
Основы статистической обработки педагогической информации
Д.В.СОЛОМАТИН
16+

Денис Соломатин

# Основы статистической обработки педагогической информации

«ЛитРес: Самиздат»

2020

**Соломатин Д. В.**

Основы статистической обработки педагогической информации / Д. В. Соломатин — «ЛитРес: Самиздат», 2020

ISBN 978-5-532-04389-3

Учебное пособие содержит текстовые сведения, иллюстрации и задания по основам статистической обработки педагогической информации в R, вольный пересказ содержимого сайта r4ds.had.co.nz, многие годы аккумулирующего труды исследователей всего мира, с занимательными дополнениями и историческими справками в попытке адаптации материала под профессиональные нужды современных онлайн-учителей. Последняя глава посвящена изучению возможностей R, позволяющих открыть собственную онлайн-школу.

ISBN 978-5-532-04389-3

# Содержание

Доброй памяти В.А.Стукалова, приоткрывшего дверь автору в бескрайний мир математической статистики, посвящается.

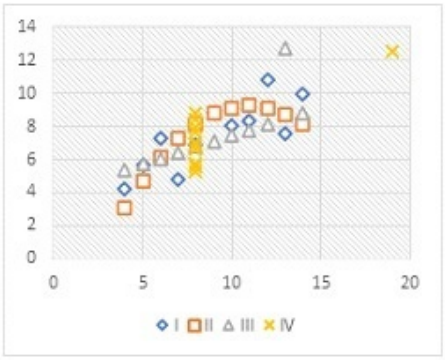
# Введение

Несомненно, в цифровую эпоху тезис «кто владеет информацией – владеет миром» обретает новые интерпретации, но одного только факта владения становится недостаточным, когда объёмы информации колоссальны и осмысление её без специального инструментария не представляется возможным. Настоящее учебное пособие являет собой дополнение к курсу теории вероятностей и математической статистики, попытку авторской систематизации опыта работы и изложения материала, адаптированного к анализу педагогических данных. На волне глобализации школьный онлайн-учитель в ходе своей профессиональной деятельности сталкивается с необходимостью статистической обработки информации, когда на смену традиционным классам ограниченного объема на виртуальные уроки приходят многомилионные аудитории подписчиков из социальных сетей. О чём вы думаете, когда видите американского солдата в экзоскелете и очках дополненной реальности способного переносить грузы значительно превышающие пределы человеческих возможностей и вести наблюдение через непрозрачные стены, либо израильского хирурга, проводящего сложнейшую операцию дистанционно, либо арабского полицейского на джетпаках патрулирующего небоскребы в эмиратах? Без сопутствующего высокотехнологического оборудования ничто из перечисленного не было бы возможным, так и современный онлайн-учитель определенно получает некоторые преимущества лишь освоив соответствующие технологии анализа данных. В первую очередь, наглядную визуализацию. На сегодняшний день в мире не так много научных организаций, целенаправленно занимающихся вопросами визуализации. Из ведущих лабораторий на память приходят: Electronic Visualization Laboratory, Kitware, Лос-Аламосская национальная лаборатория, Подразделение Передовых Суперкомпьютеров NASA, Национальный центр суперкомпьютерных приложений, Сандийские национальные лаборатории, Центр Суперкомпьютеров Сан Диего, Научный институт вычислений и визуальной информации, Техасский Центр передовых вычислительных систем. Специализированных конференций и того меньше: IEEE Visualization, SIGGRAPH, EuroVis, Конференция по вопросам влияния человеческого фактора на компьютерные системы, Eurographics, PacificVis. Отрадно сознавать, что с недавних пор сей список пополнил и ОмГПУ. Необходимость наглядного представления педагогической информации обусловлена самой природой человека, получающего порядка 80%-90% данных с помощью зрения. Наглядность важна и для понимания, весомым подтверждением тому является небезызвестный «квартет Энскомба», составленный в 1973 году английским математиком Ф. Дж. Энскомбом для иллюстрации важности применения графиков для статистического анализа и влияния выбросов значений на свойства всего набора данных. А именно, следующие четыре набора данных имеют идентичные статистические характеристики, но их графики существенно различаются:

| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| I | $x$ | 10.0 | 8.0 | 13.0 | 9.0 | 11.0 | 14.0 | 6.0 | 4.0 | 12.0 | 7.0 | 5.0 |
| | $y$ | 8.04 | 6.95 | 7.58 | 8.81 | 8.33 | 9.96 | 7.24 | 4.26 | 10.84 | 4.82 | 5.68 |
| II | $x$ | 10.0 | 8.0 | 13.0 | 9.0 | 11.0 | 14.0 | 6.0 | 4.0 | 12.0 | 7.0 | 5.0 |
| | $y$ | 9.14 | 8.14 | 8.74 | 8.77 | 9.26 | 8.1 | 6.13 | 3.1 | 9.13 | 7.26 | 4.74 |
| III | $x$ | 10.0 | 8.0 | 13.0 | 9.0 | 11.0 | 14.0 | 6.0 | 4.0 | 12.0 | 7.0 | 5.0 |
| | $y$ | 7.46 | 6.77 | 12.74 | 7.11 | 7.81 | 8.84 | 6.08 | 5.39 | 8.15 | 6.42 | 5.73 |
| IV | $x$ | 8.0 | 8.0 | 8.0 | 8.0 | 8.0 | 8.0 | 8.0 | 19.0 | 8.0 | 8.0 | 8.0 |
| | $y$ | 6.58 | 5.76 | 7.71 | 8.84 | 8.47 | 7.04 | 5.25 | 12.5 | 5.56 | 7.91 | 6.89 |

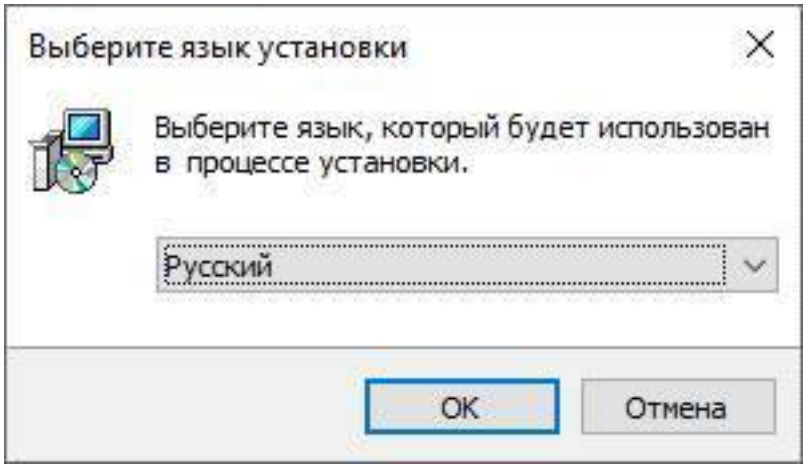
Табличное представление данных естественно, эффективно и удобно для хранения в памяти электронной вычислительной машины, но для осознания представленной информации человеком не обойтись без описательной статистики и главное – графиков:

| Характеристика | Значение |
|---|---|
| Среднее значение переменной $x$ | 9.0 |
| Дисперсия переменной $x$ | 10.0 |
| Среднее значение переменной $y$ | 7.5 |
| Дисперсия переменной $y$ | 3.75 |
| Корреляция между переменными $x$ и $y$ | 0.816 |
| Прямая линейной регрессии | $y=3+0.5x$ |
| Коэффициент детерминации линейной регрессии | 0.67 |



