Продвинутая обработка данных в dplyr

Юрченков Иван Александрович

24 09 2020

Оглавление

[Обработка данных в dplyr 1](#_Toc51967888)

[Почему dplyr 1](#_Toc51967889)

[Загрузка и подключение dplyr к проекту 2](#_Toc51967890)

[Концепция конвееризации функций 2](#_Toc51967891)

[Фильтрация данных в dplyr 5](#_Toc51967892)

[Выбор параметров и структуры данных в dplyr 8](#_Toc51967893)

[Сортировка данных в dplyr 10](#_Toc51967894)

[Вычисление параметров в dplyr 12](#_Toc51967895)

[Группировка и аггрегация в dplyr 15](#_Toc51967896)

[Обогащение данных с dplyr 18](#_Toc51967897)

# Обработка данных в dplyr

## Почему dplyr

Пакет для статистического анализа данных R славится огромным количеством сторонних пакетов, созданных сообществом разработчиков на языке R по всему миру. Они помогают нам использовать функции на уровне высоких абстракций, для того чтобы производить довольно сложные статистические процедуры, которые бы на обычном императивном языке программирования пришлось бы писать вручную, крайне долго, и, в большинстве случаев, при разработке этих методов большое количество времени приходится проводить за отладкой кода.

При использовании сторонних библиотек крупных сообществ возникает возможность использовать крайне удобный интерфейс для очень сложной логики обработки значительных массивов информации. Среди методов крайне критичных к потреблению вычислительных ресурсов в обработке данных выделяют методы предобработки результатов измерений, а именно:

1. фильтрацию;
2. сортировку;
3. вычисление новых параметров на основе существующих;
4. группировку и аггрегацию;
5. обогащение данных;

Для языка R сообществом разработчиков, создавших Rstudio, был создан пакет dplyr для удобного использования подобного рода функций в коде. Этот пакет является самым популярным среди разработчиков R на фронте преобразования данных по нескольким простым причинам:

* 1. очень неудобный механизм преобразования данных в базовой реализации языка,
  2. очень удачные аспекты применения читаемого кода на уровне преобразования данных в пакете dplyr,
  3. реализация данных процедур на языке С++ в угоду скорости работы.

Все данные особенности побуждают к рассмотрению основных возможностей по предобработке данных в пакете dplyr с целью улучшить качество написанного кода и понимание процедур преобразования результатов наблюдений.

## Загрузка и подключение dplyr к проекту

Для того, чтобы указывать подключение пакета в скрипте R и постоянно не беспокоиться о том, есть ли на другой машине, где будет исполняться данный скрипт, также данный пакет, можно воспользоваться следующей хитрой конструкцией поиска по названию пакета среди предустановленных пакетов.

# Пакет dplyr  
if ("dplyr" %in% rownames(installed.packages()) == FALSE) {  
 install.packages("dplyr")  
}  
library(dplyr)

##   
## Attaching package: 'dplyr'

## The following objects are masked from 'package:stats':  
##   
## filter, lag

## The following objects are masked from 'package:base':  
##   
## intersect, setdiff, setequal, union

Данное решение с проверкой вхождения названия пакета в список установленных пакетов на локальном компьютере позволяет сделать условный переход, пропуская установку из CRAN, если такой пакет установлен и делая загрузку и установку, если данный пакет соответственно отсутствует в списке.

Подобная команда не приведет к загрузке пакета, если локальный компьютер не подключен к сети интернет.

## Концепция конвееризации функций

Конвееризация функций в R - концепция, или способ написания кода при помощи оператора **%>%**, позволяющий улучшить читаемость последовательного применения функций в коде и повысить удобство написания такого кода.

Эта концепция была предложена Hadley Wickham’ом в пакете tidyverse для улучшения восприятия сложных вложенных функциональных конструкций в R, поскольку при использовании широких возможностей функционального программирования часто возникают конструкции типа сложных вложенных функций, когда функция верхнего уровня принимает рекурсивную цепочку операций над одним объектом данных.

Разберёмся как работает векторизация на простом игрушечном примере. Допустим, мы хотим произвести печать медианного значения количества уникальных значений в каждом столбце таблицы данных. Такая возможность реализуется в базовой концепции R такой конструкцией:

# Пакет gapminder  
if ("gapminder" %in% rownames(installed.packages()) == FALSE) {  
 install.packages("gapminder")  
}  
library(gapminder)  
  
# Уровни вложенности показаны для удобства чтения кода  
print(  
 median(  
 apply(gapminder::gapminder, 2, function(x) length(unique(x)))  
 )  
 )

## [1] 884

В коде выше был представлен такой пример для данных важнейших показателей развития стран gapminder <https://www.gapminder.org/>.

Для сравнения представим ниже код для точно такой же процедуры в конвеерном виде:

# Пакет gapminder  
if ("gapminder" %in% rownames(installed.packages()) == FALSE) {  
 install.packages("gapminder")  
}  
library(gapminder)  
  
# Конвееры %>%   
gapminder::gapminder %>%   
 apply(MARGIN = 2, FUN = function(x) length(unique(x))) %>%   
 median() %>%   
 print()

## [1] 884

Данные процедуры выполняют одну и ту же последовательность действий, просто знак конвеера %>% позволяет нам разложить вложенность функций наружу, поставив определённость в последовательности применения функций агрегации над данными. С помощью конвеера %>% можно последовательно читать последовательность операций над data.frame:

1. к gapminder по столбцам применяется функция получения количества уникальных значений (apply(MARGIN = 2, FUN = function(x) length(unique(x))));
2. К результату 1) - вектору данных применяется функция поиска медианы ( median());
3. К результату 2) - числу применяется функция вывода на печать (print()).

Философия обработки данных в **dplyr** похожа на философию обработки данных в языке SQL, за исключением большей произвольности в порядке выпоkнения операций преобразования данных.

Для всех примеров кода для преобразования данных в dplyr мы будем использовать набор данных gapminder::gapminder.

# Первые 6 наблюдений  
head(gapminder::gapminder)

## # A tibble: 6 x 6  
## country continent year lifeExp pop gdpPercap  
## <fct> <fct> <int> <dbl> <int> <dbl>  
## 1 Afghanistan Asia 1952 28.8 8425333 779.  
## 2 Afghanistan Asia 1957 30.3 9240934 821.  
## 3 Afghanistan Asia 1962 32.0 10267083 853.  
## 4 Afghanistan Asia 1967 34.0 11537966 836.  
## 5 Afghanistan Asia 1972 36.1 13079460 740.  
## 6 Afghanistan Asia 1977 38.4 14880372 786.

