринимались как непрерывные числовые показатели, и легче воспринимались на графике. Из формата `ГГГГММ` сделаем формат `ГГГГ-ММ`, а затем изменим тип столбца на фактор (т.е. категориальную переменную с пронумерованными уникальными значениями).

```
# наконец, меняем формат периода времени с ГГГГММ на ГГГГ-ММ
DT.import[, Period := as.character(Period)]
DT.import[, Period := paste0(substr(Period, 1, 4), '-', substr(Period, 5, 7))]
DT.import[, Period := factor(Period,
                             levels = paste0('2019-', formatC(1:12, width = 2,
                                                              flag = '0')))]
```

Описательные статистики и визуализация

Для начала посмотрим, как работает функция `summary()` применительно к столбцам таблицы различных типов.

```
# описательные статистики по таблице данных
summary(DT.import)
#>    Reporter              Year          Period   Trade.Flow
#>  Length:18          Min.   :2019   2019-01:3   Length:18
#>  Class :character   1st Qu.:2019   2019-02:2   Class :character
#>  Mode  :character   Median :2019   2019-03:2   Mode  :character
#>                     Mean   :2019   2019-07:2
#>                     3rd Qu.:2019   2019-10:2
#>                     Max.   :2019   2019-12:2
#>                                    (Other):5
#>    Partner           Commodity.Code Trade.Value.USD Reporter.iso2c
#>  Length:18          Min.   :86     Min.   :  891   Length:18
#>  Class :character   1st Qu.:86     1st Qu.: 1492   Class :character
#>  Mode  :character   Median :86     Median : 6416   Mode  :character
#>                     Mean   :86     Mean   :15027
#>                     3rd Qu.:86     3rd Qu.:12803
#>                     Max.   :86     Max.   :87915
#>
#>  Reporter.region    Reporter.income
#>  Length:18          Length:18
#>  Class :character   Class :character
#>  Mode  :character   Mode  :character
#>
#>
#>
#>
```

Можно видеть, что для столбцов-факторов (`Period`) `summary()` выдаёт количество уникальных значений, для непрерывных количественных (`Trade.Value.USD`) считает описательные статистики, а для категориальных (например, `Reporter`) выдаёт только класс. При наличии пропусков в столбце `summary()` подсчитает их количество. Обратим внимание, что столбец `Commodity.Code` воспринят функцией как количественный, что неверно, поскольку код товара по гармонизированной классификации это не более чем уникальный код, как ИНН организации.

Из всех столбцов нас интересует количественный `Trade.Value.USD`, поэтому имеет смысл посмотреть на его описательные статистики подробнее.

```
# описательные статистики по количественному показателю
summary(DT.import$Trade.Value.USD)
#>    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
#>     891    1492    6416   15027   12803   87915

# суммарная стоимость импорта в Уругвай в 2019 году по коду 86
sum(DT.import$Trade.Value.USD)
#> [1] 270487
```

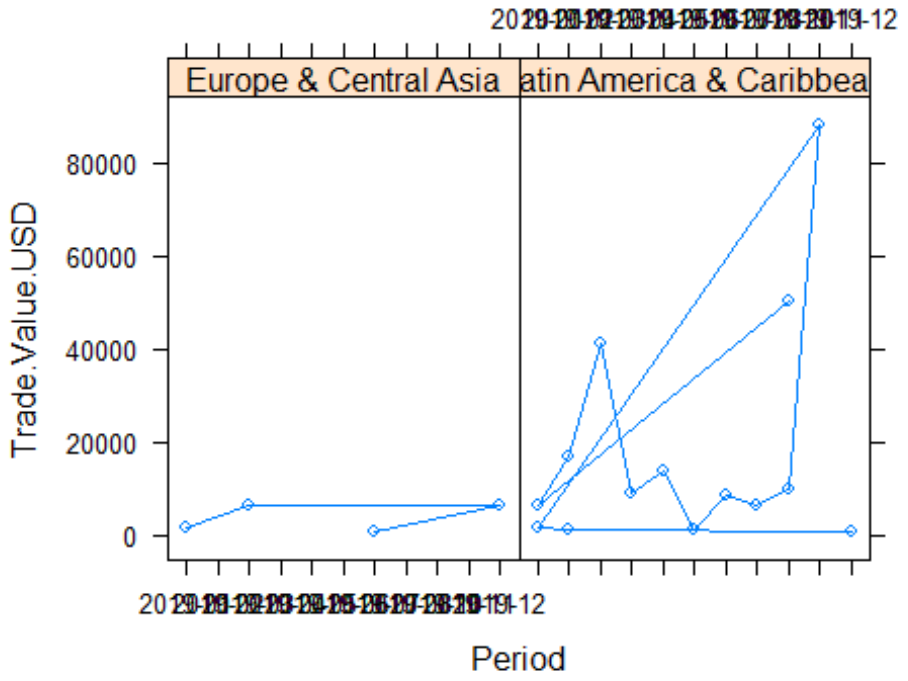
Далее будем строить графики в разрезе регионов, поэтому стоит установить, к каким категориям стран относится интересующая нас страна – Уругвай.

```
# к каким категориям относится Уругвай
filter(DT.country, Reporter.country == 'Uruguay')
#>    Reporter.iso2c Reporter.country           Reporter.region Reporter.income
#> 1:             UY         Uruguay Latin America & Caribbean     High income
```

График динамики (графическая система `lattice`)

Посмотрим, как менялась стоимость поставок в Уругвай по коду товара 86 в течение 2019 год. Система `lattice` отличается тем, что для построения графика достаточно вызова одной функции, первый аргумент которой – формула взаимосвязи переменных. Построим несколько графиков динамики на одном полотне, по регионам, к которым относятся страны-поставщики.

```
# так получается неверное отображение данных
xyplot(Trade.Value.USD ~ Period | Reporter.region, data = DT.import,
       type = 'o')
```



Несмотря на то, что функция запсиана верно, график неверно отображает данные: можно видеть, что позиции перепутаны, и месяцы не идут последовательно. Чтобы это исправить, создадим набор данных специально под этот график, где агрегируем и отсортируем значения по месяцам и регионам.

```
# чтобы отразить динамику правильно, нужно агрегировать стоимость поставок
#  по периоду и по региону мира
group_by(DT.import, Period, Reporter.region) %>%
    summarise(Trade.Value.USD = sum(Trade.Value.USD)) -> dt.plot.01
```

```
#> `summarise()` has grouped output by 'Period'. You can override using the
`.groups` argument.
```

В этом коде мы объединили несколько функций в один вызов с помощью оператора потока %>%. Он позволяет визуально выстроить последовательность действий, и сократить код за счёт того, что возвращаемое значение предыдущей функции автоматически становится первым аргументом следующей функции. Кроме того, чтобы сохранить результат операций в таблицу `dt.plot.01`, мы использовали оператор присваивания слева направо ->.

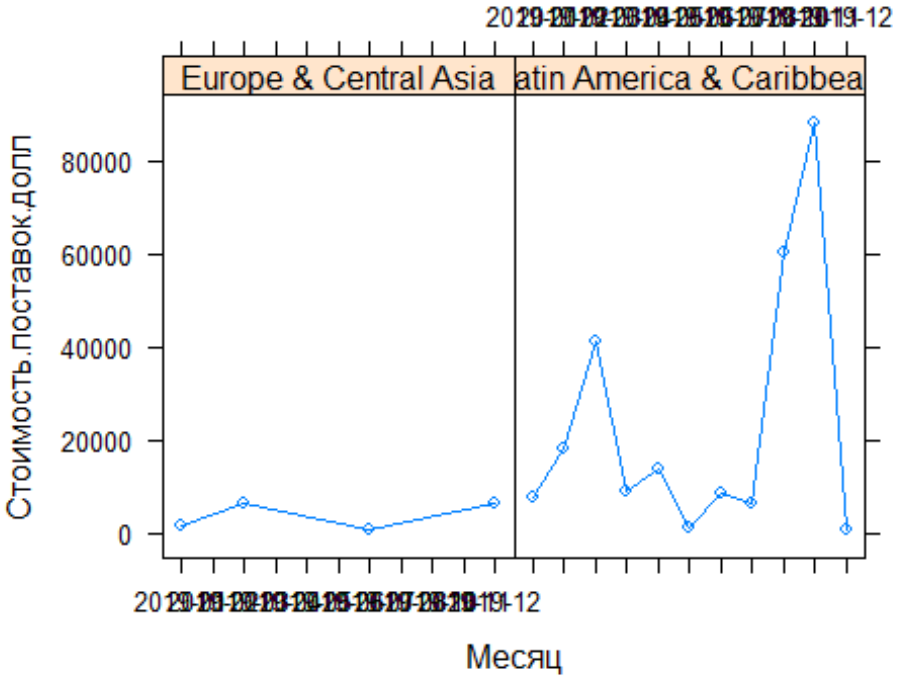
Назовём столбцы в новом наборе данных по-русски, чтобы график читался без пояснений.

```
colnames(dt.plot.01)[colnames(dt.plot.01) == 'Reporter.region'] <-
    'Регион.поставщика'
# для отображения на графиках называем столбцы по-русски
colnames(dt.plot.01)[colnames(dt.plot.01) == 'Period'] <- 'Месяц'
colnames(dt.plot.01)[colnames(dt.plot.01) == 'Trade.Value.USD'] <-
    'Стоимость.поставок.долл'

# результат: набор данных для первого графика
dt.plot.01
#> # A tibble: 15 x 3
#> # Groups:   Месяц [11]
#>    Месяц   Регион.поставщика         Стоимость.поставок.долл
#>    <fct>   <chr>                                       <int>
#>  1 2019-01 Europe & Central Asia                        1578
#>  2 2019-01 Latin America & Caribbean                    7706
#>  3 2019-02 Latin America & Caribbean                   18009
#>  4 2019-03 Europe & Central Asia                        6387
#>  5 2019-03 Latin America & Caribbean                   41073
#>  6 2019-04 Latin America & Caribbean                    9166
#>  7 2019-05 Latin America & Caribbean                   13810
#>  8 2019-06 Latin America & Caribbean                    1308
#>  9 2019-07 Europe & Central Asia                         891
#> 10 2019-07 Latin America & Caribbean                    8764
#> 11 2019-08 Latin America & Caribbean                    6237
#> 12 2019-10 Latin America & Caribbean                   60215
#> 13 2019-11 Latin America & Caribbean                   87915
#> 14 2019-12 Europe & Central Asia                        6444
#> 15 2019-12 Latin America & Caribbean                     984
```

Теперь график будет выглядеть корректно (обратите внимание: изменился аргумент `data`).

```
# теперь получается верный график динамики
xyplot(Стоимость.поставок.долл ~ Месяц | Регион.поставщика, data = dt.plot.01,
       type = 'o')
```



Графики с долями (графическая система `ggplot2`)

Круговая

Графическая система `ggplot2` предлагает другой подход: здесь график получается как результат сложения вызовов неограниченного количества функций. На первом месте всегда стоит функция-подлежащее `ggplot()`, которая определяет источник данных и роли переменных. На втором – функция-сказуемое, которая указывает, какую геометрию для изображения данных следует использовать; названия функций-сказуемых начинаются с `geom_`. Наконец, далее идут разнообразные функции-дополнения, которые задают оформление графика. При этом функции-дополнения не всегда необходимы, поскольку даже по умолчанию `ggplot2` предлагает красивое оформление графиков.