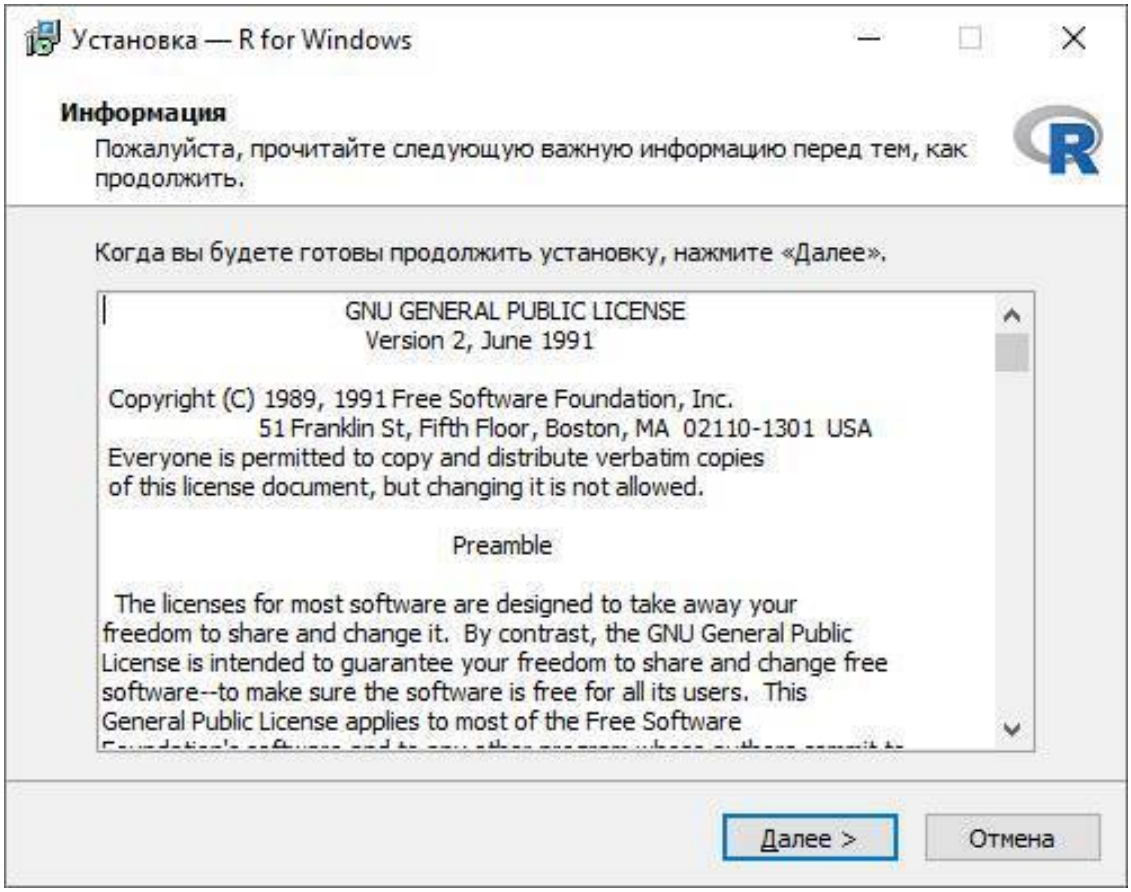

В качестве вводного примера для иллюстрации возможностей анализа и визуализации педагогической информации приведем процесс установки и запуска визуализации средствами пакета R. Предположим, что вами уже создан и активно используется онлайн-курс по математике в iSpring Suite (https://www.ispring.ru/ispring-suite) с активным использованием GeoGebra (https://www.geogebra.org). В некоторый момент вы понимаете, что объем слушателей курса превосходит ваши физические возможности для анализа успеваемости, необходимо использовать программные средства специального назначения. Как быть?

Для решения обозначенной проблемы скачайте и установите пакет R для используемой вами операционной системы (https://www.r-project.org). Прямая ссылка с сайта МГУ: https://cran.cmm.msu.ru/bin/windows/base/R-4.0.2-win.exe При первом запуске процесса установки будет предложен выбор языка установки:

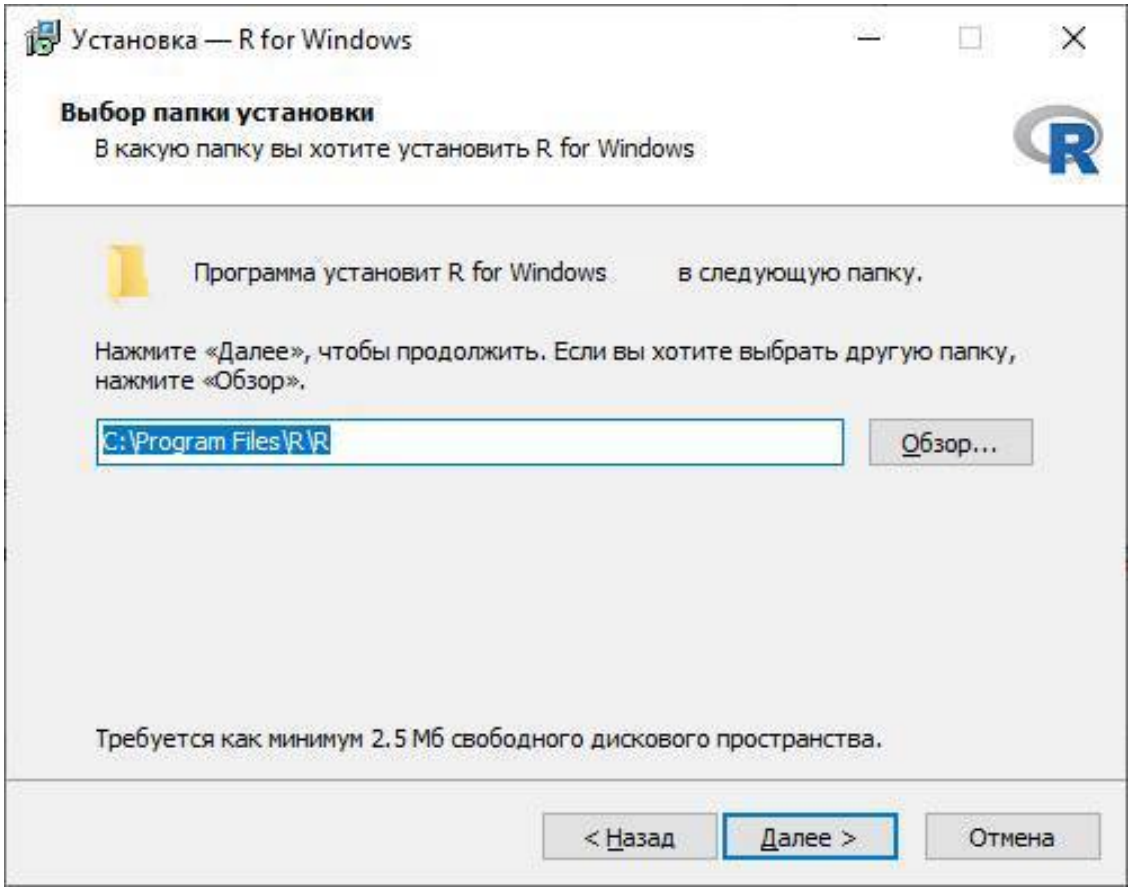



Далее важный момент, будет предложено ознакомиться с условиями универсальной общественной лицензии GNU, по которой распространяется R. Для тех, кто не знаком с ней, это лицензия на свободное программное обеспечение, созданная в рамках проекта GNU в 1988 г., по которой автор передаёт программное обеспечение в «общественную собственность». Её также сокращённо называют GNU GPL или даже просто GPL, если из контекста понятно, что речь идёт именно о данной лицензии. Как видите, R распространяется на правах второй версии этой лицензии, которая была выпущена в 1991 году. Суть в том, что в 1990 году стало очевидным наличие менее ограничивающей лицензии, которая могла бы использоваться для