# Последние 6 наблюдений  
tail(gapminder::gapminder)

## # A tibble: 6 x 6  
## country continent year lifeExp pop gdpPercap  
## <fct> <fct> <int> <dbl> <int> <dbl>  
## 1 Zimbabwe Africa 1982 60.4 7636524 789.  
## 2 Zimbabwe Africa 1987 62.4 9216418 706.  
## 3 Zimbabwe Africa 1992 60.4 10704340 693.  
## 4 Zimbabwe Africa 1997 46.8 11404948 792.  
## 5 Zimbabwe Africa 2002 40.0 11926563 672.  
## 6 Zimbabwe Africa 2007 43.5 12311143 470.

# Структура данных  
str(gapminder::gapminder)

## tibble [1,704 x 6] (S3: tbl\_df/tbl/data.frame)  
## $ country : Factor w/ 142 levels "Afghanistan",..: 1 1 1 1 1 1 1 1 1 1 ...  
## $ continent: Factor w/ 5 levels "Africa","Americas",..: 3 3 3 3 3 3 3 3 3 3 ...  
## $ year : int [1:1704] 1952 1957 1962 1967 1972 1977 1982 1987 1992 1997 ...  
## $ lifeExp : num [1:1704] 28.8 30.3 32 34 36.1 ...  
## $ pop : int [1:1704] 8425333 9240934 10267083 11537966 13079460 14880372 12881816 13867957 16317921 22227415 ...  
## $ gdpPercap: num [1:1704] 779 821 853 836 740 ...

## Фильтрация данных в dplyr

Если говорить прямо, то для всех функций из пространства имён **dplyr::** можно так или иначе применять обе описанные выше концепции написания кода, как стандартный так и конвеерный. Конечно же рекомендуемый способ для написания кода dplyr:: - конвеерный.

Первая операция преобразования данных о которой пойдет речь - фильтрация данных. Фильтрация данных позволяет при помощи логических выражений отсеивать ненужные данные для конкретного рассмотрения. Такая операция производится с помощью функции **dplyr::filter().**

Для сортировки необходимых данных, как было показано в концепции конвееризации, необходимо сначала указать набор данных, над которым необходимо произвести фильтрацию, а затем поставить оператор конвеера **%>%**, после которого будет указана нужная операция обработки, в частности **dplyr::filter( statement )** - фильтрация.

Высказывание внутри фильтра оказывается похожим на высказывание внутри условия if ( statement ), или других конструкций ветвления или цикла.

Произведем например фильтрацию данных, выделив все строки, содержащие наблюдения за 2007 год:

filtered\_data <- gapminder::gapminder %>%   
 dplyr::filter(year == 2007)  
head(filtered\_data)

## # A tibble: 6 x 6  
## country continent year lifeExp pop gdpPercap  
## <fct> <fct> <int> <dbl> <int> <dbl>  
## 1 Afghanistan Asia 2007 43.8 31889923 975.  
## 2 Albania Europe 2007 76.4 3600523 5937.  
## 3 Algeria Africa 2007 72.3 33333216 6223.  
## 4 Angola Africa 2007 42.7 12420476 4797.  
## 5 Argentina Americas 2007 75.3 40301927 12779.  
## 6 Australia Oceania 2007 81.2 20434176 34435.

Как мы видим, остались в рассмотрении только наблюдения за 2007 год, как мы и хотели в данной процедуре.

range(filtered\_data$year)

## [1] 2007 2007

При помощи **dplyr::filter()** можно организовать сколь угодно сложные фильтрации по разным полям данных. Попробуем организовать выборку из исходного набора данных на промежуток времени от 2000 до 2010 года по Азиатскому континету.

# Запись 1  
filtered\_data1 <- gapminder::gapminder %>%   
 dplyr::filter(year >= 2000 & year <= 2010) %>%   
 dplyr::filter(continent == "Asia")  
  
str(filtered\_data1)

## tibble [66 x 6] (S3: tbl\_df/tbl/data.frame)  
## $ country : Factor w/ 142 levels "Afghanistan",..: 1 1 8 8 9 9 19 19 25 25 ...  
## $ continent: Factor w/ 5 levels "Africa","Americas",..: 3 3 3 3 3 3 3 3 3 3 ...  
## $ year : int [1:66] 2002 2007 2002 2007 2002 2007 2002 2007 2002 2007 ...  
## $ lifeExp : num [1:66] 42.1 43.8 74.8 75.6 62 ...  
## $ pop : int [1:66] 25268405 31889923 656397 708573 135656790 150448339 12926707 14131858 1280400000 1318683096 ...  
## $ gdpPercap: num [1:66] 727 975 23404 29796 1136 ...

# Запись 2  
filtered\_data2 <- gapminder::gapminder %>%   
 dplyr::filter(continent == "Asia" & year >= 2000 & year <= 2010)  
  
str(filtered\_data2)

## tibble [66 x 6] (S3: tbl\_df/tbl/data.frame)  
## $ country : Factor w/ 142 levels "Afghanistan",..: 1 1 8 8 9 9 19 19 25 25 ...  
## $ continent: Factor w/ 5 levels "Africa","Americas",..: 3 3 3 3 3 3 3 3 3 3 ...  
## $ year : int [1:66] 2002 2007 2002 2007 2002 2007 2002 2007 2002 2007 ...  
## $ lifeExp : num [1:66] 42.1 43.8 74.8 75.6 62 ...  
## $ pop : int [1:66] 25268405 31889923 656397 708573 135656790 150448339 12926707 14131858 1280400000 1318683096 ...  
## $ gdpPercap: num [1:66] 727 975 23404 29796 1136 ...

Оказывается, что показанные выше записи являются эквивалентными, что логично ввиду безразличия к порядку проведения фильтрации.

Как в случае, представленном выше, стоит понимать, что *порядок фильтрации данных не будет влиять на результат, на заранее определённых критериях фильтрации*.

Чтобы убедиться в идентичности наборов данных попробуем сравнить полученные выборки при помощи определённой для всех структур данных R функции **all.equal()**:

all.equal(filtered\_data1, filtered\_data2)

## [1] TRUE

Таким образом, при помощи конвеерной обработки с помощью функции **dplyr::filter()** имеется возможность быстрого формирования подвыборки по определённым заранее заданным критериям. Такая возможность представляется весьма удобной в случае анализа сложных структур на иерархических категориальных факторах.