Для начала подготовим данные: теперь нам понадобятся данные, агрегированные только по региону страны-поставщика.

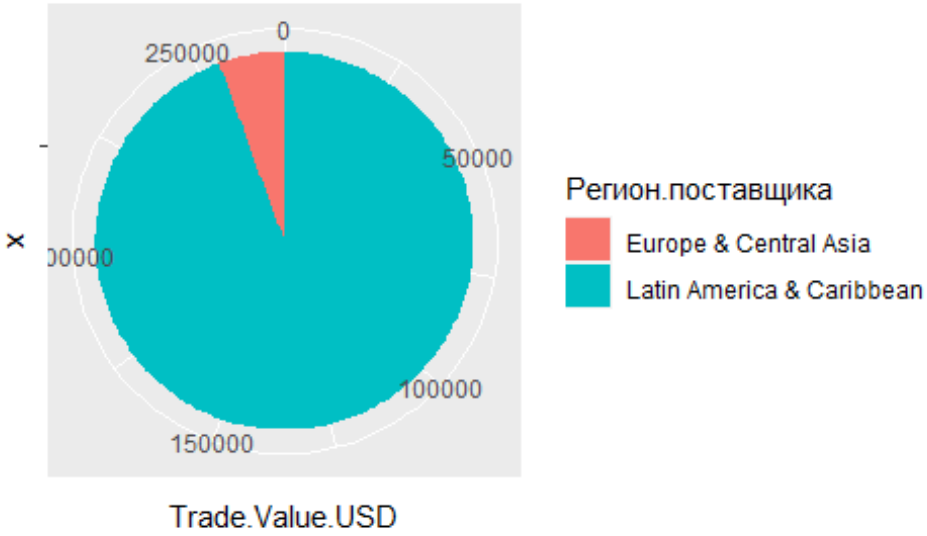
```
# готовим данные для второго графика
group_by(DT.import, Reporter.region) %>%
    summarise(Trade.Value.USD = sum(Trade.Value.USD)) -> dt.plot.02
dt.plot.02$Trade.Value.share <- paste0(round(dt.plot.02$Trade.Value.USD /
    sum(dt.plot.02$Trade.Value.USD) * 100, 1), '%')
# для отображения на графиках называем столбцы по-русски
colnames(dt.plot.02)[colnames(dt.plot.02) == 'Reporter.region'] <-
    'Регион.поставщика'

# результат: набор данных для второго графика
dt.plot.02
#> # A tibble: 2 x 3
#>   Регион.поставщика         Trade.Value.USD Trade.Value.share
```

```
#>   <chr>                                <int> <chr>
#> 1 Europe & Central Asia                15300 5.7%
#> 2 Latin America & Caribbean           255187 94.3%
```

Круговые диаграммы это первый тип диаграмм в Excel, однако в продвинутом анализе данных они используются не так часто. Поэтому в ggplot2 нет отдельной геометрии для круговой диаграммы, но можно взять столбчатую (bartplot) и “свернуть” её в полярных координатах.

```
# исходные данные для графика и роли столбцов (x пустой)
gp <- ggplot(data = dt.plot.02, aes(x = '', y = Trade.Value.USD,
                                    fill = Регион.поставщика,
                                    label = Trade.Value.share))
# добавляем геометрию: столбчатая диаграмма
#  + полярные координаты, чтобы свернуть ряд данных в окружность
gp <- gp + geom_bar(stat = 'identity', width = 1) + coord_polar('y', start = 0)
gp
```

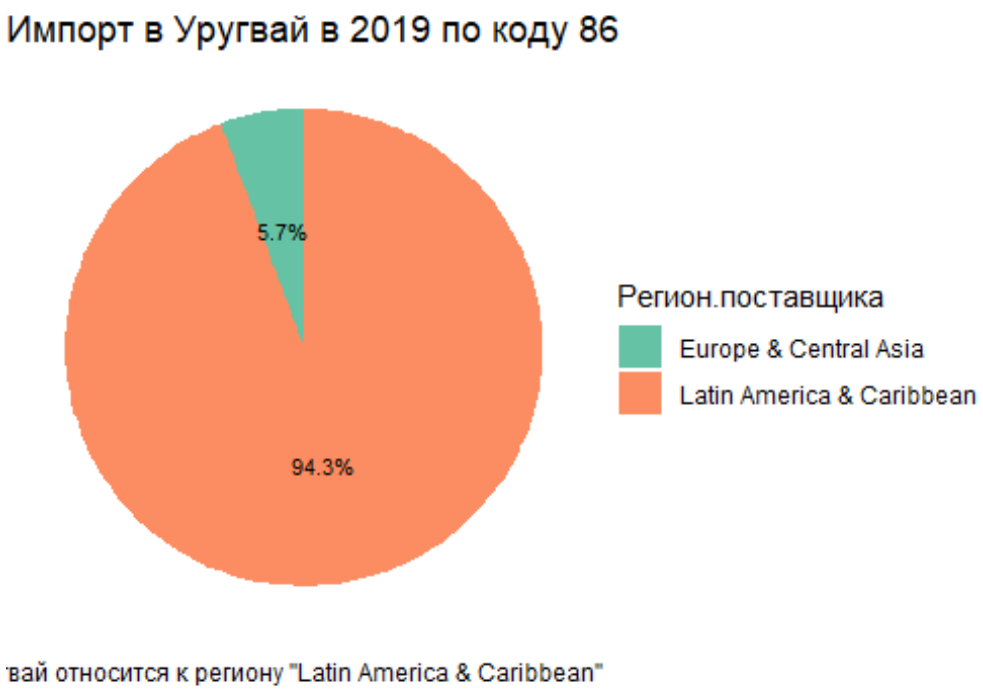


Поясним роли переменных в аргументе aes функции ggplot():

1. x – показатель, который играет роль координаты по горизонтали;
2. y – показатель с координатами по вертикали;
3. fill – показатель, в зависимости от значений которого точки, или сектора, или столбцы на графике будут отличаться цветом;
4. label – показатель, значения которого будут играть роль подписей данных.

Мы использовали только фукнцию-подлежащее и функцию-сказуемое (geom_bar()). На итоговом графике много ненужной информации: шкала по кругу, подписи осей. В то же время, не подписаны доли на секторах. Исправим это, добавив функции-дополнения.

```
# добавляем подписи секторов
gp <- gp + geom_text(size = 3, position = position_stack(vjust = 0.5))
# меняем палитру графика
gp <- gp + scale_fill_brewer(palette = 'Set2')
# добавляем заголовок
gp <- gp + ggtitle('Импорт в Уругвай в 2019 по коду 86')
# добавляем подпись под графиком
gp <- gp + labs(caption = 'Уругвай относится к региону "Latin America & 
Caribbean"')
# убираем разметку шкалы
gp <- gp + theme_void()
# отображаем график
gp
```



В таком виде график подходит для вставки в отчёт или в презентацию. В ходе настройки мы изменили в том числе палитру цветов (функция scale_fill_brewer). Посмотреть варианты палитр для различных типов данных можно, например, на странице: https://www.r-graph-gallery.com/38-rcolorbrewers-palettes.html.

Столбчатая с долями и накоплением

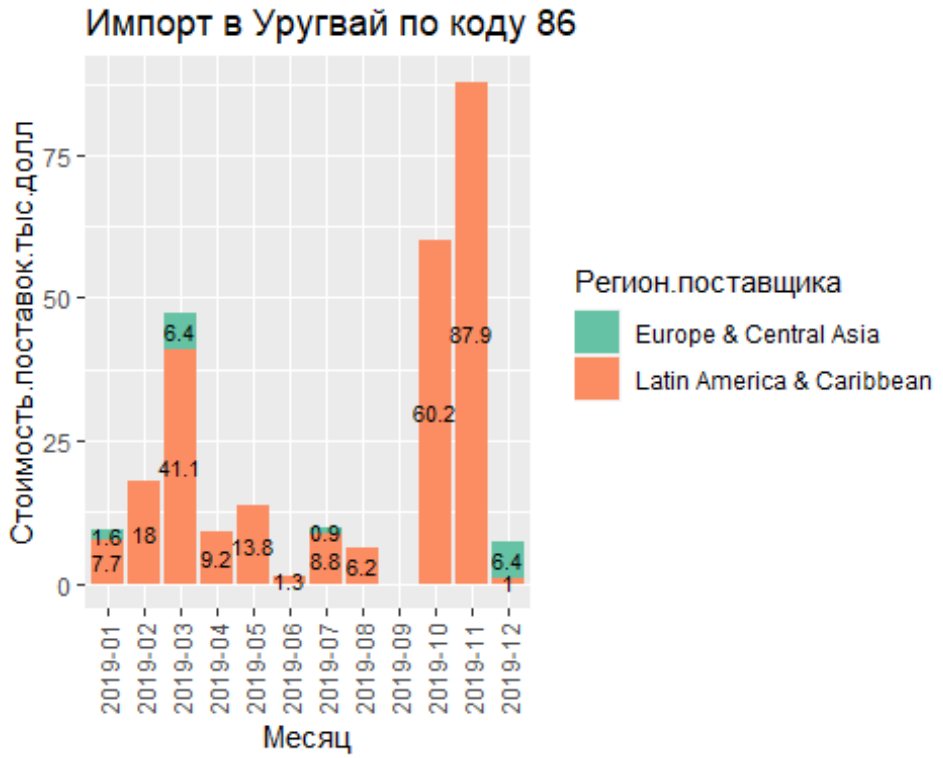
Используем агрегированные данные для первого графика, только теперь сопоставим доли и пересчитаем стоимость в тысячи долларов.

```
dt.plot.01$Стоимость.поставок.долл <-
    round(dt.plot.01$Стоимость.поставок.долл / 1000, 1)
```

```
colnames(dt.plot.01)[colnames(dt.plot.01) == 'Стоимость.поставок.долл'] <-
    'Стоимость.поставок.тыс.долл'
```

Полный код построения графика показан ниже. Благодаря функции `scale_x_discrete(drop = F)` на горизонтальной шкале графика будут отображены и те месяцы, в которые поставок по нашему коду товара не было.

```
gp <- ggplot(data = dt.plot.01, aes(x = Месяц, y = Стоимость.поставок.тыс.долл,
                                    fill = Регион.поставщика,
                                    label = Стоимость.поставок.тыс.долл))
# добавляем геометрию: столбчатая диаграмма, столбики друг на друге
gp <- gp + geom_bar(stat = 'identity')
# добавляем подписи значений
gp <- gp + geom_text(size = 3, position = position_stack(vjust = 0.5))
# настройки горизонтальной оси
gp <- gp + scale_x_discrete(drop = F)
# меняем палитру графика
gp <- gp + scale_fill_brewer(palette = 'Set2')
# разворачиваем подписи по горизонтальной оси на 90 градусов
gp <- gp + theme(axis.text.x = element_text(angle = 90, vjust = 0.5, hjust = 1))
# добавляем заголовок
gp <- gp + ggtitle('Импорт в Уругвай по коду 86')
# отображаем график
gp
```