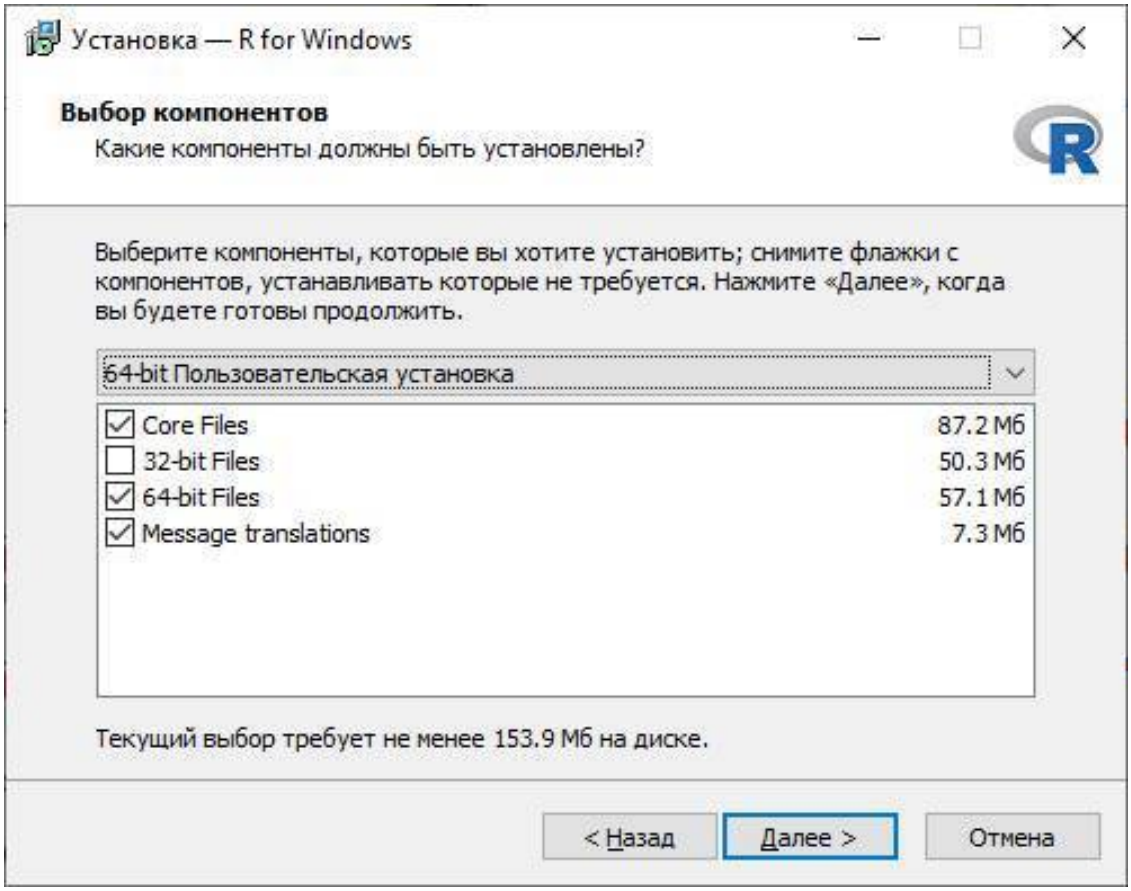
некоторых библиотек ПО; когда версия 2 GPL была выпущена в июне 1991 года, была введена в обращение и GNU Library General Public License (GNU LGPL, LGPL), также получившая номер 2, для обозначения того, что эти две лицензии являются взаимодополняющими.



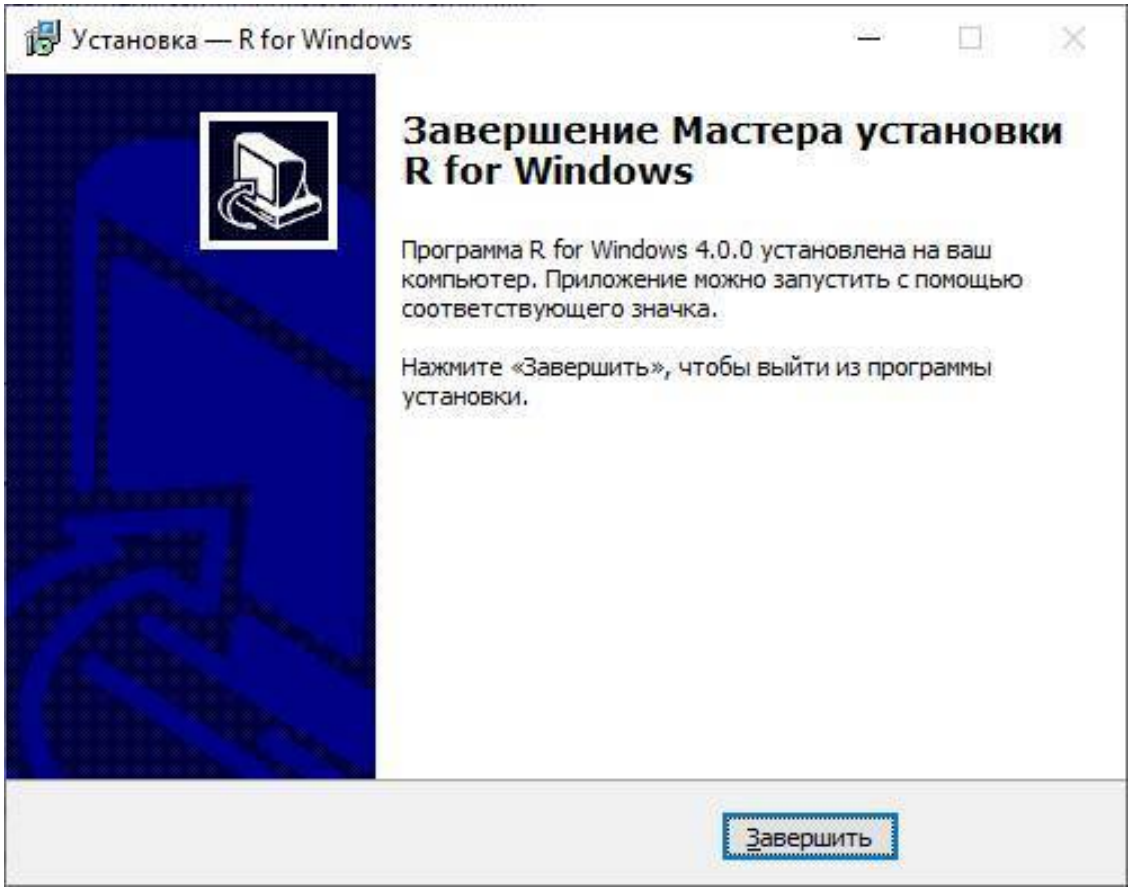

Далее следует выбор папки установки:




Следующий важный момент – выбор разрядности приложения. Основное и едва ли не единственное отличие x64 от x32 в том, что версия x64 может работать с памятью вплоть до 32 Гбайт и запускать одновременно и 64-битные, и 32-битные приложения, тогда как традиционная x32 способна адресовать лишь до 4 Гбайт памяти, запускать только 32-битные программы для которых доступно только 3 Гбайт (говоря проще, даже если в компьютере 4 Гбайт (и более) памяти, то 32-битная система будет отображать и работать лишь с 3-мя, а остальная память будет попросту простаивать, ибо ни система, ни программы, попросту её не увидят). Как видите, отличия лишь в допустимых объемах обрабатываемой информации и общей производительности системы.