# удалим из окружения объявленные подвыборки  
rm("filtered\_data", "filtered\_data1", "filtered\_data2")

Фильтрация данных в **dplyr** также представлена функцией **dplyr::slice()** и её производными, позволяющими сделать произвольные выборки строк по их индексу. Среди производных функций slice можно выделить:

1. ***dplyr::slice()*** - принимает на вход вектор целых значений - индексов строк данных, которые мы хотим выделить в отдельную подтаблицу, что может быть полезно при организации каких-то сложных подвыборок, и крайне эффективно сочетается с принципом дополнения размера вектора при проведении векторных операций;
2. ***dplyr::slice\_max()*** и ***dplyr::slice\_min()*** - принимает на вход переменную, т.е. поле, по которому будут выделены строки с максимальным и минимальным значением по данному выбранному полю переменных для подсчёта актуальных значений и выделения выделяющихся наблюдений из выборки;
3. ***dplyr::slice\_head()*** и ***dplyr::slice\_tail()*** - принимает на вход n первых или последних соответственно строк для выделения данных из исходного набора в подвыборку, что может быть применено при первоначальном исследовании набора данных;
4. ***dplyr::slice\_sample(n = , prop = , replace = T/F)*** - функция, обеспечивающая выбор случайных n строк из исходной подвыборки с возвращением или без, что удобно для выделения обучающих, тестовых и валидационных подвыборок для моделей машинного обучения.

Продемонстрируем на примерах работу функций семейства **slice()**:

Пример. Выборка нечетных строк:

# Выборка из нечетных строк используя дополнение до большего вектора  
sliced\_data1 <- gapminder::gapminder %>%   
 dplyr::slice(seq(1, nrow(gapminder::gapminder), 2)) %>%   
 dplyr::slice\_head(n = 10)  
  
print(sliced\_data1)

## # A tibble: 10 x 6  
## country continent year lifeExp pop gdpPercap  
## <fct> <fct> <int> <dbl> <int> <dbl>  
## 1 Afghanistan Asia 1952 28.8 8425333 779.  
## 2 Afghanistan Asia 1962 32.0 10267083 853.  
## 3 Afghanistan Asia 1972 36.1 13079460 740.  
## 4 Afghanistan Asia 1982 39.9 12881816 978.  
## 5 Afghanistan Asia 1992 41.7 16317921 649.  
## 6 Afghanistan Asia 2002 42.1 25268405 727.  
## 7 Albania Europe 1952 55.2 1282697 1601.  
## 8 Albania Europe 1962 64.8 1728137 2313.  
## 9 Albania Europe 1972 67.7 2263554 3313.  
## 10 Albania Europe 1982 70.4 2780097 3631.

В примере выше нами была проведена не только процедура выделения интервалов измерения по 10 лет для каждой страны, но и для удобства отображения данных результата также отсечены все данные кроме 10 первых.

Пример. Выборка данных за 1952 год:

# Минимум по дате и 6 последних записей  
sliced\_data2 <- gapminder::gapminder %>%   
 dplyr::slice\_min(year) %>%   
 dplyr::slice\_tail(n = 6)  
  
print(sliced\_data2)

## # A tibble: 6 x 6  
## country continent year lifeExp pop gdpPercap  
## <fct> <fct> <int> <dbl> <int> <dbl>  
## 1 Venezuela Americas 1952 55.1 5439568 7690.  
## 2 Vietnam Asia 1952 40.4 26246839 605.  
## 3 West Bank and Gaza Asia 1952 43.2 1030585 1516.  
## 4 Yemen, Rep. Asia 1952 32.5 4963829 782.  
## 5 Zambia Africa 1952 42.0 2672000 1147.  
## 6 Zimbabwe Africa 1952 48.5 3080907 407.

Пример. Выборка случайных 6 записей из набора данных:

# Выборка 6 случайных строк из набора данных  
sliced\_data3 <- gapminder::gapminder %>%   
 dplyr::slice\_sample(n = 6)  
  
print(sliced\_data3)

## # A tibble: 6 x 6  
## country continent year lifeExp pop gdpPercap  
## <fct> <fct> <int> <dbl> <int> <dbl>  
## 1 Poland Europe 1957 65.8 28235346 4734.  
## 2 Ecuador Americas 1967 56.7 5432424 4579.  
## 3 Cote d'Ivoire Africa 1987 54.7 10761098 2157.  
## 4 Sao Tome and Principe Africa 1962 51.9 65345 1072.  
## 5 Cambodia Asia 1957 41.4 5322536 434.  
## 6 Togo Africa 1982 55.5 2644765 1345.

# Очистка от сгенерированных данных  
rm("sliced\_data1", "sliced\_data2", "sliced\_data3")

## Выбор параметров и структуры данных в dplyr

Структура табличных данных - фиксированный набор полей данных с предопределёнными типом, шкалами и назначением, для которого производится написание сценария обработки данных. Структура табличных данных - концепция обработки данных, используя которую крайне удобным оказывается конкретно разработка скрипта или сценария обработки данных. Используя данную концепцию, можно на каждом этапе разработки быть уверенным в логике работы вашей программы и прописывать для данного сценария уникальную последовательность обработки, присущую только данной структуре таблиц проекта.

Понимание основ работы с фиксированными табличными структурами оказывается крайне важным при работе с огромными массивами информации ввиду того, что написанная логика обработки данных может прогоняться каждый раз на новых подвыборках из огромного массива данных без предварительной настройки сценария.

В данной концепции важным является определение перечня ключевых полей по которым будет производится обработка данных в сценарии. В **dplyr** существует возможность обеспечения выбора конкретно подвыборки (подмножества данных по исходному множеству) с фиксированным перечнем полей из исходной выборки данных. Такую функциональность обеспечивает **dplyr::select()**.

В качестве параметров **dplyr::select()** принимает явное перечисление имён полей исходных данных для организации подвыборки.

Пример: выделение и данных населения и средней продолжительности жизни по странам мира:

selected\_data <- gapminder::gapminder %>%   
 dplyr::select(country, year, lifeExp, pop)  
  
head(selected\_data)

## # A tibble: 6 x 4  
## country year lifeExp pop  
## <fct> <int> <dbl> <int>  
## 1 Afghanistan 1952 28.8 8425333  
## 2 Afghanistan 1957 30.3 9240934  
## 3 Afghanistan 1962 32.0 10267083  
## 4 Afghanistan 1967 34.0 11537966  
## 5 Afghanistan 1972 36.1 13079460  
## 6 Afghanistan 1977 38.4 14880372

Таким образом можно отбрасывать привязку наблюдений к конкретным категориям для обеспечения правильной работы преднаписанных сценариев обработки данных.