Топ-5 стран-поставщиков

На последнем шаге нашего дескриптивного анализа получим топ-5 стран-поставщиков по коду 86 в Уругвай в 2019 году. Для начала по всему множеству стран мира.

```
# по всем странам
group_by(DT.import, Reporter) %>%
    summarise(Trade.Value.USD = sum(Trade.Value.USD)) %>%
    arrange(-Trade.Value.USD) -> df.top
# первые 5 строк отсортированной таблицы
df.top[1:5, ]
#> # A tibble: 5 x 2
#>   Reporter Trade.Value.USD
#>   <chr>              <int>
#> 1 Paraguay          201093
#> 2 Chile              50434
#> 3 Germany            12831
#> 4 Peru                3660
#> 5 Belgium             1578
```

Далее, посмотрим на топ-5 таких стран из географического региона Уругвая.

```
# по странам латиноамериканского региона
group_by(filter(DT.import, Reporter.region == 'Latin America & Caribbean'),
         Reporter) %>%
    summarise(Trade.Value.USD = sum(Trade.Value.USD)) %>%
    arrange(-Trade.Value.USD) -> df.top.region
# первые 5 строк отсортированной таблицы
df.top.region[1:5, ]
#> # A tibble: 5 x 2
#>   Reporter Trade.Value.USD
#>   <chr>              <int>
#> 1 Paraguay          201093
#> 2 Chile              50434
#> 3 Peru                3660
#> 4 <NA>                  NA
#> 5 <NA>                  NA
```

Поскольку таких стран меньше 5, мы получили пустые строки в итоговой таблице. Чтобы этого не допустить, исправим последнюю строку:

```
# первые 5 строк (пустые не выводить)
df.top.region[1:min(nrow(df.top.region), 5), ]
#> # A tibble: 3 x 2
#>   Reporter Trade.Value.USD
#>   <chr>              <int>
#> 1 Paraguay          201093
#> 2 Chile              50434
#> 3 Peru                3660
```

Наконец, топ-5 (или сколько есть) стран-поставщиков из категории стран с высоким доходом:

```
# по странам с высоким доходом
group_by(filter(DT.import, Reporter.income == 'High income'),
         Reporter) %>%
    summarise(Trade.Value.USD = sum(Trade.Value.USD)) %>%
    arrange(-Trade.Value.USD) -> df.top.income
# первые 5 строк отсортированной таблицы
```

```
df.top.income[1:min(nrow(df.top.income), 5), ]
#> # A tibble: 4 x 2
#>   Reporter    Trade.Value.USD
#>   <chr>                 <int>
#> 1 Chile                 50434
#> 2 Germany               12831
#> 3 Belgium                1578
#> 4 Netherlands             891
```

Построение линейных регрессионных моделей для рейтинга лёгкости ведения бизнеса

Пример №2. Построим для стран с высоким и средне-высоким доходом (High income & Upper middle income) модели зависимости рейтинга лёгкости ведения бизнеса (это зависимая переменная, `IC.BUS.EASE.XQ` из базы Всемирного банка) от объясняющих показателей:

- `NY.GDP.PCAP.CD` – валовой внутренний продукт на душу населения в текущих ценах, долларов США (GDP per capita, current US$);
- `IC.REG.COST.PC.ZS` – затраты на создание бизнеса, % от валового национального дохода на душу населения (Cost of business start-up procedures, % of GNI per capita);
- `IC.REG.DURS` – время на создание бизнеса, дней (Time required to start a business, days);
- `IC.TAX.TOTL.CP.ZS` – налоговая нагрузка на бизнес, % от прибыли (Total tax and contribution rate, % of profit);
- `IC.TAX.DURS` – время на выплату налогов, часов (Time to prepare and pay taxes, hours).

Данные по всем старанам мы загрузили в лабораторной №1. Теперь сделаем выборку для решения нашеё задачи.

```
# коды и названия показателей по странам
ind.names <- c('NY.GDP.PCAP.CD', 'IC.REG.COST.PC.ZS',
               'IC.REG.DURS', 'IC.TAX.TOTL.CP.ZS',
               'IC.TAX.DURS', 'IC.BUS.EASE.XQ')
ind.labels <- c('ВВП на душу, текущие цены, USD',
                'Затраты на создание бизнеса, % от ВНД на душу',
                'Время на создание бизнеса, дней',
                'Налоговая нагрузка на бизнес, % от прибыли',
                'Время на выплату налогов, часов',
                'Рейтинг лёгкости ведения бизнеса')
names(ind.labels) <- ind.names

# читаем ранее скачанные данные за 2019 год
dest.file <- './data/wdi_2019.csv'
DT.wdi.2019 <- data.table(read.csv(dest.file, stringsAsFactors = F))

# смотрим первые строки таблицы
head(DT.wdi.2019)
#>    iso2c              country NY.GDP.PCAP.CD IC.REG.COST.PC.ZS IC.REG.DURS
```

```
#> 1:    AE United Arab Emirates     43103.3363              17.2         3.8
#> 2:    AF          Afghanistan       507.1034               6.8         8.5
#> 3:    AG  Antigua and Barbuda     17113.3498               8.0        19.0
#> 4:    AL              Albania      5355.8478              10.8         4.5
#> 5:    AM              Armenia      4622.7382               0.8         4.0
#> 6:    AO               Angola      2809.6261              11.1        36.0
#>    IC.TAX.TOTL.CP.ZS IC.TAX.DURS IC.BUS.EASE.XQ              income
#> 1:              15.9         116             16         High income
#> 2:              71.4         270            173          Low income
#> 3:              43.0         177            113         High income
#> 4:              36.6         252             82 Upper middle income
#> 5:              22.6         264             47 Upper middle income
#> 6:              49.1         287            177 Lower middle income
#>                        region
#> 1: Middle East & North Africa
#> 2:                 South Asia
#> 3:  Latin America & Caribbean
#> 4:      Europe & Central Asia
#> 5:      Europe & Central Asia
#> 6:         Sub-Saharan Africa
# и размерность таблицы
dim(DT.wdi.2019)
#> [1] 183  10
```

Чтобы отфильтровать страны по нужному уровню дохода, используем функцию `filter()` пакета `dplyr`.

```
# фильтруем страны по регионам из варианта
unique(DT.country$Reporter.income)
#> [1] "High income"         "Low income"          "Aggregates"
#> [4] "Lower middle income" "Upper middle income"
DT.wdi.2019 <- filter(DT.wdi.2019,
                      income %in% c('High income', 'Upper middle income'))
# размерность отфильтрованной таблицы
dim(DT.wdi.2019)
#> [1] 111  10
```

Очистим данные от пропусков. Здесь мы сталкиваемся с интересным моментом: ISO код Намибии – “NA” – совпадает с обозначением, принятым в R для пропущенных наблюдений. Чтобы не потерять наблюдение, вручную заменим этот технический “пропуск” на текстовое значение `"NA"`.

```
# сколько пропусков в столбцах таблицы
sapply(DT.wdi.2019, function(x){sum(is.na(x))})
#>             iso2c           country    NY.GDP.PCAP.CD IC.REG.COST.PC.ZS
#>                 1                 0                 0                 0
#>       IC.REG.DURS IC.TAX.TOTL.CP.ZS       IC.TAX.DURS    IC.BUS.EASE.XQ
#>                 0                 0                 0                 0
#>            income            region
#>                 0                 0

# код Намибии NA совпадает с меткой пропущенного наблюдения в R
DT.wdi.2019[is.na(DT.wdi.2019$iso2c), ]
```

```
#>    iso2c country NY.GDP.PCAP.CD IC.REG.COST.PC.ZS IC.REG.DURS IC.TAX.TOTL.CP.ZS
#> 1:  <NA> Namibia       5037.343               8.9          54              20.7
#>    IC.TAX.DURS IC.BUS.EASE.XQ              income              region
#> 1:         302            104 Upper middle income Sub-Saharan Africa
# исправляем это
DT.wdi.2019[is.na(DT.wdi.2019$iso2c), iso2c := 'NA']

# убираем строки с пропусками
dim(DT.wdi.2019)
#> [1] 111  10
DT.wdi.2019 <- na.omit(DT.wdi.2019)
dim(DT.wdi.2019)
#> [1] 111  10
```

В итоге в этой категории стран мы не потеряли ни одного наблюдения. Прежде чем строить модели, сделаем дескриптивный анализ из двух шагов:

1. Посчитаем коэффициенты вариации переменных по формуле $CV = \widehat{s_x}/\bar{x} \cdot 100\%$ (т.е. среднеквадратическое отклонение показателя разделить на его среднее значение). Коэффициент вариации может принимать любое значение (в т.ч. больше 100% и меньше 0%). Он показывает, насколько данные неоднородны. Однородным считается показатель с $CV < 33\%$, однако у реальных данных допустимо использовать в моделях переменные с более высокими значениями. Значения $CV > 200\%$ могут считаться рекомендацией к логарифмированию показателя.

2. Построим графики взаимного разброса переменных с парными линейными коэффициентами корреляции. Коэффициент корреляции лежит в пределах от -1 до 1 и показывает тесноту линейной взаимосвязи между двумя показателями. В линеной регрессии линейная связь между зависимой переменной и объясняющими должна быть значимой (значительно отличаться от 0 в большую или в меньшую сторону), а между объясняющими, наоборот, связь должна в идеале отсутствовать. Допускаются значимые линейные взаимосвязи между объясняющими переменными, если они не теснее связей с зависимой.

Ниже показан расчёт описательных статистик и коэффициентов вариации.