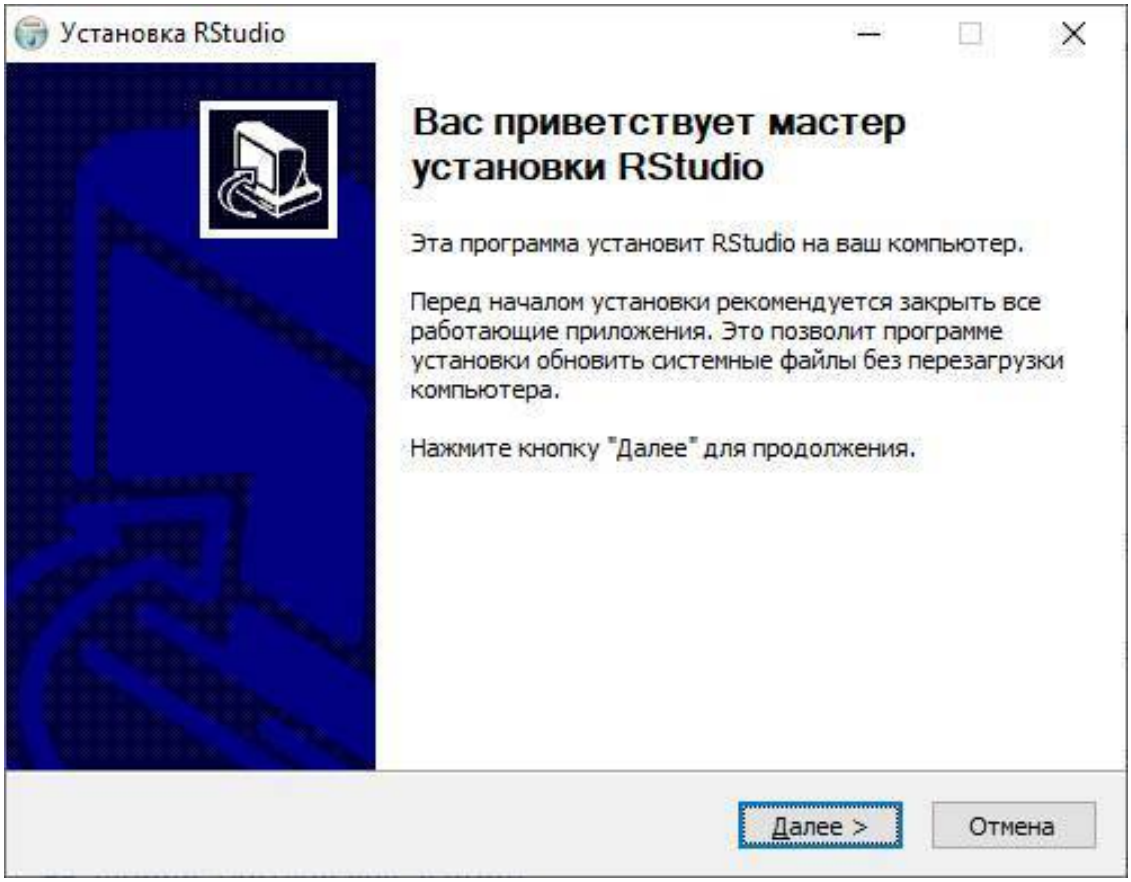



Настройки запуска программы и дополнительные опции не принципиальны. Предлагаемое на следующих экранах по умолчанию можно оставить без изменений, на том установка будет завершена.

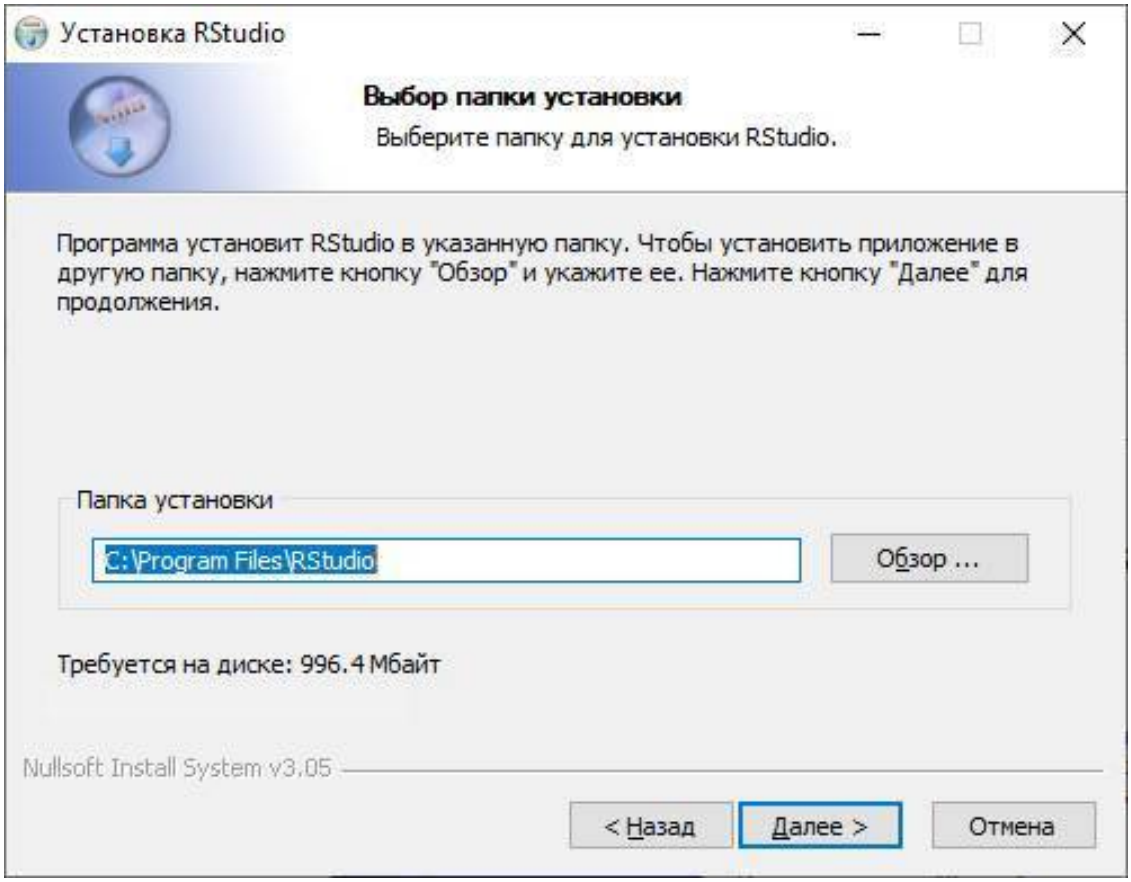



Вторым шагом скачайте и установите RStudio (https://rstudio.com) – этот этап можно пропустить, пока не планируете профессионального освоения данного инструмента. Тем не менее, имейте ввиду, что RStudio в первую очередь это свободная среда разработки программного обеспечения с открытым исходным кодом для языка программирования R. Дистрибутивы RStudio Desktop доступны для Linux, OS X и Windows. Более того, в RStudio Server предоставляется доступ через браузер к RStudio установленной на удаленном Linux-сервере.