Можно осуществить ту же самую логику работы примера, представленного выше другим способом, а именно убирая ненужные столбцы, а не выделяя явно нужные. Это действие можно осуществить при помощи знака минуса *“-”* перед указанным полем:

selected\_data1 <- gapminder::gapminder %>%   
 dplyr::select(-continent, -gdpPercap)  
  
head(selected\_data1)

## # A tibble: 6 x 4  
## country year lifeExp pop  
## <fct> <int> <dbl> <int>  
## 1 Afghanistan 1952 28.8 8425333  
## 2 Afghanistan 1957 30.3 9240934  
## 3 Afghanistan 1962 32.0 10267083  
## 4 Afghanistan 1967 34.0 11537966  
## 5 Afghanistan 1972 36.1 13079460  
## 6 Afghanistan 1977 38.4 14880372

# Проверка идентичности операций  
all.equal(selected\_data1, selected\_data)

## [1] TRUE

# Очистка данных  
rm("selected\_data")

## Сортировка данных в dplyr

Сортировка данных - операция, позволяющая упорядочить данные в соответствии определённым иерархическим или порядковым описанием списка столбцов, по которому происходит упорядочивание, в направлении, которое указывается для каждого из столбцов сортировки в отдельности.

Сортировка может проводиться по возрастанию (от меньших к большим в относительных критериях величины) и убыванию (от больших к меньшим в схожем понимании).

Сортировка по списку столбцов или признаков производится “по старшинству”, т.е. упорядоченно, иерархически в соответствии с порядком, указанным в списке полей для сортировки. Таким образом этот порядок настраивается пользователем операции.

Упорядочивание по старшинству производится в явно нетривиальном виде. Для начала производится обычная сортировка по первому столбцу, указанному в списке на выполнение, после чего на выходе операции мы получаем упорядоченные строки данных в соответствии со старшим критерием. Далее внутри данных групп по старшинству, не меняя положения значений старшего столбца производится сортировка по второму столбцу, так чтобы строки выстроились по первому столбцу без изменений в соответствии с правилами перестановки похожих элементов, а по второму столбцу в том направлении, в котором было указано. Таким образом эффект от дальнейших критериев сортировки сводится к минимуму по очевидным причинам.

В пакете dplyr операция упорядочивания может быть произведена с помощью функции **dplyr::arrange()**, в стандартном конвеерном виде, принимающего произвольное количество имен столбцов в натуральном виде, обрамленного функцией **dplyr::desc(colname)** в случае, если мы хотим установить направления упорядочивания по убыванию для данного столбца или не обрамлённого, для установки направления упорядочивания по возрастанию.

Приведём пример сортировки данных для исходных данных **gapminder::gapminder**. Для примера, попробуем отобразить первые 6 самых крупных стран по экономическому развитию.

ordered\_data <- gapminder::gapminder %>%   
 dplyr::filter(year == max(year)) %>%   
 dplyr::arrange(dplyr::desc(gdpPercap)) %>%   
 head()  
  
print(ordered\_data)

## # A tibble: 6 x 6  
## country continent year lifeExp pop gdpPercap  
## <fct> <fct> <int> <dbl> <int> <dbl>  
## 1 Norway Europe 2007 80.2 4627926 49357.  
## 2 Kuwait Asia 2007 77.6 2505559 47307.  
## 3 Singapore Asia 2007 80.0 4553009 47143.  
## 4 United States Americas 2007 78.2 301139947 42952.  
## 5 Ireland Europe 2007 78.9 4109086 40676.  
## 6 Hong Kong, China Asia 2007 82.2 6980412 39725.

В показанном примере мы применили конвеерную обработку для последовательного выполнения нескольких операций преобразования данных – фильтрации, а затем сортировки. Это было важно исходя из преследуемых нами целей, нам нужно было определить самые экономически развитые страны в плане экономики, а для этого нам необходимо просмотреть только актуальную информацию. Чтобы не зависеть от того, какую свежесть данных мы имеем при работе, стоит знать основные стандартные функции агрегации данных, как например при использовании функции **dplyr::filter()** нами была использована функция **max()**.

Далее после фильтрации мы обозначили в сортировке то поле, по которому будет происходить упорядочивание. Передав функцию **dplyr::desc()** поле gpdPercap, мы для функции **dplyr::arrange()** явно указываем направление упорядочивания строк данных по значениям данного столбца.

Также для обеспечения вывода только 6 первых строк данных мы также могли воспользоваться функцией из пакета dplyr, но гораздо проще воспользоваться стандартной функцией **head()**, отбрасывающей все наблюдения, за исключением верхних n строк.

На данном примере было комплексно продемонстрировано решение довольно непростой задачи преобразования данных в довольно простом для чтения виде. В этом и состоим преимущество использования dplyr для обработки данных.

Похожим образом мы будем производить преобразования по конвееру %>% для остальных операций преобразования, освещаемых в данном разделе.

Пример иерархической сортировки:

# Сложная сортировка  
ordered\_data1 <- gapminder::gapminder %>%   
 dplyr::arrange(dplyr::desc(year), dplyr::desc(gdpPercap)) %>%   
 head()  
  
print(ordered\_data1)

## # A tibble: 6 x 6  
## country continent year lifeExp pop gdpPercap  
## <fct> <fct> <int> <dbl> <int> <dbl>  
## 1 Norway Europe 2007 80.2 4627926 49357.  
## 2 Kuwait Asia 2007 77.6 2505559 47307.  
## 3 Singapore Asia 2007 80.0 4553009 47143.  
## 4 United States Americas 2007 78.2 301139947 42952.  
## 5 Ireland Europe 2007 78.9 4109086 40676.  
## 6 Hong Kong, China Asia 2007 82.2 6980412 39725.

Пример выше показывает возможности сортировки по двум столбцам в порядке убывания. Для начала отсортированные данные по годам, вывели в верхние строчки записи, удовлетворяющие условию **(year == 2007)**, что является для данной таблицы максимальным (из предыдущего примера). Далее производятся перестановки строк по убыванию показателя gdpPercap таким образом, чтобы строки с **(year == 2007)** не ушли со своего места в совокупности, хотя между собой им меняться разрешено.

При помощи такой сортировки нами была выполнена та же задача, что и в примере с одной сортировкой выше, т.е. мы так же определили первые 6 стран с самым большим показателем экономического развития на 2007 год.