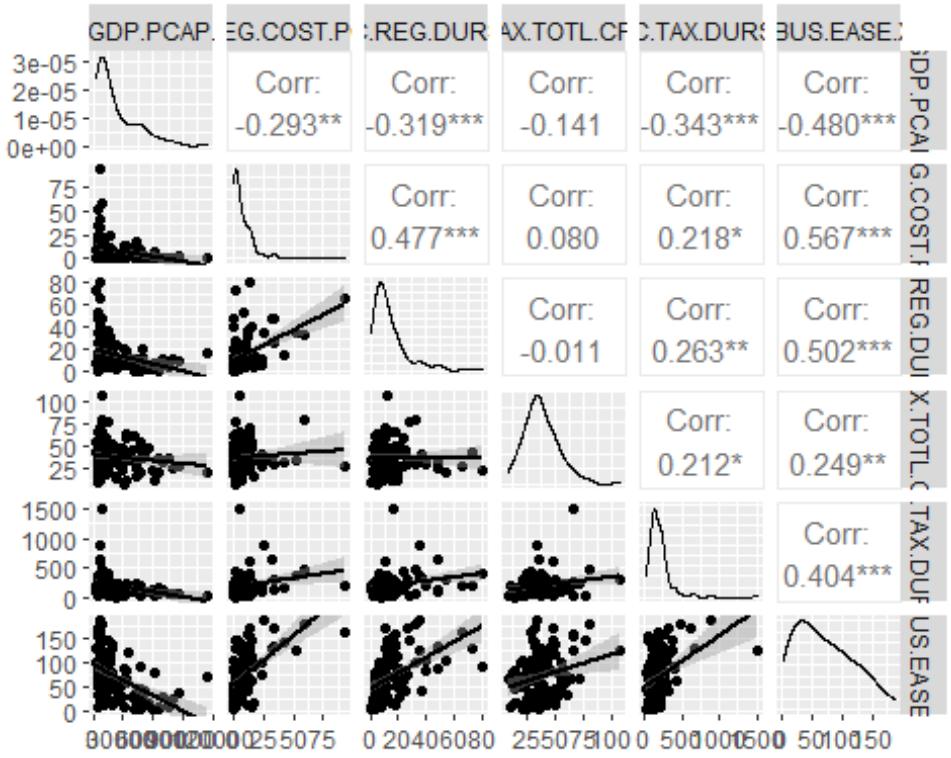
```
# описательные статистики
summary(DT.wdi.2019[, ..ind.names])
#>  NY.GDP.PCAP.CD   IC.REG.COST.PC.ZS  IC.REG.DURS    IC.TAX.TOTL.CP.ZS
#>  Min.   :  3115   Min.   : 0.00     Min.   : 0.50   Min.   :  8.00
#>  1st Qu.:  7336   1st Qu.: 1.05     1st Qu.: 5.50   1st Qu.: 27.70
#>  Median : 14908   Median : 4.20     Median :11.00   Median : 35.20
#>  Mean   : 22852   Mean   : 8.59     Mean   :15.18   Mean   : 37.45
#>  3rd Qu.: 32612   3rd Qu.:11.20     3rd Qu.:18.25   3rd Qu.: 46.85
#>  Max.   :114685   Max.   :93.50     Max.   :80.00   Max.   :106.30
#>   IC.TAX.DURS     IC.BUS.EASE.XQ
#>  Min.   :  22.5   Min.   :  1.00
#>  1st Qu.: 119.2   1st Qu.: 29.50
#>  Median : 170.0   Median : 61.00
#>  Mean   : 208.4   Mean   : 69.72
#>  3rd Qu.: 242.0   3rd Qu.:102.50
#>  Max.   :1501.0   Max.   :186.00
```

```
# коэффициенты вариации
CV <- round(sapply(DT.wdi.2019[, ..ind.names], sd) /
                sapply(DT.wdi.2019[, ..ind.names], mean) * 100, 1)
df.CV <- data.frame(Показатель = ind.labels[ind.names], Коэфф.вариации.прц = CV)
df.CV
#>                                                    Показатель
#> NY.GDP.PCAP.CD                      ВВП на душу, текущие цены, USD
#> IC.REG.COST.PC.ZS Затраты на создание бизнеса, % от ВНД на душу
#> IC.REG.DURS                         Время на создание бизнеса, дней
#> IC.TAX.TOTL.CP.ZS    Налоговая нагрузка на бизнес, % от прибыли
#> IC.TAX.DURS                         Время на выплату налогов, часов
#> IC.BUS.EASE.XQ                    Рейтинг лёгкости ведения бизнеса
#>                   Коэфф.вариации.прц
#> NY.GDP.PCAP.CD                  94.4
#> IC.REG.COST.PC.ZS              153.6
#> IC.REG.DURS                     97.6
#> IC.TAX.TOTL.CP.ZS               42.7
#> IC.TAX.DURS                     85.8
#> IC.BUS.EASE.XQ                  68.8
```

Для нас важно, что коэффициент вариации для зависимой переменной `IC.BUS.EASE.XQ` не выше 100%. Больше всего вариация у `IC.REG.COST.PC.ZS`. На этом этапе мы не будем предпринимать дополнительных преобразований данных.

Теперь построим графики разброса с рассчитаем коэффициенты корреляции, изобразив всё это на одном полотне функцией `ggpairs()`.

```
# строим графики взаимного разброса
ggpairs(DT.wdi.2019[, ..ind.names], lower = list(continuous = 'smooth'))
```



Мы получили матричный график, где коэффициенты корреляции находятся в правом верхнем треугольнике матрицы, по диагонали изображены графики функций плотности распределения, а в нижнем левом треугольнике изображены графики разброса с линиями регрессии.

Проанализируем графики разброса. Чем ближе точки к прямой регрессии, тем теснее взаимосвязь между парой переменных: в заголовке столбца, в котором расположен график, и в заголовке строки. Посмотрим на нижнюю строку графиков, где по вертикали отложена зависимая переменная `IC.BUS.EASE.XQ` (заголовок строки). Теснее всего выглядит линейная связь на втором графике, и наоборот, самое разреженное облако точек видно на четвёртом. Соответствующие коэффициенты корреляции нужно искать в последнем столбце матрицы (заголовок столбца `IC.BUS.EASE.XQ`). Прежде всего, обратим внимание, что после каждого коэффициента стоят три звёздочки. Это указывает на их высокую значимость. Мы возьмём стандартный уровень значимости 0.05, следовательно будем считать значимым те коэффициенты, после которых стоит одна (значимость 0.05), две (значимость 0.01) или три звёздочки (значимость свыше 0.01). Итак, исходя из коэффициентов корреляции в последнем столбце матрицы, все объясняющие переменные можно включать в таблицу.

Посмотрим на коэффициенты корреляции в других столбцах. Они отражают тесноту линейных связей между объясняющими переменными модели. Так, можно заметить, что корреляция между объясняющими `IC.REG.COST.PC.ZS` и `IC.REG.DURS` (0.477) значима и выше, чем значимая корреляция `IC.REG.DURS` с зависимой переменной `IC.BUS.EASE.XQ` (0.404), поэтому корректнее будет включить `IC.REG.COST.PC.ZS` и `IC.REG.DURS` по отдельности в две разные модели.

```
# модель 1
fit.1 <- lm(IC.BUS.EASE.XQ ~ . -IC.REG.COST.PC.ZS,
            data = DT.wdi.2019[, ..ind.names])
summary(fit.1)
#>
#> Call:
#> lm(formula = IC.BUS.EASE.XQ ~ . - IC.REG.COST.PC.ZS, data = DT.wdi.2019[,
#>     ..ind.names])
#>
#> Residuals:
#>     Min      1Q  Median      3Q     Max
#> -69.724 -27.190  -7.996  25.268  76.834
#>
#> Coefficients:
#>                     Estimate Std. Error t value Pr(>|t|)
#> (Intercept)       36.0943385 12.0710781   2.990 0.003467 **
#> NY.GDP.PCAP.CD    -0.0006165  0.0001800  -3.426 0.000873 ***
#> IC.REG.DURS        1.1960861  0.2548723   4.693 8.09e-06 ***
#> IC.TAX.TOTL.CP.ZS  0.5299304  0.2260815   2.344 0.020944 *
#> IC.TAX.DURS        0.0466098  0.0216127   2.157 0.033298 *
#> ---
#> Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
#>
#> Residual standard error: 36.82 on 106 degrees of freedom
#> Multiple R-squared:  0.4324, Adjusted R-squared:  0.411
#> F-statistic: 20.19 on 4 and 106 DF,  p-value: 2.194e-12
```

```
# модель 2
fit.2 <- lm(IC.BUS.EASE.XQ ~ . -IC.REG.DURS,
            data = DT.wdi.2019[, ..ind.names])
summary(fit.2)
#>
#> Call:
#> lm(formula = IC.BUS.EASE.XQ ~ . - IC.REG.DURS, data = DT.wdi.2019[,
#>     ..ind.names])
#>
#> Residuals:
#>     Min      1Q  Median      3Q     Max
#> -67.304 -24.112  -8.101  24.822  82.860
#>
#> Coefficients:
#>                     Estimate Std. Error t value Pr(>|t|)
#> (Intercept)       43.9828470 10.8109220   4.068 9.13e-05 ***
#> NY.GDP.PCAP.CD    -0.0005970  0.0001703  -3.505 0.000669 ***
#> IC.REG.COST.PC.ZS  1.5857892  0.2672319   5.934 3.77e-08 ***
#> IC.TAX.TOTL.CP.ZS  0.4073749  0.2143556   1.900 0.060089 .
#> IC.TAX.DURS        0.0503860  0.0203778   2.473 0.015004 *
#> ---
#> Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
#>
#> Residual standard error: 35.06 on 106 degrees of freedom
#> Multiple R-squared:  0.4855, Adjusted R-squared:  0.466
#> F-statistic:    25 on 4 and 106 DF,  p-value: 1.356e-14
```

Сравним модели по скорректированным R-квадратам (Adjusted R-squared): у второй он выше (0.466 > 0.411). Но у второй модели один из параметров незначим на уровне 0.05: p-значение для `IC.TAX.TOTL.CP.ZS` больше 0.05, и в конце соответствующей строки с коэффициентом стоит не звёздочка, а точка. Исключим эту переменную из модели.

```
# модель 2, параметры значимы на уровне 0.05
fit.3 <- lm(IC.BUS.EASE.XQ ~ . -IC.REG.DURS -IC.TAX.TOTL.CP.ZS,
            data = DT.wdi.2019[, ..ind.names])
summary(fit.3)
#>
#> Call:
#> lm(formula = IC.BUS.EASE.XQ ~ . - IC.REG.DURS - IC.TAX.TOTL.CP.ZS,
#>     data = DT.wdi.2019[, ..ind.names])
#>
#> Residuals:
#>     Min      1Q  Median      3Q     Max
#> -66.840 -24.196  -7.751  22.067  89.482
#>
#> Coefficients:
#>                     Estimate Std. Error t value Pr(>|t|)
#> (Intercept)       58.2674213  7.8647368   7.409 3.11e-11 ***
#> NY.GDP.PCAP.CD    -0.0006189  0.0001720  -3.599 0.000486 ***
#> IC.REG.COST.PC.ZS  1.5950943  0.2704282   5.898 4.35e-08 ***
#> IC.TAX.DURS        0.0570736  0.0203151   2.809 0.005902 **
```

```
#> ---
#> Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
#>
#> Residual standard error: 35.48 on 107 degrees of freedom
#> Multiple R-squared:  0.4679, Adjusted R-squared:  0.453
#> F-statistic: 31.37 on 3 and 107 DF,  p-value: 1.255e-14
```

Теперь все параметры значимы на уровне 0.05. Дадим интерпретацию, учитывая смысл переменных и их единицы измерения:

- коэффициент при `NY.GDP.PCAP.CD` равен -0.0006, следовательно, при увеличении валового внутреннего продукта на душу населения на 1 доллар позиция в рейтинге лёгкости ведения бизнеса уменьшится (т.к. коэффициент меньше нуля) на 0.0006. То есть с увеличением ВВП страна поднимается в рейтинге выше, и условия для бизнеса лучше. Это выглядит логично.

- коэффициент при `IC.REG.COST.PC.ZS` равен 1.5951, следовательно, при увеличении затрат на создание бизнеса на 1% от ВНД на душу населения позиция в рейтинге лёгкости ведения бизнеса увеличится почти на 1.6. Таким образом, страна упадёт более чем на полторы позиции в рейтинге. Эта объясняющая переменная работает как барьер для ведения бизнеса, и интерпретация, опять же, выглядит логичной.

- коэффициент при `IC.TAX.DURS` равен 0.057, следовательно, при увеличении время на создание бизнеса на 1 день позиция в рейтинге лёгкости ведения бизнеса увеличится на 0.057.