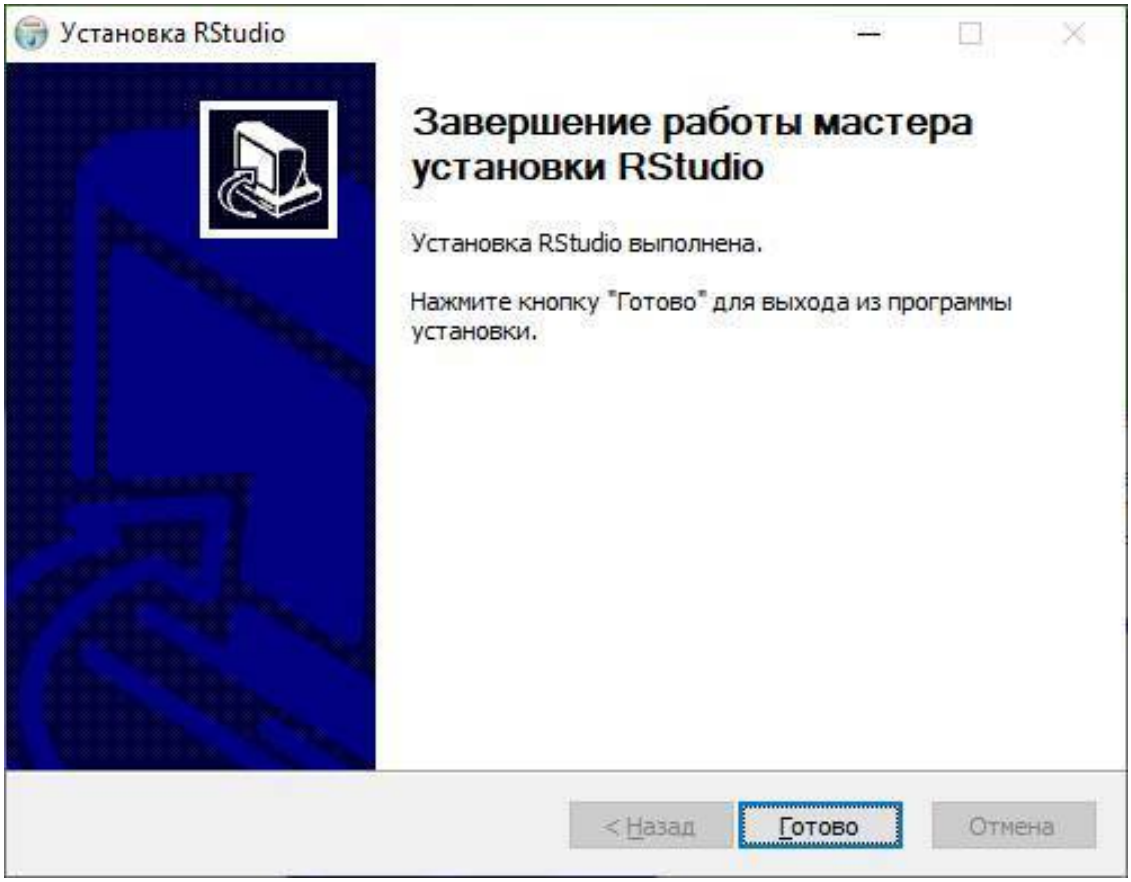
Процесс установки RStudio не многим отличается от установки R.



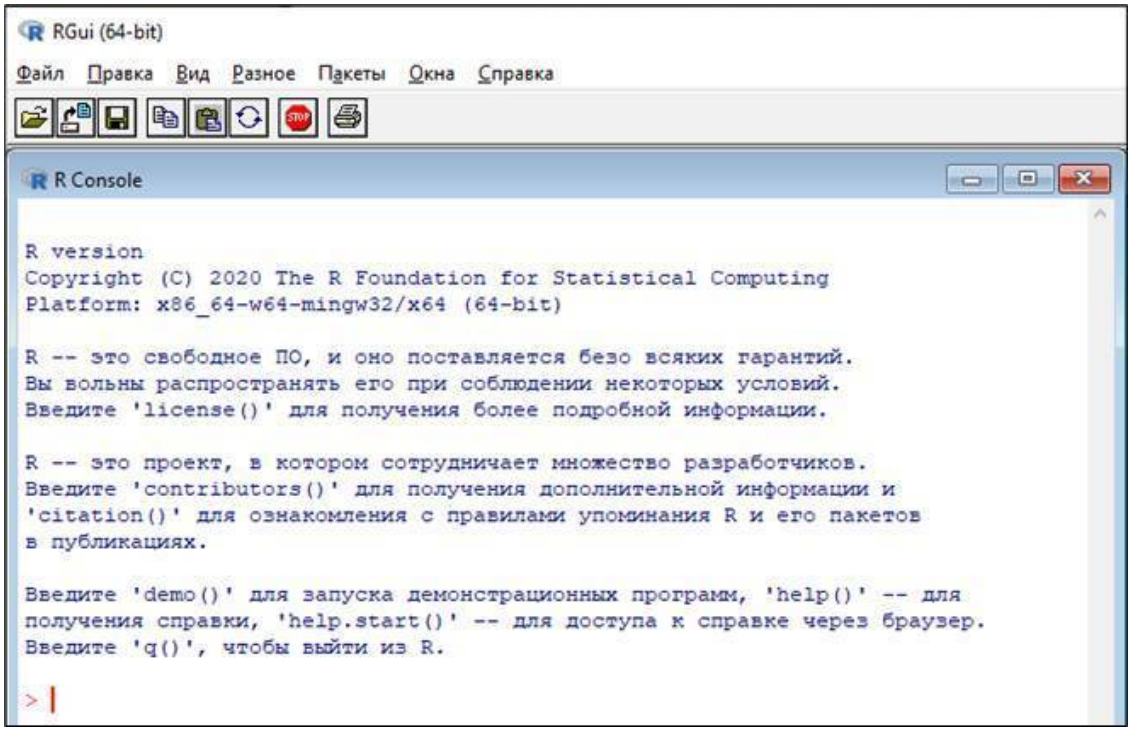

В на подготовительном этапе предлагается выбрать папку установки на диске, имеющем достаточно свободного места для развёртывания дистрибутива.




Последующие этапы позволяют выбрать опциональное создание ярлыков программы, по умолчанию располагаемых в меню «Пуск», и наблюдать за процессом завлечения и копирования файлов RStudio в папку назначения. Так как RStudio написана на языке программирования C++ и использует фреймворк Qt для графического интерфейса пользователя, то в операционную систему будут установлены все необходимые для запуска дополнительные библиотеки сторонних разработчиков. Наиболее крупной из которых, в частности, является Qt5WebEngineCore.dll. По окончанию процесса копирования работа мастера установки будет завершена.




Если пользоваться R без RStudio, то запускаем консоль в Пуск/R/R x64, после разрядности записывается номер установленной версии программного комплекса. Интерфейс будет чуть проще чем RStudio, но для решения элементарных задач достаточным:

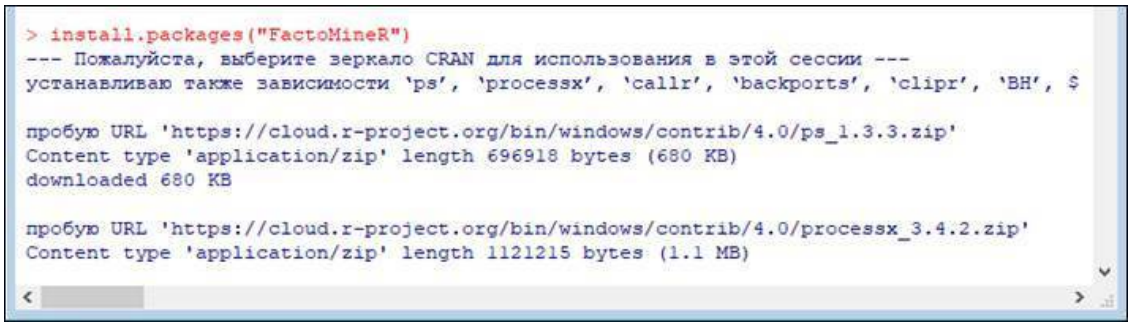


Третьим шагом установите надстройку для многофакторного анализа (http://factominer.free.fr), для этого в консоли R просто вводится команда

install.packages("FactoMineR")

install.packages("Factoshiny")

после чего выбирается зеркало для загрузки надстройки.