# Проверка идентичности подвыборок данных  
all.equal(ordered\_data, ordered\_data1)

## [1] TRUE

Это показывает возможность решения реальных задач различными методами, при помощи стандартных операций преобразования данных.

# Удалим выделенные подвыборки из окружения  
rm("ordered\_data", "ordered\_data1")

## Вычисление параметров в dplyr

Вычисление новых параметров или изменение существующих измерений необходимо в обработке наблюдений по нескольким причинам:

1. изменение единиц измерения для абсолютных или относительных шкал;
2. приведение данных к единому формату (стандартизация данных);
3. вычисление новых показателей на основе имеющихся;
4. построение нелинейных аналитических зависимостей;

Для преобразования данных используется функция **dplyr::mutate()**. Она позволяет заменить вычисляемым значением одно из полей таблицы, если имена вычисляемого поля и одного из полей таблицы совпадают, или добавить новое поле, которое добавится справа в качестве нового столбца к таблице данных.

В подобных преобразованиях важно всегда держать в голове о возможности таких преобразований и целях, которые мы преследуем такими действиями. Не над всеми шкалами данных и единицами измерения уместно производить операции, скажем возведения в степень или умножения. Такие преобразования с физическими данными приводят их единицы измерения в не очень реальный вид, с которыми работать в точных науках не имеет смысла, а вычисление новых полей на основе арифметических операций от категориальных или порядковых полей абсолютно некорректно.

Рассмотрим пример вычисления новой переменной в таблице данных **gapminder::gapminder**:

# Приведение существуюущего столбца к новым единицам измерения  
mutated\_data <- gapminder::gapminder %>%   
 dplyr::filter(country == "Hong Kong, China") %>%   
 dplyr::mutate(pop\_millions = pop / 1000000) %>%   
 dplyr::mutate(pop\_diff = c(pop\_millions[1], diff(pop\_millions)))  
  
mutated\_data

## # A tibble: 12 x 8  
## country continent year lifeExp pop gdpPercap pop\_millions pop\_diff  
## <fct> <fct> <int> <dbl> <int> <dbl> <dbl> <dbl>  
## 1 Hong Kong, Ch~ Asia 1952 61.0 2.13e6 3054. 2.13 2.13   
## 2 Hong Kong, Ch~ Asia 1957 64.8 2.74e6 3629. 2.74 0.610  
## 3 Hong Kong, Ch~ Asia 1962 67.6 3.31e6 4693. 3.31 0.569  
## 4 Hong Kong, Ch~ Asia 1967 70 3.72e6 6198. 3.72 0.418  
## 5 Hong Kong, Ch~ Asia 1972 72 4.12e6 8316. 4.12 0.393  
## 6 Hong Kong, Ch~ Asia 1977 73.6 4.58e6 11186. 4.58 0.468  
## 7 Hong Kong, Ch~ Asia 1982 75.4 5.26e6 14561. 5.26 0.681  
## 8 Hong Kong, Ch~ Asia 1987 76.2 5.58e6 20038. 5.58 0.320  
## 9 Hong Kong, Ch~ Asia 1992 77.6 5.83e6 24758. 5.83 0.245  
## 10 Hong Kong, Ch~ Asia 1997 80 6.50e6 28378. 6.50 0.666  
## 11 Hong Kong, Ch~ Asia 2002 81.5 6.76e6 30209. 6.76 0.267  
## 12 Hong Kong, Ch~ Asia 2007 82.2 6.98e6 39725. 6.98 0.218

Для данного примера очень интересным казалось посмотреть рост населения, например Гонг Конга в миллионах человек каждые 5 лет. Для этого было необходимо вычислить новые значение популяции в миллионах человек, приведя поле населения к новым единицам измерения, а затем на основе этого поля посмотреть на первые разности значений населения каждый год.

По свойству первых разностей, значений в новом векторе **diff(pop\_millions)** получается на единичку меньше, поэтому одним из выходов из данной ситуации было дополнение вычисленного нового вектора до размера исходного массива путём склеивания с самым первым значением из **pop\_millions** для единообразия в измеряемых величин.

Рассмотрим ещё один пример, связанный с вычислением общего ВВП на основе существующих полей **pop** - население и **gdpPercap** - ВВП на душу населения. Для всех стран по всем годам

mutated\_data <- gapminder::gapminder %>%   
 dplyr::mutate(pop = pop/1000000) %>%   
 dplyr::mutate(GDP\_mill = pop \* gdpPercap)  
  
head(mutated\_data)

## # A tibble: 6 x 7  
## country continent year lifeExp pop gdpPercap GDP\_mill  
## <fct> <fct> <int> <dbl> <dbl> <dbl> <dbl>  
## 1 Afghanistan Asia 1952 28.8 8.43 779. 6567.  
## 2 Afghanistan Asia 1957 30.3 9.24 821. 7585.  
## 3 Afghanistan Asia 1962 32.0 10.3 853. 8759.  
## 4 Afghanistan Asia 1967 34.0 11.5 836. 9648.  
## 5 Afghanistan Asia 1972 36.1 13.1 740. 9679.  
## 6 Afghanistan Asia 1977 38.4 14.9 786. 11698.

tail(mutated\_data)

## # A tibble: 6 x 7  
## country continent year lifeExp pop gdpPercap GDP\_mill  
## <fct> <fct> <int> <dbl> <dbl> <dbl> <dbl>  
## 1 Zimbabwe Africa 1982 60.4 7.64 789. 6024.  
## 2 Zimbabwe Africa 1987 62.4 9.22 706. 6508.  
## 3 Zimbabwe Africa 1992 60.4 10.7 693. 7423.  
## 4 Zimbabwe Africa 1997 46.8 11.4 792. 9038.  
## 5 Zimbabwe Africa 2002 40.0 11.9 672. 8015.  
## 6 Zimbabwe Africa 2007 43.5 12.3 470. 5783.

Для вычисления показателя общего ВВП (в млн. долларов) для начала мы заменили исходный столбец населения на столбец измеряемый в миллионах человек в рамках страны. Затем, было вычислено произведение двух столбцов, в результате чего мы получили ответ на поставленный вопрос.