Мы интерпретировали построенную модель и можем сказать, что, согласно статистике Всемирного банка, среди стран с высоким и средне-высоким доходом значимыми барьерами для ведения бизнеса являются:

- затраты на создание бизнеса, выраженные в процентах от ВНД на душу населения;

- время на создание бизнеса, в днях;

- ВВП на душу населения.

Из них государство может напрямую влиять только на второй.

Индивидуальные задания на анализ данных

1. Посчитать описательные статистики, построить дескриптивные графики на данных по импорту в страну, загруженных из базы данных UN COMTRADE в индивидуальном задании к первой лабораторной. Найти топ-5 стран-партнёров в целом по миру, а также внутри групп стран по региону и уровню дохода.

2. Построить регрессионные модели для рейтинга простоты ведения бизнеса на данных по странам мира, загруженных с помощью пакета `WDI` в индивидуальном задании к первой лабораторной. Проанализировать взаимосвязи между показателями, оценить параметры моделей, проинтерпретировать параметры лучшей модели.

Дополнительная информация: работа с `data.table` и функциями пакета `dplyr` на примере данных по авиарейсам

Преобразование данных с помощью пакета `dplyr`

Пример №3 взят из книги “Язык R в задачах науки о данных” Х.Уикема и Г.Гроулмунда [14]. Применим функции манипуляции данными из пакета `dplyr` к данным по 336 776,00 авиарейсам из аэропортов Нью-Йорка в 2013 году из пакета `nycflights13`.

```
library('dplyr')            # функции манипуляции данными
library('nycflights13')     # данные по авиарейсам из Нью-Йорка
library('data.table')          # объекты "таблица данных"
```

Пакет `dplyr` переписывает некоторые базовые функции, о чём R сообщает в консоль. Так, чтобы обратиться к исходной версии функции `filter()`, нужно использовать полное имя с указанием пространства имён её пакета: `stats::filter()`. Нужные нам данные записаны в таблице `flights`, посмотрим на её содержимое.

```
flights
#> # A tibble: 336,776 x 19
#>     year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time
#>    <int> <int> <int>    <int>          <int>     <dbl>    <int>          <int>
#>  1  2013     1     1      517            515         2      830            819
#>  2  2013     1     1      533            529         4      850            830
#>  3  2013     1     1      542            540         2      923            850
#>  4  2013     1     1      544            545        -1     1004           1022
#>  5  2013     1     1      554            600        -6      812            837
#>  6  2013     1     1      554            558        -4      740            728
#>  7  2013     1     1      555            600        -5      913            854
#>  8  2013     1     1      557            600        -3      709            723
#>  9  2013     1     1      557            600        -3      838            846
#> 10  2013     1     1      558            600        -2      753            745
#> # ... with 336,766 more rows, and 11 more variables: arr_delay <dbl>,
#> #   carrier <chr>, flight <int>, tailnum <chr>, origin <chr>, dest <chr>,
#> #   air_time <dbl>, distance <dbl>, hour <dbl>, minute <dbl>, time_hour <dttm>
```

Таблица `flights` – это не фрейм данных, а объект класса *tibble*, или *тиббл-фрейм*. Это фреймы, оптимизированные для работы с пакетами *tidyverse*, и первое отличие, которое бросается в глаза – удобное представление таблицы в консоли (причём нам даже не понадобилась функция, чтобы его получить). В третьей строке указаны псевдонимы для типов данных. Всего их семь:

- `int` – целые числа;
- `dbl` – Числа с плавающей точкой (вещественные);

---

[14] Хэдли Уикем, Гарретт Гроулмунд, Язык R в задачах науки о данных. – М.: Диалектика, 2018.

- `chr` – символьные векторы (строки);
- `dttm` – дата + время;
- `lgl` – логические (`TRUE` и `FALSE`);
- `fctr` – факторы;
- `date` – даты.

Основные функции пакета `dplyr`, которые являются глаголами манипулирования данными:

- `filter()` – выбор наблюдений по их значениям, т.е. отбор строк таблицы;
- `select()` – выбор переменных по их именам, т.е. отбор столбцов таблицы;
- `arrange()` – перестановка строк;
- `mutate()` – создание новых переменных из существующих столбцов таблицы;
- `summarize()` – агрегирование таблицы;
- `group_by()` – функция для группировки строк таблицы по заданному критерию, может использоваться со всеми вышеперечисленными функциями.

Согласно философии `dplyr`, каждый глагол, преобразующий таблицу, подчиняется следующим принципам:

- Первый аргумент – фрейм данных.
- Следующие аргументы, в качестве которых используются имена переменных без кавычек, описывают действия, которые должны быть выполнены с фреймом.
- Результат – новый фрейм данных.

Обратите внимание: функции пакета `dplyr` никогда не изменяют входной набор данных (это один из принципов опрятной обработки), поэтому для сохранения результатов используйте присваивание `<-`.

Фильтруем строки с `filter()`

Отберём все авиарейсы за 1 января:

```
filter(flights, month == 1, day == 1)
#> # A tibble: 842 x 19
#>     year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time
#>    <int> <int> <int>    <int>          <int>     <dbl>    <int>          <int>
#>  1  2013     1     1      517            515         2      830            819
#>  2  2013     1     1      533            529         4      850            830
#>  3  2013     1     1      542            540         2      923            850
#>  4  2013     1     1      544            545        -1     1004           1022
#>  5  2013     1     1      554            600        -6      812            837
#>  6  2013     1     1      554            558        -4      740            728
#>  7  2013     1     1      555            600        -5      913            854
```

```
#>  8  2013     1     1      557            600        -3      709            723
#>  9  2013     1     1      557            600        -3      838            846
#> 10  2013     1     1      558            600        -2      753            745
#> # ... with 832 more rows, and 11 more variables: arr_delay <dbl>,
#> #   carrier <chr>, flight <int>, tailnum <chr>, origin <chr>, dest <chr>,
#> #   air_time <dbl>, distance <dbl>, hour <dbl>, minute <dbl>, time_hour <dttm>
```

Предыдущая функция только вывела результаты в консоль. Команда с присваиванием создаст новый объект, но обойдётся без вывода результата. Иногда требуется сделать и то, и другое одновременно, и для этого функцию надо заключить в круглые скобки:

```
(jan.1 <- filter(flights, month == 1, day == 1))
#> # A tibble: 842 x 19
#>     year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time
#>    <int> <int> <int>    <int>          <int>     <dbl>    <int>          <int>
#>  1  2013     1     1      517            515         2      830            819
#>  2  2013     1     1      533            529         4      850            830
#>  3  2013     1     1      542            540         2      923            850
#>  4  2013     1     1      544            545        -1     1004           1022
#>  5  2013     1     1      554            600        -6      812            837
#>  6  2013     1     1      554            558        -4      740            728
#>  7  2013     1     1      555            600        -5      913            854
#>  8  2013     1     1      557            600        -3      709            723
#>  9  2013     1     1      557            600        -3      838            846
#> 10  2013     1     1      558            600        -2      753            745
#> # ... with 832 more rows, and 11 more variables: arr_delay <dbl>,
#> #   carrier <chr>, flight <int>, tailnum <chr>, origin <chr>, dest <chr>,
#> #   air_time <dbl>, distance <dbl>, hour <dbl>, minute <dbl>, time_hour <dttm>
```

В фильтре часто применяются сравнения с заданным числом. Первый тонкий момент связан с округлением, которое не делает компьютер, оперируя вещественными числами. Сравните:

```
sqrt(2) ^ 2 == 2       # без округления равенство не выполняется
#> [1] FALSE
near(sqrt(2) ^ 2, 2)   # а вот функция near() делает приближённое сравнение
#> [1] TRUE
```

Второй тонкий момент связан с использованием логических операторов (все они перечислены на Рис.1).

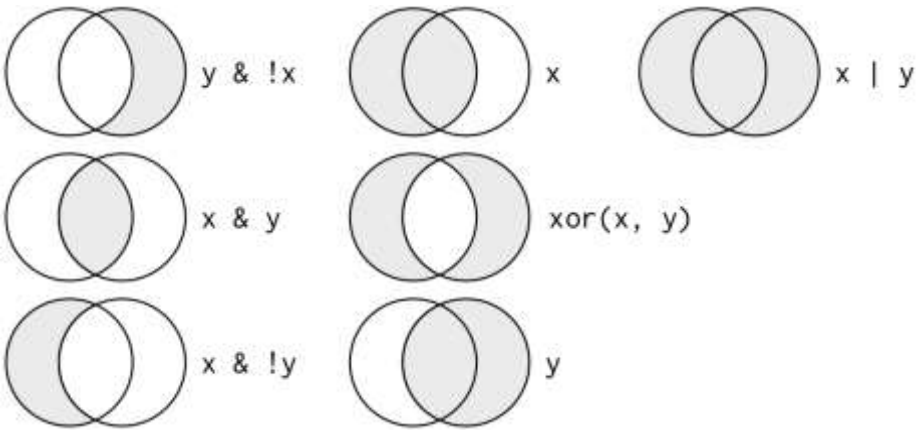


Рис.1. Логические операторы и соответствующие множества [15]

Сравните:

```
filter(flights, month == 11 | month == 12)   # рейсы в январе и декабре
#> # A tibble: 55,403 x 19
#>     year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time
#>    <int> <int> <int>    <int>          <int>     <dbl>    <int>          <int>
#>  1  2013    11     1        5           2359         6      352            345
#>  2  2013    11     1       35           2250       105      123           2356
#>  3  2013    11     1      455            500        -5      641            651
#>  4  2013    11     1      539            545        -6      856            827
#>  5  2013    11     1      542            545        -3      831            855
#>  6  2013    11     1      549            600       -11      912            923
#>  7  2013    11     1      550            600       -10      705            659
#>  8  2013    11     1      554            600        -6      659            701
#>  9  2013    11     1      554            600        -6      826            827
#> 10  2013    11     1      554            600        -6      749            751
#> # ... with 55,393 more rows, and 11 more variables: arr_delay <dbl>,
#> #   carrier <chr>, flight <int>, tailnum <chr>, origin <chr>, dest <chr>,
#> #   air_time <dbl>, distance <dbl>, hour <dbl>, minute <dbl>, time_hour <dttm>
filter(flights, month == 11 | 12)            # вся таблица
#> # A tibble: 336,776 x 19
#>     year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time
#>    <int> <int> <int>    <int>          <int>     <dbl>    <int>          <int>
#>  1  2013     1     1      517            515         2      830            819
#>  2  2013     1     1      533            529         4      850            830
#>  3  2013     1     1      542            540         2      923            850
#>  4  2013     1     1      544            545        -1     1004           1022
#>  5  2013     1     1      554            600        -6      812            837
#>  6  2013     1     1      554            558        -4      740            728
#>  7  2013     1     1      555            600        -5      913            854
#>  8  2013     1     1      557            600        -3      709            723
#>  9  2013     1     1      557            600        -3      838            846
#> 10  2013     1     1      558            600        -2      753            745
#> # ... with 336,766 more rows, and 11 more variables: arr_delay <dbl>,
#> #   carrier <chr>, flight <int>, tailnum <chr>, origin <chr>, dest <chr>,
#> #   air_time <dbl>, distance <dbl>, hour <dbl>, minute <dbl>, time_hour <dttm>
11 | 12                                      # вот почему
#> [1] TRUE
```

Вторая функция ничего не фильтрует, потому что операция `11 | 12` выполняется перед фильтрацией по значению, и с итоге мы ищем в столбце с номером месяца значение TRUE. Положительные числовые значения из столбца `month` неявно преобразуются в `TRUE`, и потому на выходе мы получаем всю таблицу.