Для включения пакета следует выбрать пункт меню «Пакеты/Включить пакет…/FactoMineR/ОК», либо установить в пункте меню «Пакеты/установить пакеты…/Factoshiny/ОК». После установки консоль готова для работы.

Кратко опишем, в каких целях используется пакет Factoshiny. Не секрет, что качественная графическая иллюстрация зачастую говорит больше, чем длинная речь оратора, поэтому крайне важно улучшать графики, полученные любыми основными компонентами методами (например: Метод главных компонент «PCA», Анализ соответствий «CA», Анализ повторяющихся соответствий «MCA», Многофакторный анализ «MFA», и многие другие). Factoshiny позволяет легко улучшить графики в интерактивном режиме. Этот удобный интерфейс позволяет параметризовать используемые методы и изменять графические параметры. При этом не нужно знать, как программировать.

Внося изменения в интерактивном режиме через интуитивно понятный интерфейс, становится очевидным, как улучшаются соответствующие графики. Результаты настроек графиков и параметров обновляются автоматически. В дальнейшем можно скачать получившиеся графики, а также строки настроенного кода, чтобы повторить анализ. Кроме того, можно сохранить, а затем повторно использовать объект, полученный из Factoshiny, для дальнейшей модификации графиков. При каждом новом запуске интерфейс открывается с теми настройками, которые были выбраны при последнем выходе из программы, следовательно, быстро можно продолжить изменение параметров выбранного метода факторного анализа или визуализации графиков.

Подготовив рабочее окружение, можно в качестве демонстрации из Google-документов выгрузить Excel-таблицу table.xlsx успеваемости своего онлайн-класса (с оценками для 5-7 учеников по 7-10 темам) и выполнить анализ данных созданной электронной таблицы средствами R. Для этого достаточно ввести следующую серию команд в консоли R (начинающиеся с символа # строки пропускаются, так как воспринимаются системой в качестве комментариев, подробнее необходимость комментирования исходных кодов будет обоснована в следующих разделах):

1) подключаем библиотеку импорта данных из .xls

library(readxl)

2) подключаем библиотеку многофакторного анализа

library(Factoshiny)

3) загружаем в переменную My_table содержимое файла table.xlsx

My_table <– read_excel("C:/путь к файлу/table.xlsx")

4) запускаем графический интерфейс для визуальной настройки и получения статотчетов PCA, в примере 1, 2, 3, 4, 5, 6, 7, 8 – номера импортируемых колонок из электронной таблицы My_table

PCAshiny(My_table[,c(1, 2, 3, 4, 5, 6, 7, 8)])

5) делаем выводы на предмет ведущих факторов, тем, вызвавших наибольшие/наименьшие затруднения учащихся и их взаимовлияния, тенденции развития.

Предположим, что электронный журнал, экспортированный в файл D:\test.xlsx содержит следующие данные об успеваемости обучающихся в 7а и 7б классах:

| Класс | Фимилия Имя | Тема1 | Тема2 | Тема3 | Тема4 | Тема 5 | №№ |
|---|---|---|---|---|---|---|---|
| 7а | Иванов Иван | 5 | 2 | 1 | 4 | 1 | 1 |
| 7а | Петров Петр | 5 | 3 | 2 | 5 | 2 | 2 |
| 7а | Сидоров Сидор | 5 | 3 | 2 | 5 | 2 | 3 |
| 7а | Егоров Егор | 5 | 2 | 1 | 4 | 2 | 4 |
| 7а | Романов Роман | 5 | 2 | 1 | 4 | 1 | 5 |
| 7а | Николаев Николай | 5 | 3 | 2 | 4 | 2 | 6 |
| 7а | Григорьев Григогий | 5 | 3 | 2 | 5 | 1 | 7 |
| 7а | Викторов Виктор | 5 | 2 | 1 | 5 | 1 | 8 |
| 7а | Михайлов Михаил | 5 | 2 | 2 | 5 | 2 | 9 |
| 7а | Тимуриев Тимур | 5 | 3 | 2 | 4 | 2 | 10 |
| 7б | Ульянова Ульяна | 5 | 4 | 1 | 5 | 1 | 11 |
| 7б | Ольгина Ольга | 5 | 5 | 2 | 5 | 2 | 12 |
| 7б | Людмилова Людмила | 5 | 5 | 2 | 4 | 2 | 13 |
| 7б | Дарьева Дарья | 5 | 4 | 1 | 4 | 1 | 14 |
| 7б | Кристинина Кристина | 5 | 4 | 1 | 5 | 1 | 15 |
| 7б | Натальина Наталья | 5 | 4 | 2 | 5 | 2 | 16 |
| 7б | Глафирова Глафира | 5 | 5 | 2 | 4 | 2 | 17 |
| 7б | Янина Яна | 5 | 5 | 2 | 4 | 1 | 18 |
| 7б | Иринова Ирина | 5 | 4 | 1 | 5 | 2 | 19 |
| 7б | Валентинова Валентина | 5 | 5 | 2 | 4 | 5 | 20 |
| эталон | Идеальный ученик | 5 | 5 | 5 | 5 | 5 | 21 |
| отстающий | Другая крайность | 1 | 1 | 1 | 1 | 1 | 22 |

Запустим RStudio с предустановленными пакетами многофакторного анализа и в консоли R введём серию команд:

library(readxl)

library(Factoshiny)

My_table <– read_excel("D:/test.xlsx")