В R нет проблем с вычислением больших чисел, поэтому данный столбец можно вычислить и просто в долларах, как было изначально, не вычисляя миллионы населения стран.

mutated\_data <- gapminder::gapminder %>%   
 dplyr::mutate(GDP = pop \* gdpPercap)  
  
range(mutated\_data$GDP)

## [1] 5.278469e+07 1.293446e+13

Для полученных данных получили большой разброс, что и следует ожидать от показателей мировой экономики. Также можно узнать для каких стран в 2007 году можно привести в соответствие максимум и минимум общего ВВП:

# На основе вычисленного поля произведем выбор строк с   
# max(GDP) и min(GDP)  
mutated\_data %>%  
 dplyr::filter(year == 2007) %>%   
 dplyr::filter(GDP == max(GDP) | GDP == min(GDP))

## # A tibble: 2 x 7  
## country continent year lifeExp pop gdpPercap GDP  
## <fct> <fct> <int> <dbl> <int> <dbl> <dbl>  
## 1 Sao Tome and Principe Africa 2007 65.5 199579 1598. 3.19e 8  
## 2 United States Americas 2007 78.2 301139947 42952. 1.29e13

Из показанного примера видно как важно иметь возможность быстро создавать новые показатели из имеющихся для получения более широкого представления о процессе или об изучаемой системе.

rm("mutated\_data")

## Группировка и агрегация в dplyr

**Группировка и агрегация** - два неразрывных понятия, которые позволяют в совокупности отражать некоторыми статистическими функциями характер целой группы, её ценность, нижние уровни, пики, или характеристики распределения значений внутри этих групп.

Важным применением группировки и агрегации является уменьшение величины наблюдений без большой потери точности для изучаемых процессов. Такое применение для данных функций исходит из задач обработки сильно детализированных данных (например, по часам). Такие данные с их показателями при решении глобальной задачи без большой потери точности можно свести к изучению средних показателей от данных наблюдений в пределах дня. Для рассматриваемого промежутка времени, например в 4 года, в целях оценки качества системы, такая потеря точности несущественна, однако количество данных сокращается ровно в 24 раза, что влечёт за собой большое ускорение в работе многих статистических функций или алгоритмов обработки наблюдений.

Также группировка и агрегация служат важной частью практически всех основных статистических процедур, связанных с вычислениями гистограмм, построениями доверительных интервалов, проверками гипотез и т.д. Такие простые для понимания операции являются крайне сильным инструментом в руках опытного аналитика.

Третий немаловажный фактор важности группировки данных это -составление отчетов, составление репрезентативной информации о процессе, помогающей принимать правильные решения по управлению развитием системы.

Группировка - метод, позволяющий уменьшить размер данных (количество строк в таблице наблюдений) за счёт повторяющихся дискретных значений какого-то фактора. Процедура агрегации позволяет над группируемыми строками совершить операцию по отображению важной информации о показателе в целом, не прибегая к перечислению её значений в группируемых строчках. То есть при помощи процедуры агрегации происходит уменьшение количества данных, где исходным данным по определенной группе ставится в соответствие одно значение, отражающее основной описательный статистический показатель (среди них: среднее данных, медиана данных, СКО, дисперсия, квартили и т.д.).

За группировку в пакете dplyr отвечает функция **dplyr::group\_by()**, в которой необходимо указать столбец дискретного (желательно) фактора по которому будет производиться группировка. Результат функции группировки без применения функции агрегации над данными приведет к тому же результату, что обычный вывод значений данных. Таким образом без функции агрегации нет смысла использовать группировку.

Рассмотрим пример группировки без агрегации и убедимся в бесполезности данной операции в R:

grouped\_data <- gapminder::gapminder %>%   
 dplyr::group\_by(continent)  
head(grouped\_data)

## # A tibble: 6 x 6  
## # Groups: continent [1]  
## country continent year lifeExp pop gdpPercap  
## <fct> <fct> <int> <dbl> <int> <dbl>  
## 1 Afghanistan Asia 1952 28.8 8425333 779.  
## 2 Afghanistan Asia 1957 30.3 9240934 821.  
## 3 Afghanistan Asia 1962 32.0 10267083 853.  
## 4 Afghanistan Asia 1967 34.0 11537966 836.  
## 5 Afghanistan Asia 1972 36.1 13079460 740.  
## 6 Afghanistan Asia 1977 38.4 14880372 786.

# Проверка на сопоставление двух подвыборок  
all.equal(nrow(grouped\_data), nrow(gapminder::gapminder))

## [1] TRUE

Рассмотрим теперь операцию агрегации. Для данной операции в dplyr предусмотрена функция **dplyr::summarize()**. Данная функция позволяет нам произвести описание групп в **dplyr::group\_by()** с помощью стандартных статистических процедур агрегации, а также собственных методов агрегации данных. Вообще язык R славится своей совместимостью между пакетами, потому что основные функции из всех пакетов производят операции над всеми основными базовыми структурами данных R. Это может помощь нами использовать не только стандартные методы при агрегации данных.

Рассмотрим пример агрегации данных по континентам, на основе показателей, над которыми проводить агрегацию имеет смысл:

summarized\_data <- gapminder::gapminder %>%   
 dplyr::filter(year == 2007)%>%   
 dplyr::group\_by(continent) %>%   
 dplyr::summarise(mean\_GDPpercap = mean(gdpPercap),  
 sum\_pop = sum(pop),  
 sd\_gdpPercap = sd(gdpPercap),  
 median\_lifeExp = median(lifeExp))

## `summarise()` ungrouping output (override with `.groups` argument)

print(summarized\_data)

## # A tibble: 5 x 5  
## continent mean\_GDPpercap sum\_pop sd\_gdpPercap median\_lifeExp  
## <fct> <dbl> <dbl> <dbl> <dbl>  
## 1 Africa 3089. 929539692 3618. 52.9  
## 2 Americas 11003. 898871184 9713. 72.9  
## 3 Asia 12473. 3811953827 14155. 72.4  
## 4 Europe 25054. 586098529 11800. 78.6  
## 5 Oceania 29810. 24549947 6541. 80.7

Как видно из приведенных данных, самые лучшие показатели качества жизни наблюдаются в странах Океании, Австралии, как по продолжительности жизни, так и по экономическим показателям в среднем на душу населения.

Процедура агрегации данных по континентам была произведена на предварительно отфильтрованных данных по максимальному году наблюдения.

С каждым новым заданием сложность конвееров растёт. В больших по размерности данных как от количества параметров, так и от количества наблюдений, в зависимости также от сложности агрегации, сложность самих конвееров будет только расти, а читаемость кода при этом будет сохраняться из-за линейного расположения порядка применения преобразований.