Наш выбор – более универсальная и безопасная конструкция `x %in% y` – все значения вектора `x`, которые содержатся в векторе `y`:

---

[15] Репозиторий к книге “Язык R в задачах науки о данных”. URL: https://github.com/hadley/r4ds

```
filter(flights, month %in% c(11, 12))   # рейсы в январе и декабре
#> # A tibble: 55,403 x 19
#>     year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time
#>    <int> <int> <int>    <int>          <int>     <dbl>    <int>          <int>
#>  1  2013    11     1        5           2359         6      352            345
#>  2  2013    11     1       35           2250       105      123           2356
#>  3  2013    11     1      455            500        -5      641            651
#>  4  2013    11     1      539            545        -6      856            827
#>  5  2013    11     1      542            545        -3      831            855
#>  6  2013    11     1      549            600       -11      912            923
#>  7  2013    11     1      550            600       -10      705            659
#>  8  2013    11     1      554            600        -6      659            701
#>  9  2013    11     1      554            600        -6      826            827
#> 10  2013    11     1      554            600        -6      749            751
#> # ... with 55,393 more rows, and 11 more variables: arr_delay <dbl>,
#> #   carrier <chr>, flight <int>, tailnum <chr>, origin <chr>, dest <chr>,
#> #   air_time <dbl>, distance <dbl>, hour <dbl>, minute <dbl>, time_hour <dttm>
```

Если условия отбора определяются по нескольким столбцам, мы перечисляем их через запятую. Отберём все рейсы с задержкой и по прибытию, и по отправке не более двух часов.

```
filter(flights, arr_delay <= 120, dep_delay <= 120)
#> # A tibble: 316,050 x 19
#>     year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time
#>    <int> <int> <int>    <int>          <int>     <dbl>    <int>          <int>
#>  1  2013     1     1      517            515         2      830            819
#>  2  2013     1     1      533            529         4      850            830
#>  3  2013     1     1      542            540         2      923            850
#>  4  2013     1     1      544            545        -1     1004           1022
#>  5  2013     1     1      554            600        -6      812            837
#>  6  2013     1     1      554            558        -4      740            728
#>  7  2013     1     1      555            600        -5      913            854
#>  8  2013     1     1      557            600        -3      709            723
#>  9  2013     1     1      557            600        -3      838            846
#> 10  2013     1     1      558            600        -2      753            745
#> # ... with 316,040 more rows, and 11 more variables: arr_delay <dbl>,
#> #   carrier <chr>, flight <int>, tailnum <chr>, origin <chr>, dest <chr>,
#> #   air_time <dbl>, distance <dbl>, hour <dbl>, minute <dbl>, time_hour <dttm>
```

Ещё один момент связан с обработкой пропущенных значений (`NA`). Функция `filter()` включает в результат только те строки, у которых условие равно `TRUE`, т.е. исключаются и те строки, для которых условие не выполняется, и отсутствующие значения.

Переставляем строки с `arrange()`

Функция `arrange()` используется для сортировке таблиц. Аргументы – таблица и имена столбцов, по которым её надо отсортировать (по первому из них сортируем в первую очередь). Рейсы в порядке возрастания даты:

```
arrange(flights, year, month, day)     # по возрастанию даты
#> # A tibble: 336,776 x 19
#>     year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time
```

```
#>    <int> <int> <int>    <int>          <int>     <dbl>    <int>          <int>
#>  1  2013     1     1      517            515         2      830            819
#>  2  2013     1     1      533            529         4      850            830
#>  3  2013     1     1      542            540         2      923            850
#>  4  2013     1     1      544            545        -1     1004           1022
#>  5  2013     1     1      554            600        -6      812            837
#>  6  2013     1     1      554            558        -4      740            728
#>  7  2013     1     1      555            600        -5      913            854
#>  8  2013     1     1      557            600        -3      709            723
#>  9  2013     1     1      557            600        -3      838            846
#> 10  2013     1     1      558            600        -2      753            745
#> # ... with 336,766 more rows, and 11 more variables: arr_delay <dbl>,
#> #   carrier <chr>, flight <int>, tailnum <chr>, origin <chr>, dest <chr>,
#> #   air_time <dbl>, distance <dbl>, hour <dbl>, minute <dbl>, time_hour <dttm>
```

Рейсы в порядке убывания (функция `desc()`) задержки прибытия:

```
arrange(flights, desc(arr_delay))      # по убыванию задержки
#> # A tibble: 336,776 x 19
#>    year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time
#>   <int> <int> <int>    <int>          <int>     <dbl>    <int>          <int>
#>  1  2013     1     9      641            900      1301     1242           1530
#>  2  2013     6    15     1432           1935      1137     1607           2120
#>  3  2013     1    10     1121           1635      1126     1239           1810
#>  4  2013     9    20     1139           1845      1014     1457           2210
#>  5  2013     7    22      845           1600      1005     1044           1815
#>  6  2013     4    10     1100           1900       960     1342           2211
#>  7  2013     3    17     2321            810       911      135           1020
#>  8  2013     7    22     2257            759       898      121           1026
#>  9  2013    12     5      756           1700       896     1058           2020
#> 10  2013     5     3     1133           2055       878     1250           2215
#> # ... with 336,766 more rows, and 11 more variables: arr_delay <dbl>,
#> #   carrier <chr>, flight <int>, tailnum <chr>, origin <chr>, dest <chr>,
#> #   air_time <dbl>, distance <dbl>, hour <dbl>, minute <dbl>, time_hour <dttm>
```

Отсутствующие значения всегда будут в конце отсортированной таблицы:

```
df <- tibble(x = c(5, 2, NA))
arrange(df, x)
#> # A tibble: 3 x 1
#>       x
#>   <dbl>
#> 1     2
#> 2     5
#> 3    NA
```

Отбираем столбцы с `select()`

Функция `select()` полезна тем, что её синтаксис гораздо короче, чем базовые конструкции с оператором квадратных скобок для фреймов. Кроме того, есть приятные бонусы, например, с именами столбцов работают операторы “минус” и “двоеточие”.

```
# выбрать столбцы по имени year, month, day
select(flights, year, month, day)
```

```
#> # A tibble: 336,776 x 3
#>     year month   day
#>    <int> <int> <int>
#>  1  2013     1     1
#>  2  2013     1     1
#>  3  2013     1     1
#>  4  2013     1     1
#>  5  2013     1     1
#>  6  2013     1     1
#>  7  2013     1     1
#>  8  2013     1     1
#>  9  2013     1     1
#> 10  2013     1     1
#> # ... with 336,766 more rows
# выбрать столбцы между year и day (включая их)
select(flights, year:day)
#> # A tibble: 336,776 x 3
#>     year month   day
#>    <int> <int> <int>
#>  1  2013     1     1
#>  2  2013     1     1
#>  3  2013     1     1
#>  4  2013     1     1
#>  5  2013     1     1
#>  6  2013     1     1
#>  7  2013     1     1
#>  8  2013     1     1
#>  9  2013     1     1
#> 10  2013     1     1
#> # ... with 336,766 more rows
# выбрать столбцы кроме year и day (и кроме них)
select(flights, -(year:day))
#> # A tibble: 336,776 x 16
#>    dep_time sched_dep_time dep_delay arr_time sched_arr_time arr_delay carrier
#>       <int>          <int>     <dbl>    <int>          <int>     <dbl> <chr>
#>  1      517            515         2      830            819        11 UA
#>  2      533            529         4      850            830        20 UA
#>  3      542            540         2      923            850        33 AA
#>  4      544            545        -1     1004           1022       -18 B6
#>  5      554            600        -6      812            837       -25 DL
#>  6      554            558        -4      740            728        12 UA
#>  7      555            600        -5      913            854        19 B6
#>  8      557            600        -3      709            723       -14 EV
#>  9      557            600        -3      838            846        -8 B6
#> 10      558            600        -2      753            745         8 AA
#> # ... with 336,766 more rows, and 9 more variables: flight <int>,
#> #   tailnum <chr>, origin <chr>, dest <chr>, air_time <dbl>, distance <dbl>,
#> #   hour <dbl>, minute <dbl>, time_hour <dttm>
```

Но ещё полезнее функции гибкого поиска по именам столбцов:

- `starst_with('abc')` – имена, которые начинаются на `'abc'`.

- `ends_with('abc')` – имена, которые заканчиваются на `'abc'`.
- `contains('ijk')` – имена, которые содержат `'ijk'`.
- `matches('(.)\\1'` – имена, соответствующие регулярному выражению.
- `num_range('x', 1:3)` – имена x1, x2 и x3.

```
# столбцы, имена которых заканчиваются на 'delay' (задержка)
select(flights, ends_with('_delay'))
#> # A tibble: 336,776 x 2
#>    dep_delay arr_delay
#>        <dbl>     <dbl>
#>  1         2        11
#>  2         4        20
#>  3         2        33
#>  4        -1       -18
#>  5        -6       -25
#>  6        -4        12
#>  7        -5        19
#>  8        -3       -14
#>  9        -3        -8
#> 10        -2         8
#> # ... with 336,766 more rows
```

Иногда нужно всего лишь переставить в начало таблицы несколько столбцов, и перечислять имена остальных накладно. Здесь поможет вспомогательная функция `everything()`, которая означает “все остальные столбцы”:

```
# переставить время рейса и время полёта в начало таблицы
select(flights, time_hour, air_time, everything())
#> # A tibble: 336,776 x 19
#>    time_hour           air_time  year month   day dep_time sched_dep_time
#>    <dttm>                 <dbl> <int> <int> <int>    <int>          <int>
#>  1 2013-01-01 05:00:00      227  2013     1     1      517            515
#>  2 2013-01-01 05:00:00      227  2013     1     1      533            529
#>  3 2013-01-01 05:00:00      160  2013     1     1      542            540
#>  4 2013-01-01 05:00:00      183  2013     1     1      544            545
#>  5 2013-01-01 06:00:00      116  2013     1     1      554            600
#>  6 2013-01-01 05:00:00      150  2013     1     1      554            558
#>  7 2013-01-01 06:00:00      158  2013     1     1      555            600
#>  8 2013-01-01 06:00:00       53  2013     1     1      557            600
#>  9 2013-01-01 06:00:00      140  2013     1     1      557            600
#> 10 2013-01-01 06:00:00      138  2013     1     1      558            600
#> # ... with 336,766 more rows, and 12 more variables: dep_delay <dbl>,
#> #   arr_time <int>, sched_arr_time <int>, arr_delay <dbl>, carrier <chr>,
#> #   flight <int>, tailnum <chr>, origin <chr>, dest <chr>, distance <dbl>,
#> #   hour <dbl>, minute <dbl>
```