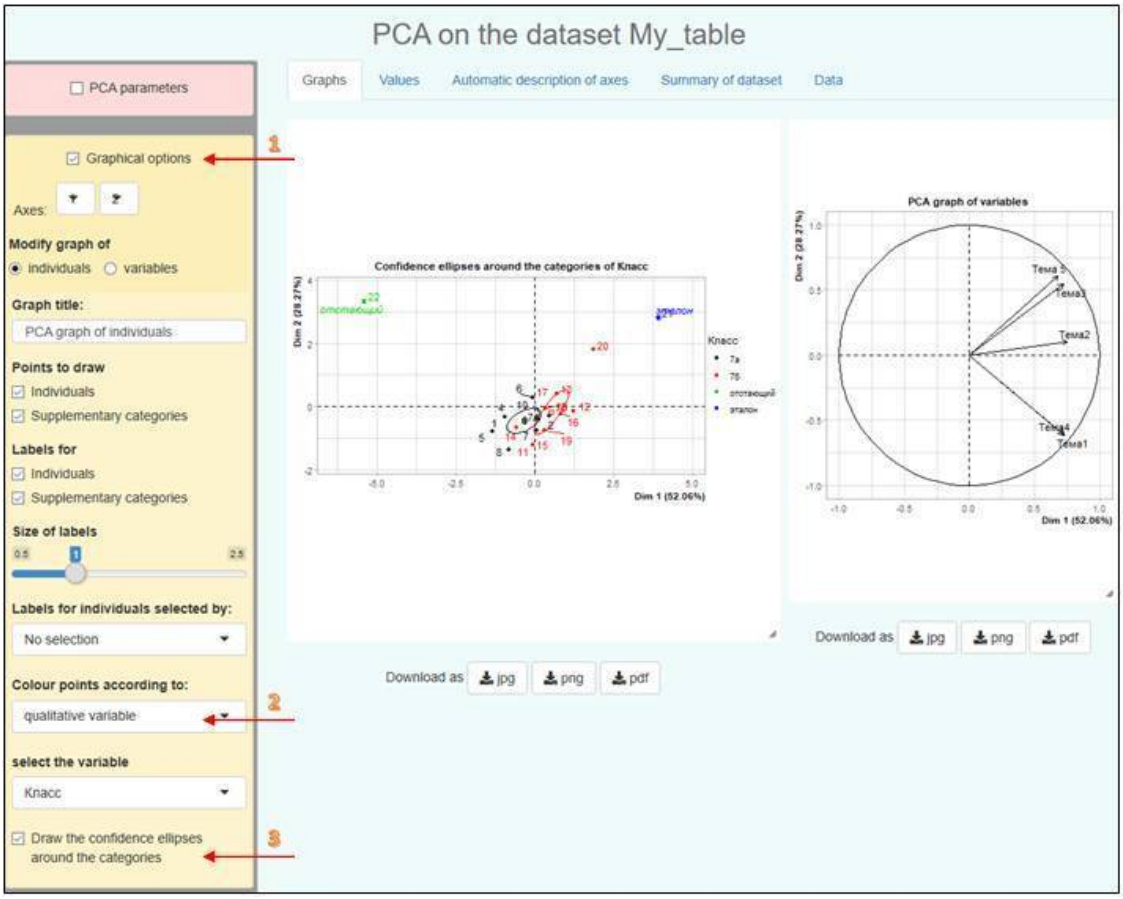
PCAshiny(My_table[,c(1, 3, 4, 5, 6, 7)])

Система запишет лог выполнения:

```
> library(readxl)
> library(Factoshiny)
Загрузка требуемого пакета: FactoMineR
Загрузка требуемого пакета: shiny
Загрузка требуемого пакета: FactoInvestigate
Загрузка требуемого пакета: ggplot2
> My_table <- read_excel("D:/test.xlsx")
> PCAshiny(My_table[,c(1, 3, 4, 5, 6, 7)])

Listening on http://127.0.0.1:4670
```

В открывшемся окне браузера настроим некоторые опции. Под номером 1 на рисунке отмечено включение дополнительных параметров построения графика; ПОД номером 2 настраивается способ выделения переменных цветом; под номером 3 включается изображение эллипсов доверительных интервалов значений переменных из разных категорий:




По полученному рисунку становится очевидным следующее:

– так как на круге корреляций вектора Тема1 и Тема4 фактически совпадают, то с этими темами большинство справились одинаково хорошо (если быть более точным, разделение по горизонтальной оси охватывает 52.06%, а по вертикальной – 28.27% тестируемых);

– эталонный ученик оказался в первой четверти, где лежат вектора Тема2, Тема3 и Тема5, значит остальным хуже дались перечисленные Тема2, Тема3 и Тема5;

– ученик №20 лучше всех освоил пройденный материал, так как ближе к эталонному отличнику, а с учениками 1, 4, 6, 8 следует позаниматься дополнительно;

– Тема2 в 7б была освоена лучше, чем в 7а, так как красный и черный эллипсы оказались разнесены вдоль направления вектора Тема2;

– так как центры обоих эллипсов лежат в нижней полуплоскости, снесены от начала координат по направлению векторов Тема1 и Тема4, следовательно статистическому большинству Тема3 и Тема5 далась хуже, чем Тема1 и Тема4, поэтому Темы 3 и 5 необходимо изучить детальнее.

Сказанное выше соотносится с исходными табличными данными, но на большом количестве факторов и аналитических данных графическое представление для обнаружения закономерностей оказывается гораздо удобнее.

# Глава 1. Первое знакомство

Внимательный читатель наверняка понял из введения, что эта книга поможет в сфере анализа педагогических данных с помощью R: научит, как импортировать данные в R, систематизировать их наиболее эффективным способом, преобразовать данные, визуализировать и смоделировать возможную динамику. Аналогично тому, как начинающий математик учится ставить мысленные эксперименты, формулировать гипотезы, рассуждать по аналогии, формировать доказательную базу, вы узнаете, как представлять данные, строить графики и многое другое. Эти навыки позволяют состояться онлайн-учителю как исследователю, и в этой книге собраны проверенные оптимальные способы работы с R, освоив которые будет легко использовать язык графиков, чтобы экономить время. Кроме того, станет ясным, как достичь понимания в процессе визуализации и исследования данных. Наука о данных – это захватывающая дисциплина, которая позволяет превратить необработанные исходные разрозненные данные в систематизированные, породив понимание и новое знание. Таким образом, основная цель этой книги – помочь читателю изучить наиболее важные инструменты в R, позволяющие заниматься наукой о педагогических данных. После прочтения этой книги у вас появятся инструменты для решения широкого круга задач средствами R.

# §1. Основы статистической обработки информацией

Наука о данных – это огромная сфера человеческой деятельности, общепринятый подход к освоению которой, прослеживающийся в каждом исследовательском проекте как правило следующий. Сначала данные импортируются в R. Обычно это означает, что берете данные, хранящиеся в файле, базе данных или интернете, и загружаете их в таблицу данных R. Если не можете импортировать свои данные в R, то дальнейший анализ данных в R не представляется возможным и стоит рассмотреть альтернативные варианты.

После того, как импортировали свои данные в R, неплохо было бы привести их в порядок. Очистка ваших данных означает хранение их в согласованном виде, который соответствует семантике набора данных. Короче говоря, когда данные структурированы, каждый столбец является переменной, и каждый ряд – это наблюдение. Структурированные отфильтрованные данные важны еще и потому, что последовательная запись позволяет сосредоточиться на вопросах о непосредственно самих данных, а не на вопросах о получении данные в правильном формате для разных функций.

После того, как у вас есть структурированные данные, общим первым шагом является их преобразование, включающее в себя:

1) фильтрацию по наблюдениям (например, все люди обучающиеся в одном городе, или все данные за последний учебный год);

2) создание новых переменных, которые являются функциями от существующих переменных (например, вычисление продолжительности обучения или длительности прохождения тестов);

3) вычисление набора сводных статистических данных (например, наивысший балл из набранных обучающимися).

После того, как у вас есть структурированные данные с вычисленными переменными запускаются два основных генератора новых знаний: визуализация и моделирование. Оба имеют свои сильные и слабые стороны, и любой реальный анализ будет происходить в процессе их многократного чередования.

Визуализация – это фундаментальная человеческая деятельность. Одна хорошая визуализация покажет вам то, чего даже не ожидали, или поднимет новые вопросы об анализируемых данных. Хорошая визуализация также может намекнуть, что задаете неправильный вопрос, или что нужно собирать дополнительные данные. Визуализация может вдохновить вас, но не стоит обольщаться, так как для интерпретации результатов всё же требуется участие человека.

Моделирование является дополнительным инструментам визуализации. После того, как достаточно точно сформулировали свои вопросы, можете попробовать использовать математическую модель, чтобы ответить на них. Модели в R принципиально являются математическими и представляют собой вычислительный инструментарий, поэтому они хорошо масштабируются. Нередко бывает дешевле купить больше компьютеров, чем это купить больше мозгов. Но каждая модель при этом генерирует лишь предположения, и по своей природе математическая модель не может подвергать сомнению свои собственные предположения. Это означает, что модель сама по себе не может сделать принципиальное открытие.