Рассмотрим более глубокое описание экономических показателей стран мира:

economic\_right <- gapminder::gapminder %>%   
 dplyr::mutate(GDP\_mill = pop \* gdpPercap / 1000000) %>%   
 dplyr::group\_by(country) %>%   
 dplyr::summarise(mean\_gdp\_m = mean(GDP\_mill),  
 mean\_diff\_gdp\_m = mean(diff(GDP\_mill)),  
 mean\_gdpPercap = mean(gdpPercap),  
 mean\_diff\_gdpPercap = mean(diff(gdpPercap))) %>%   
 dplyr::arrange(desc(mean\_gdpPercap))

## `summarise()` ungrouping output (override with `.groups` argument)

head(economic\_right)

## # A tibble: 6 x 5  
## country mean\_gdp\_m mean\_diff\_gdp\_m mean\_gdpPercap mean\_diff\_gdpPercap  
## <fct> <dbl> <dbl> <dbl> <dbl>  
## 1 Kuwait 57946. 9199. 65333. -5552.  
## 2 Switzerland 178220. 19309. 27074. 2070.  
## 3 Norway 112736. 17712. 26747. 3569.  
## 4 United States 6396827. 975474. 26261. 2633.  
## 5 Canada 592346. 94967. 22411. 2268.  
## 6 Netherlands 315928. 46994. 21749. 2532.

В данном примере мы на основе группировки и агрегации данных имеем возможность оценить на данных по времени средние показатели ВВП и ВВП на душу населения, а так же средние первые разности от ВВП и ВВП на душу населения во времени, что экономически показывает средние приросты в год по данному показателю, в случае если величина положительная и среднее уменьшение каждый год с 1952 года по настоящее время, если величина отрицательная.

Например, у Кювейта наблюдается рост общего ВВП и снижение ВВП на душу населения, что связано с большим приростом численности при практически константных ресурсах относительно приростов. Это может быть фатальным для экономического развития страны.

Таким образом, важным выводом из данных примеров является большая польза от статистических агрегаций данных на поприще первичного анализа наборов данных, поскольку они позволяют качественно проследить за динамикой наблюдаемой системы, или посмотреть на ограничения и внутренние механизмы изучаемой системы.

# Очистка памяти от подвыборок  
rm("grouped\_data", "summarized\_data", "economic\_right")

## Обогащение данных с dplyr

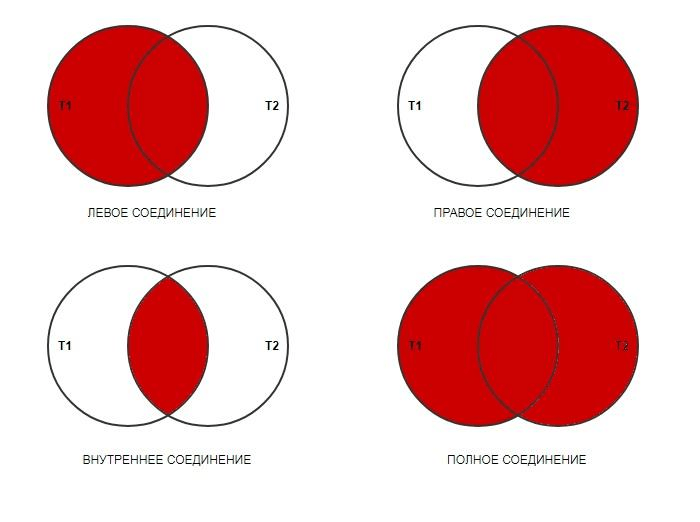
Операции обогащения данных в dplyr представлены небогатым ассортиментом различных функций со стандартным числом их реализаций. Так же как и в других статистических пакетах основными функциями обогащения данных в dplyr являются:

1. cлияние (join);
2. объединение (union);

***Слияние данных***

Важнейшими среди всех операций обогащения являются слияния, обозначающиеся как **join()** . Выделяют различные их реализации в соответствии с теорией множеств:

1. внутреннее соединение - обогащение на основе совпадающих значений ключевых полей в правой и левой таблицах данных;
2. левое соединение – значения присоединяемой таблицы добавляются по строкам к главной таблице только по наличию совпадения значений в ключевых полях, так что возможны пропуски;
3. правое соединение – то же самое что и левое, но таблицы меняются местами;
4. полное соединение – итоговая таблица будет содержать строки данных для которых не нашлось совпадений как при правом так и при левом соединении, причём эти строки будут перечислены наряду с соединенными относительно ключевых полей строками;



Пример внутреннего слияния данных на основе стран, дополняя данные столбцами кодов стран и их номеров ISO.

# inner\_join внутреннее слияние  
joined\_data <- gapminder::gapminder %>%   
 dplyr::inner\_join(y = gapminder::country\_codes, by = "country")  
  
str(joined\_data)

## tibble [1,704 x 8] (S3: tbl\_df/tbl/data.frame)  
## $ country : chr [1:1704] "Afghanistan" "Afghanistan" "Afghanistan" "Afghanistan" ...  
## $ continent: Factor w/ 5 levels "Africa","Americas",..: 3 3 3 3 3 3 3 3 3 3 ...  
## $ year : int [1:1704] 1952 1957 1962 1967 1972 1977 1982 1987 1992 1997 ...  
## $ lifeExp : num [1:1704] 28.8 30.3 32 34 36.1 ...  
## $ pop : int [1:1704] 8425333 9240934 10267083 11537966 13079460 14880372 12881816 13867957 16317921 22227415 ...  
## $ gdpPercap: num [1:1704] 779 821 853 836 740 ...  
## $ iso\_alpha: chr [1:1704] "AFG" "AFG" "AFG" "AFG" ...  
## $ iso\_num : int [1:1704] 4 4 4 4 4 4 4 4 4 4 ...

В результате данного действия данные, находящиеся в главной таблице, были дополнены записями из таблицы кодов стран и их кодов в ISO на основе сопоставления названий стран. Можно проверить, что в результате слияния никакие данные не были потеряны:

joined\_data %>%   
 is.na() %>%   
 any()

## [1] FALSE

Также можно проверить, являются ли данные, полученные в результате данного слияния такими же в своей основной части.

joined\_data$country <- as.factor(joined\_data$country)  
all.equal(gapminder::gapminder, joined\_data[,-(7:8)])

## [1] TRUE

# left\_join левое слияние  
leftjoined\_data <- gapminder::gapminder %>%   
 dplyr::left\_join(y = gapminder::country\_codes, by = "country")  
  
str(leftjoined\_data)

## tibble [1,704 x 8] (S3: tbl\_df/tbl/data.frame)  
## $ country : chr [1:1704] "Afghanistan" "Afghanistan" "Afghanistan" "Afghanistan" ...  
## $ continent: Factor w/ 5 levels "Africa","Americas",..: 3 3 3 3 3 3 3 3 3 3 ...  
## $ year : int [1:1704] 1952 1957 1962 1967 1972 1977 1982 1987 1992 1997 ...  
## $ lifeExp : num [1:1704] 28.8 30.3 32 34 36.1 ...  
## $ pop : int [1:1704] 8425333 9240934 10267083 11537966 13079460 14880372 12881816 13867957 16317921 22227415 ...  
## $ gdpPercap: num [1:1704] 779 821 853 836 740 ...  
## $ iso\_alpha: chr [1:1704] "AFG" "AFG" "AFG" "AFG" ...  
## $ iso\_num : int [1:1704] 4 4 4 4 4 4 4 4 4 4 ...