Отметим, что функция `select()` не предназначена для переименовывания столбцов, поскольку все столбцы, не упомянутые в аргументах функции, выпадают. Для этого служит `rename()`:

```
# переименовать столбец с использованием змеиного регистра
(flights.mod <- rename(flights, tail_num = tailnum))
#> # A tibble: 336,776 x 19
#>     year month   day dep_time sched_dep_time dep_delay arr_time sched_arr_time
#>    <int> <int> <int>    <int>          <int>     <dbl>    <int>          <int>
#>  1  2013     1     1      517            515         2      830            819
#>  2  2013     1     1      533            529         4      850            830
#>  3  2013     1     1      542            540         2      923            850
#>  4  2013     1     1      544            545        -1     1004           1022
#>  5  2013     1     1      554            600        -6      812            837
#>  6  2013     1     1      554            558        -4      740            728
#>  7  2013     1     1      555            600        -5      913            854
#>  8  2013     1     1      557            600        -3      709            723
#>  9  2013     1     1      557            600        -3      838            846
#> 10  2013     1     1      558            600        -2      753            745
#> # ... with 336,766 more rows, and 11 more variables: arr_delay <dbl>,
#> #   carrier <chr>, flight <int>, tail_num <chr>, origin <chr>, dest <chr>,
#> #   air_time <dbl>, distance <dbl>, hour <dbl>, minute <dbl>, time_hour <dttm>
# убедимся, что столбец переименован
select(flights.mod, tail_num, everything())
#> # A tibble: 336,776 x 19
#>    tail_num  year month   day dep_time sched_dep_time dep_delay arr_time
#>    <chr>    <int> <int> <int>    <int>          <int>     <dbl>    <int>
#>  1 N14228    2013     1     1      517            515         2      830
#>  2 N24211    2013     1     1      533            529         4      850
#>  3 N619AA    2013     1     1      542            540         2      923
#>  4 N804JB    2013     1     1      544            545        -1     1004
#>  5 N668DN    2013     1     1      554            600        -6      812
#>  6 N39463    2013     1     1      554            558        -4      740
#>  7 N516JB    2013     1     1      555            600        -5      913
#>  8 N829AS    2013     1     1      557            600        -3      709
#>  9 N593JB    2013     1     1      557            600        -3      838
#> 10 N3ALAA    2013     1     1      558            600        -2      753
#> # ... with 336,766 more rows, and 11 more variables: sched_arr_time <int>,
#> #   arr_delay <dbl>, carrier <chr>, flight <int>, origin <chr>, dest <chr>,
#> #   air_time <dbl>, distance <dbl>, hour <dbl>, minute <dbl>, time_hour <dttm>
```

Добавляем новые столбцы с `mutate()`

Функция `mutate()` добавляет новые переменные в конец набора данных, поэтому для наглядности сформируем таблицу поменьше:

```
(flights_sml <- select(flights, year:day, ends_with('delay'),
                       distance, air_time))
#> # A tibble: 336,776 x 7
#>     year month   day dep_delay arr_delay distance air_time
#>    <int> <int> <int>     <dbl>     <dbl>    <dbl>    <dbl>
#>  1  2013     1     1         2        11     1400      227
#>  2  2013     1     1         4        20     1416      227
#>  3  2013     1     1         2        33     1089      160
#>  4  2013     1     1        -1       -18     1576      183
#>  5  2013     1     1        -6       -25      762      116
#>  6  2013     1     1        -4        12      719      150
```

```
#>  7  2013     1     1        -5        19     1065      158
#>  8  2013     1     1        -3       -14      229       53
#>  9  2013     1     1        -3        -8      944      140
#> 10  2013     1     1        -2         8      733      138
#> # ... with 336,766 more rows
```

Посчитаем время, которое удалось нагнать, т.е. разницу между задержкой прибытия и задержкой вылета (`gain`), и седнюю скорость полёта:

```
# нагнанное время и скорость
mutate(flights_sml,
       gain = arr_delay - dep_delay,
       speed = distance / air_time * 60)
#> # A tibble: 336,776 x 9
#>     year month   day dep_delay arr_delay distance air_time  gain speed
#>    <int> <int> <int>     <dbl>     <dbl>    <dbl>    <dbl> <dbl> <dbl>
#>  1  2013     1     1         2        11     1400      227     9  370.
#>  2  2013     1     1         4        20     1416      227    16  374.
#>  3  2013     1     1         2        33     1089      160    31  408.
#>  4  2013     1     1        -1       -18     1576      183   -17  517.
#>  5  2013     1     1        -6       -25      762      116   -19  394.
#>  6  2013     1     1        -4        12      719      150    16  288.
#>  7  2013     1     1        -5        19     1065      158    24  404.
#>  8  2013     1     1        -3       -14      229       53   -11  259.
#>  9  2013     1     1        -3        -8      944      140    -5  405.
#> 10  2013     1     1        -2         8      733      138    10  319.
#> # ... with 336,766 more rows
```

Рассчитаем, сколько времени рейсы в среднем нагоняли за час, воспользовавшись тем, что на только что созданные столбцы можно ссылаться:

```
mutate(flights_sml,
       gain = arr_delay - dep_delay,
       hours = air_time / 60,
       gain_per_hour = gain / hours)
#> # A tibble: 336,776 x 10
#>     year month   day dep_delay arr_delay distance air_time  gain hours
#>    <int> <int> <int>     <dbl>     <dbl>    <dbl>    <dbl> <dbl> <dbl>
#>  1  2013     1     1         2        11     1400      227     9 3.78
#>  2  2013     1     1         4        20     1416      227    16 3.78
#>  3  2013     1     1         2        33     1089      160    31 2.67
#>  4  2013     1     1        -1       -18     1576      183   -17 3.05
#>  5  2013     1     1        -6       -25      762      116   -19 1.93
#>  6  2013     1     1        -4        12      719      150    16 2.5
#>  7  2013     1     1        -5        19     1065      158    24 2.63
#>  8  2013     1     1        -3       -14      229       53   -11 0.883
#>  9  2013     1     1        -3        -8      944      140    -5 2.33
#> 10  2013     1     1        -2         8      733      138    10 2.3
#> # ... with 336,766 more rows, and 1 more variable: gain_per_hour <dbl>
```

Если требуется сохранить только пересчитанные столбцы, используйте `transmute()`:

```
transmute(flights_sml,
          gain = arr_delay - dep_delay,
```

```
          hours = air_time / 60,
          gain_per_hour = gain / hours)
#> # A tibble: 336,776 x 3
#>     gain hours gain_per_hour
#>    <dbl> <dbl>         <dbl>
#>  1     9 3.78           2.38
#>  2    16 3.78           4.23
#>  3    31 2.67          11.6
#>  4   -17 3.05          -5.57
#>  5   -19 1.93          -9.83
#>  6    16 2.5            6.4
#>  7    24 2.63           9.11
#>  8   -11 0.883        -12.5
#>  9    -5 2.33          -2.14
#> 10    10 2.3            4.35
#> # ... with 336,766 more rows
```

При вычислении нового столбца можно также использовать функции агрегирования (`sum()`, `mean()`), логарифмирования (`log()`, `log2()`, `log10()`), модульной арифметики (операторы `%/%` и `%%`), смещения (`lag()`, `lead()`), ранжирования (`min_rank()`, `row_number()`, `cume_dist()`) – и это далеко неполный список. Всё многообразие трансформаций охватить одним примером нельзя, поэтому просто скажем, что `mutate()` позволяет производить со столбцами преобразования любой сложности.

Агрегируем таблицу с `summarize()`

Глагол `summarize()` сворачивает таблицу в одну строку (или в одну строку на каждую подвыборку по функции `group_by()`). Среднее время задержки вылета (пропуски выбрасываем):

```
summarize(flights, delay = mean(dep_delay, na.rm = T))
#> # A tibble: 1 x 1
#>   delay
#>   <dbl>
#> 1  12.6
```

То же среднее время, но по датам:

```
by_day <- group_by(flights, year, month, day)
summarize(by_day, delay = mean(dep_delay, na.rm = T))
#> `summarise()` has grouped output by 'year', 'month'. You can override using the
`.groups` argument.
#> # A tibble: 365 x 4
#> # Groups:   year, month [12]
#>     year month   day delay
#>    <int> <int> <int> <dbl>
#>  1  2013     1     1 11.5
#>  2  2013     1     2 13.9
#>  3  2013     1     3 11.0
#>  4  2013     1     4  8.95
#>  5  2013     1     5  5.73
#>  6  2013     1     6  7.15
#>  7  2013     1     7  5.42
```

```
#>  8  2013     1     8  2.55
#>  9  2013     1     9  2.28
#> 10  2013     1    10  2.84
#> # ... with 355 more rows
```

Создадим таблицу для сопоставления пункта назначения (нас интересуют континентальные рейсы, поэтому мы не будем учитывать Гонолулу) и среднего времени задержки рейса.

```
# группируем рейсы по пунктам назначения
by_dest <- group_by(flights, dest)
# по группам считаем число рейсов, средние расстояние и задержку прибытия
delay <- summarize(by_dest,
                   count = n(),
                   dist = mean(distance, na.rm = T),
                   delay = mean(arr_delay, na.rm = T))
# фильтруем строки: пункты с числом опозданий больше 20, кроме Гонолулу
(delay <- filter(delay, count > 20, dest != 'HNL'))
#> # A tibble: 96 x 4
#>    dest  count  dist delay
#>    <chr> <int> <dbl> <dbl>
#>  1 ABQ     254 1826   4.38
#>  2 ACK     265  199   4.85
#>  3 ALB     439  143  14.4
#>  4 ATL   17215  757. 11.3
#>  5 AUS    2439 1514.  6.02
#>  6 AVL     275  584.  8.00
#>  7 BDL     443  116   7.05
#>  8 BGR     375  378   8.03
#>  9 BHM     297  866. 16.9
#> 10 BNA    6333  758. 11.8
#> # ... with 86 more rows
```

А теперь сделаем то же самое с использованием **каналов** %>%. Аналог оператора %>% в используемой нами грамматике обработки данных – слово “затем”. То, что стоит слева от канала %>%, передаётся функции, которая стоит справа. Функции можно сцеплять в один канал, как вагоны поезда.

```
(delays <- flights %>%
     group_by(dest) %>%
     summarize(count = n(),
               dist = mean(distance, na.rm = T),
               delay = mean(arr_delay, na.rm = T)) %>%
     filter(count > 20, dest != 'HNL'))
#> # A tibble: 96 x 4
#>    dest  count  dist delay
#>    <chr> <int> <dbl> <dbl>
#>  1 ABQ     254 1826   4.38
#>  2 ACK     265  199   4.85
#>  3 ALB     439  143  14.4
#>  4 ATL   17215  757. 11.3
#>  5 AUS    2439 1514.  6.02
#>  6 AVL     275  584.  8.00
```

```
#>  7 BDL     443  116   7.05
#>  8 BGR     375  378   8.03
#>  9 BHM     297  866. 16.9
#> 10 BNA    6333  758. 11.8
#> # ... with 86 more rows
```

Обратите внимание: у функций-глаголов в канале отсутствует первый аргумент. Его роль для `group_by()` играет таблица `flights`, которая стоит слева от `%>%`, для `summarize()` – таблица-результат функции `group_by()`, и так далее. В итоге каналы делают код более коротким и связным.

В функции `summarize()` можно использовать в принципе любые агрегирующие операции. Особенно полезны следующие функции:

- меры среднего: `mean(x)`, `median(x)`;
- меры разброса: `sd(x)`, `IQR(x)`, `mad(x)`;
- меры ранжирования: `min(x)`, `quantile(x, .25)`, `max(x)`;
- порядковые меры: `first(x)`, `nth(x)`, `last(x)`;
- счётчики: `n(x)`, `sum(!is.na(x))` (количество пропусков), `ndistinct()`;
- количество и доли логических значений: `sum(x>0)`, `sum(x>0)/length(x)`.

Объекты для хранения больших таблиц: `data.table`

В 2015 году был предложен новый тип объектов для хранения таблиц – data.table, или таблица данных [16]. Этот объект наследует методы data.frame и добавляет к ним свои, направленные на более эффективное выполнение операций выборки, обновления, группировки данных. Кроме того, методы data.table написаны на C, поэтому работают гораздо быстрее, чем аналогичные у data.frame. Разница становится заметной при обработке массивов данных порядка миллионов строк. Есть бенчмарки, которые показывают, что операции группировки `data.table` выполняются быстрее, чем тиббл-таблиц средствами пакета `dplyr`, а также быстрее, чем средствами библиотеки `pandas` для Python [17] [18].

Объекты типа `data.table` тоже имеют формат отображения в консоли, отличный от обычных фреймов данных. Он не такой подробный, как у тиббл-таблиц, но лаконичен и по-своему удобен, например, показывает первые и последние строки.

```
# объект типа data.table
DT.flights <- data.table(flights)
```

---

[16] M Dowle, A Srinivasan, T Short, S Lianoglou with contributions from R Saporta and E Antonyan (2015). data.table: Extension of Data.frame. R package version 1.9.6. https://CRAN.R-project.org/package=data.table

[17] https://github.com/Rdatatable/data.table/wiki/Benchmarks-:-Grouping

[18] https://rpubs.com/edwardcooper/data_table_benchmark

```
DT.flights
#>         year month day dep_time sched_dep_time dep_delay arr_time
#>      1: 2013     1   1      517            515         2      830
#>      2: 2013     1   1      533            529         4      850
#>      3: 2013     1   1      542            540         2      923
#>      4: 2013     1   1      544            545        -1     1004
#>      5: 2013     1   1      554            600        -6      812
#>     ---
#> 336772: 2013     9  30       NA           1455        NA       NA
#> 336773: 2013     9  30       NA           2200        NA       NA
#> 336774: 2013     9  30       NA           1210        NA       NA
#> 336775: 2013     9  30       NA           1159        NA       NA
#> 336776: 2013     9  30       NA            840        NA       NA
#>         sched_arr_time arr_delay carrier flight tailnum origin dest air_time
#>      1:            819        11      UA   1545  N14228    EWR  IAH      227
#>      2:            830        20      UA   1714  N24211    LGA  IAH      227
#>      3:            850        33      AA   1141  N619AA    JFK  MIA      160
#>      4:           1022       -18      B6    725  N804JB    JFK  BQN      183
#>      5:            837       -25      DL    461  N668DN    LGA  ATL      116
#>     ---
#> 336772:           1634        NA      9E   3393    <NA>    JFK  DCA       NA
#> 336773:           2312        NA      9E   3525    <NA>    LGA  SYR       NA
#> 336774:           1330        NA      MQ   3461  N535MQ    LGA  BNA       NA
#> 336775:           1344        NA      MQ   3572  N511MQ    LGA  CLE       NA
#> 336776:           1020        NA      MQ   3531  N839MQ    LGA  RDU       NA
#>         distance hour minute            time_hour
#>      1:     1400    5     15 2013-01-01 05:00:00
#>      2:     1416    5     29 2013-01-01 05:00:00
#>      3:     1089    5     40 2013-01-01 05:00:00
#>      4:     1576    5     45 2013-01-01 05:00:00
#>      5:      762    6      0 2013-01-01 06:00:00
#>     ---
#> 336772:      213   14     55 2013-09-30 14:00:00
#> 336773:      198   22      0 2013-09-30 22:00:00
#> 336774:      764   12     10 2013-09-30 12:00:00
#> 336775:      419   11     59 2013-09-30 11:00:00
#> 336776:      431    8     40 2013-09-30 08:00:00
```

Поскольку `data.table` обычно используется для хранения больших таблиц, полезно просматривать список таблиц этого типа в оперативной памяти:

```
# вывести список таблиц в памяти
tables()
#>           NAME    NROW NCOL MB
#> 1:  DT.country     304    4  0
#> 2:  DT.flights 336,776   19 39
#> 3:   DT.import      18   10  0
#> 4: DT.wdi.2019     111   10  0
#>
COLS
#> 1:
Reporter.iso2c,Reporter.country,Reporter.region,Reporter.income
#> 2:
```

```
year,month,day,dep_time,sched_dep_time,dep_delay,...
#> 3:
Reporter,Year,Period,Trade.Flow,Partner,Commodity.Code,...
#> 4:
iso2c,country,NY.GDP.PCAP.CD,IC.REG.COST.PC.ZS,IC.REG.DURS,IC.TAX.TOTL.CP.ZS,...
#>         KEY
#> 1:
#> 2:
#> 3: Reporter
#> 4:
#> Total: 39MB
```

Обратим внимание на столбец “KEY”: в объекты data.table можно добавлять ключевые столбцы и делать с их помощью операции пересечения и объединения по принципу SQL-запросов. Кроме того, ключи ускоряют обработку таких таблиц.

Выборки строк из таблицы можно делать так же, как из фрейма: указывая их номера перед запятой в квадратных скобках после имени таблицы. Также можно применять условия на значения, т.е. использовать в квадратных скобках не номера строк, а логические векторы. Однако выбор столбцов уже не работает так, как с объектом `data.frame`. При работе с таблицами надо придерживаться следующего синтаксиса:

`DF[<`*условие_на_строки*`>, <`*список_столбцов*`>, <`*условие_группировки*`>]`

Условие на строки – это логическое выражение. В таблицу войдёт те строки, для которых значение этого выражения равно `TRUE`.

Условие группировки – аргумент `by`, которому мы присваиваем название столбца без кавычек, либо список из названий нескольких. Аналогично функции `group_by()` из пакета `dplyr`, аргумент `by` делает подвыборки таблицы данных по всем уникальным значениям столбцов из условия группировки, поэтому нужно следить, чтобы эти столбцы содержали дискретные данные.

В *списке столбцов* могут стоять конструкции, которые создают новые переменные, в том числе и довольно сложные, на базе функций `apply()`. Кроме того, в `data.table` применяются специальные выражения, которые начинаются с символа точки. Остановимся на них подробнее.

- `.N` – числовой вектор длины 1, количество строк в группе;
- `.SD` – это объект data.table, который содержит подвыборку исходной таблицы по каждой из групп, исключая группы по переменным, указанным в аргументе “by”;
- `.BY` – список единичных векторов, по одному на каждую группу в аргументе “by” (удобно, если группы заранее неизвестны);
- `.I` – числовой вектор, который хранит номера строк в исходной таблице для каждого элемента в группе;

- `.GRP` – числовой вектор длины 1, простой счётчик групп: 1 для первой группы, 2 для второй, и т.д [19].

Повторим сложные трансформации таблицы рейсов, которые сделали выше средствами `dplyr`.

```
# делаем то же, что сделали средствами dplyr:
# группировка + агрегирование + фильтрация
delay.2 <- DT.flights[, list(count = .N,
                             dist = mean(distance, na.rm = T),
                             delay = mean(arr_delay, na.rm = T)),
                      by = dest]
delay.2
#>      dest count      dist       delay
#>   1:  IAH  7198 1407.2067   4.2407904
#>   2:  MIA 11728 1091.5524   0.2990598
#>   3:  BQN   896 1578.9833   8.2454955
#>   4:  ATL 17215  757.1082  11.3001128
#>   5:  ORD 17283  729.0008   5.8766148
#>  ---
#> 101:  LEX     1  604.0000 -22.0000000
#> 102:  CHO    52  305.0000   9.5000000
#> 103:  TVC   101  652.3861  12.9684211
#> 104:  ANC     8 3370.0000  -2.5000000
#> 105:  LGA     1   17.0000         NaN
```

Отметим, что в отличие от функций `dplyr`, этот способ выдаёт результирующую таблицу, не отсортированную по первому столбцу. Повторим пример с расчётом времени задержки, которое рейсы нагоняют за час полёта, учитывая, что новые столбцы нельзя использовать сразу:

```
# в data.frame нельзя сразу обращаться к новым столбцам
DT.flights[, list(gain = arr_delay - dep_delay,
                  hours = air_time / 60,
                  gain_per_hour = (arr_delay - dep_delay) / (air_time / 60))]
#>         gain    hours gain_per_hour
#>      1:    9 3.783333      2.378855
#>      2:   16 3.783333      4.229075
#>      3:   31 2.666667     11.625000
#>      4:  -17 3.050000     -5.573770
#>      5:  -19 1.933333     -9.827586
#>     ---
#> 336772:   NA       NA           NA
#> 336773:   NA       NA           NA
#> 336774:   NA       NA           NA
#> 336775:   NA       NA           NA
#> 336776:   NA       NA           NA
```

---

[19] M Dowle, A Srinivasan, T Short, S Lianoglou with contributions from R Saporta, E Antonyan. Package 'data.table' Reference Manual, September 19, 2015. URL: https://cran.r-project.org/web/packages/data.table/data.table.pdf

В данном случае результат аналогичен тому, что возвращает нам функция `transmute()`, т.е. мы видим только новые столбцы.

Составим ещё одну конструкцию с использованием синтаксиса `data.table`. Пусть нас интересует среднее время задержки рейсов в пункт назначения `'DSM'` по месяцам. Насколько отличаются задержки по месяцам?

```
# отбор только наблюдений из группы, их усреднение и сортировка по убыванию
DT.flights.sml <- DT.flights[, list(month, arr_delay, dest)]
DT.flights.sml[dest == 'DSM',
               list(count = .N,
                    mean_arr_delay_DSM = mean(arr_delay, na.rm = T)),
               by = month][, .SD[order(-mean_arr_delay_DSM)]]
#>     month count mean_arr_delay_DSM
#>  1:     3    26          60.576923
#>  2:     1    27          53.583333
#>  3:     2    24          48.181818
#>  4:     4    26          36.791667
#>  5:    12    40          34.305556
#>  6:     6    49          33.351351
#>  7:     7    76          16.424242
#>  8:    11    55          15.685185
#>  9:     5    49           5.021277
#> 10:    10    85           3.560976
#> 11:     8    56           2.000000
#> 12:     9    56           1.962264
```

У нас получилась довольно нагруженная конструкция с двойным оператором квадратных скобок: к таблице применяется первый, а к результату – следующий. Тем не менее, такой синтаксис гораздо лаконичнее стандартного для фреймов данных.