# right\_join правое слияние  
rightjoined\_data <- gapminder::gapminder %>%   
 dplyr::right\_join(y = gapminder::country\_codes, by = "country")  
  
str(rightjoined\_data)

## tibble [1,749 x 8] (S3: tbl\_df/tbl/data.frame)  
## $ country : chr [1:1749] "Afghanistan" "Afghanistan" "Afghanistan" "Afghanistan" ...  
## $ continent: Factor w/ 5 levels "Africa","Americas",..: 3 3 3 3 3 3 3 3 3 3 ...  
## $ year : int [1:1749] 1952 1957 1962 1967 1972 1977 1982 1987 1992 1997 ...  
## $ lifeExp : num [1:1749] 28.8 30.3 32 34 36.1 ...  
## $ pop : int [1:1749] 8425333 9240934 10267083 11537966 13079460 14880372 12881816 13867957 16317921 22227415 ...  
## $ gdpPercap: num [1:1749] 779 821 853 836 740 ...  
## $ iso\_alpha: chr [1:1749] "AFG" "AFG" "AFG" "AFG" ...  
## $ iso\_num : int [1:1749] 4 4 4 4 4 4 4 4 4 4 ...

# full\_join полное слияние  
fulljoined\_data <- gapminder::gapminder %>%   
 dplyr::full\_join(y = gapminder::country\_codes, by = "country")  
  
str(fulljoined\_data)

## tibble [1,749 x 8] (S3: tbl\_df/tbl/data.frame)  
## $ country : chr [1:1749] "Afghanistan" "Afghanistan" "Afghanistan" "Afghanistan" ...  
## $ continent: Factor w/ 5 levels "Africa","Americas",..: 3 3 3 3 3 3 3 3 3 3 ...  
## $ year : int [1:1749] 1952 1957 1962 1967 1972 1977 1982 1987 1992 1997 ...  
## $ lifeExp : num [1:1749] 28.8 30.3 32 34 36.1 ...  
## $ pop : int [1:1749] 8425333 9240934 10267083 11537966 13079460 14880372 12881816 13867957 16317921 22227415 ...  
## $ gdpPercap: num [1:1749] 779 821 853 836 740 ...  
## $ iso\_alpha: chr [1:1749] "AFG" "AFG" "AFG" "AFG" ...  
## $ iso\_num : int [1:1749] 4 4 4 4 4 4 4 4 4 4 ...

***Объединение данных***

*Объединение данных* - способ обогащения данных, при котором к строкам одной таблицы данных снизу добавляются строки другой таблицы данных одинаковой структуры. Обеспечение единой структуры данных при объединении является важным для корректной работы функции по обогащению данных снизу по строкам.

Операция объединение в dplyr может быть произведена при помощи функций:

1. **dplyr::union(x, y)** - объединение данных по строкам исключая из рассмотрения повторяющиеся строки
2. **dplyr::union\_all(x, y)** - объединение данных по строкам не исключая из рассмотрения повторяющиеся строки

Пример объединения данных схожей структуры.

# union - объединение, объединение данных с исключением похожих строк  
unioned\_data <- gapminder::gapminder %>%   
 dplyr::union(y = gapminder::gapminder\_unfiltered)  
  
str(unioned\_data)

## tibble [3,313 x 6] (S3: tbl\_df/tbl/data.frame)  
## $ country : Factor w/ 187 levels "Afghanistan",..: 1 1 1 1 1 1 1 1 1 1 ...  
## $ continent: Factor w/ 6 levels "Africa","Americas",..: 3 3 3 3 3 3 3 3 3 3 ...  
## $ year : int [1:3313] 1952 1957 1962 1967 1972 1977 1982 1987 1992 1997 ...  
## $ lifeExp : num [1:3313] 28.8 30.3 32 34 36.1 ...  
## $ pop : int [1:3313] 8425333 9240934 10267083 11537966 13079460 14880372 12881816 13867957 16317921 22227415 ...  
## $ gdpPercap: num [1:3313] 779 821 853 836 740 ...

# union\_all - полное объединение без исключения повторений  
unioned\_data <- gapminder::gapminder %>%   
 dplyr::union\_all(y = gapminder::gapminder\_unfiltered)  
  
str(unioned\_data)

## tibble [5,017 x 6] (S3: tbl\_df/tbl/data.frame)  
## $ country : Factor w/ 187 levels "Afghanistan",..: 1 1 1 1 1 1 1 1 1 1 ...  
## $ continent: Factor w/ 6 levels "Africa","Americas",..: 3 3 3 3 3 3 3 3 3 3 ...  
## $ year : int [1:5017] 1952 1957 1962 1967 1972 1977 1982 1987 1992 1997 ...  
## $ lifeExp : num [1:5017] 28.8 30.3 32 34 36.1 ...  
## $ pop : int [1:5017] 8425333 9240934 10267083 11537966 13079460 14880372 12881816 13867957 16317921 22227415 ...  
## $ gdpPercap: num [1:5017] 779 821 853 836 740 ...

Необходимо провести эксперимент на возможность проведения объединения данных с неполным перечнем столбцов:

if ("testthat" %in% rownames(installed.packages()) == FALSE) {  
 install.packages("testthat")  
}  
library(testthat) # пакет тестирования  
  
testthat::expect\_error(  
 gapminder::gapminder %>%   
 dplyr::select(-gdpPercap) %>%   
 dplyr::union(gapminder::gapminder\_unfiltered)   
)

Если обозначенный тест на ошибку из **testthat::expect\_error()** завершился с успехом, значит данная запись неверна и мы просто не можем объединять данные от разных структур таблиц наблюдений.

rm("joined\_data", "fulljoined\_data", "rightjoined\_data", "leftjoined\_data", "unioned\_data")