Последняя ступень анализа данных: представление полученных результатов, – самая критическая часть любого аналитического проекта. При этом не важно, насколько хороши ваши модели или визуализации, если не можете передать свои идеи и результаты другим людям.

Объединяет все названные этапы – программирование, оно красной нитью проходит через каждый этап проекта, но не нужно быть опытным программистом, чтобы анализировать данные, достаточно знания базовых концепций, и желания постоянно улучшать свои навыки

программирования, так как последнее позволяет автоматизировать частые задачи и проще решать новые.

Вы будете использовать названные инструменты практически в каждом проекте, но для большинства проектов их недостаточно. Есть эмпирический принцип 80/20 (закон Парето): можно решить около 80% задач каждого проекта используя методы, которыми уже владеете, но всегда понадобятся новые знания, чтобы справиться с оставшимися 20%.

Предыдущее описание инструментов обработки данных организована примерно в соответствии с той последовательностью, в которой они используются в статистическом анализе (хотя, конечно, любое правило имеет исключения). По собственному опыту, лучший порядок их освоения таков:

1) Начинать изучение с импорта данных, их очистки и систематизации является неоптимальным, так как 80% времени будет занято рутиной. Вместо этого, начнем с визуализации и преобразования данных, которые уже были импортированы и отфильтрованы. Таким образом, когда будете импортировать и приводить в порядок собственные данные, ваша мотивация останется высокой, потому что понимаете, к чему движетесь.

2) Некоторые темы лучше объяснить объединив их. Например, легче понять, как работают модели, если уже знаете о визуализации, структурированных данных, и программировании.

3) Инструменты программирования не самоцель, но все же позволяют взяться за решение значительно более сложных проблем. Поэтому весь спектр инструментов программирования будет представлен в середине книги, а затем увидите, как они могут сочетаться с другими статистическими инструментами, при решении интересных задач моделирования.

В рамках каждой главы постараемся придерживаться подхода из введения по аналогичному шаблону: начинать с некоторых мотивирующих примеров, чтобы увидеть картину в целом, а затем погрузиться в детали. Каждый раздел книги сопровождается упражнениями, позволяющими на практике закреплять пройденный материал. Хотя и бывает заманчивым пропускать их, но очень хорошо сказал Д. Пойа по этому поводу: «лучший способ научиться решать задачи состоит в том, чтобы самому решать эти задачи».

Есть несколько важных тем, которые в данной книге не будут охвачены, с той лишь целью, чтобы сосредоточиться на самом главном и как можно быстрее начать работать в R. А именно, не будут охвачены популярные ныне большие данные (так называемые «биг дата»). Сфокусируемся на небольших, располагаемых в памяти персонального компьютера наборах данных, что вполне оправдано для начала, ведь невозможно заниматься большими данными, если у вас нет опыта работы с малыми. При этом, сам инструмент освоите, и будете легко обрабатывать сотни мегабайт данных, и с тем же успехом сможете использовать полученные навыки для работы с 1-2 Гб данных. Для сравнения, базы рабочих учебных программ дисциплин подготавливаемых каждым преподавателем ежегодно занимают порядка 1 Гб. Если же регулярно работаете с большими данными (порядка 10-100 Гб и более), то должны будете узнать больше об иных инструментах для их обработки. Эта книга не учит работе с большими таблицами данных, появляющимися на международных образовательных платформах, из облачных хранилищ. Но если действительно работаете с большими данными, то для повышения производительности своего труда стоит приложить дополнительные усилия к освоению необходимых инструментов.

Если данные действительно большие, тщательно подумайте, может ли задача с большими данными быть решена на небольших наборах данных. Хотя исходные данные могут быть большими, часто данные, необходимые для ответа на конкретный вопрос, невелики. Возможно, найдется подмножество, подвыборка или сводка, которая помещается в память и все еще позволяет ответить на интересующий вопрос. Проблема здесь заключается в том, чтобы найти правильные небольшие данные, что часто требует много итераций. Альтернативный вариант

заключается в том, что задача с большими данными является совокупностью задач с малыми входными данными, а значит, решение легко поддается распараллеливанию. Например, каждая подзадача может поместиться в локальной памяти, но у вас их миллионы. В этом примере можно построить соответствующую модель для каждого наблюдения в наборе данных. Это было бы тривиально, если бы было всего 10 или 100 наблюдений, но вместо этого у вас их миллион. К счастью, порой анализ каждого наблюдения можно осуществлять независимо от других, тогда понадобится система (например, Hadoop или Spark), позволяющая отправлять различные наборы данных на разные компьютеры для параллельной обработки. После того, как нашли способ решения своей задачи для фиксированного подмножества входных данных с помощью описанных в этой книге инструментов, примените иные инструменты, для решения её на всём наборе данных.

Далее, из этой книги ничего не узнаете о Python, Julia или любом другом языке программирования, полезном при обработке данных. Это не потому, что эти инструменты плохие, отнюдь. На практике большинство команд аналитиков используют смешение языков, часто такое происходит с R и Python. Однако, лучше осваивать один инструмент за раз. Подобно ныряльщику за жемчугом, если сгруппироваться при входе, то движение к заветной цели в новой среде будет и глубже и быстрее. Это вовсе не значит, что следует изучить только одну тему, хотелось лишь напомнить, что как правило, учиться гораздо легче, если во время обучения методом погружения придерживаться одного направления. Следует так же стремиться узнавать новое на протяжении всей своей карьеры онлайн-учителя.

Поистине, R это отличная отправная точка для путешествия в мире науки о данных. Ведь R это не просто язык программирования, а интерактивная среда для совместной работы над анализом научных данных. Для поддержки взаимодействия пользователей, R является гораздо более гибким языком, чем многие из них его ровесники. Эта гибкость имеет своими недостатки, но большой плюс в том, как легко можно развивать адаптированные грамматики для конкретных частей процесса обработки данных. Эти вспомогательные мини-языки помогают думать о решаемых проблемах в привычной терминологии, поддерживая пластичное взаимодействие между вашим мозгом и компьютером.

Эта книга посвящена исключительно табличным данным: коллекции значений, каждая из которых связана с переменной и наблюдением. При том, что есть много наборов данных, которые не вписываются естественным образом в эту парадигму, например, изображения, звуки, деревья и текст. Но таблицы чрезвычайно распространены в науке и промышленности, они являются отличной стартовой площадкой для анализа данных.

Можно разделить анализ данных на следующие два этапа: генерация гипотез и подтверждение гипотезы (иногда называемый подтверждающим анализом). Основное внимание в этой книге уделяется генерации гипотез или исследованию данных. Будем внимательно смотреть на данные и в сочетании предметной областью генерировать много интересных гипотез, чтобы помочь найти объяснение, почему данные ведут себя именно так. Относитесь к гипотезам непредвзято, скептически, с разных сторон подходя критически.

Естественным продолжением процесса генерации гипотез является подтверждение одной из гипотез. Подтверждение гипотезы бывает трудным по двум причинам:

1) Для этого понадобится точная математическая модель, чтобы генерировать фальсифицируемые прогнозы. Это часто требует значительных усилий и статистической изощренности.

2) Наблюдение можно использовать только один раз для подтверждения гипотеза. Как только используете его больше, чем один раз, возвращайтесь к проведению исследовательского анализа. Это значит, чтобы считать гипотезу подтвержденной, нужно написать заранее весь план анализа, и не отклоняться от него. Позднее мы поговорим о некоторых стратегиях, которых стоит придерживаться, для упрощения моделирования.

Ошибочно полагать моделирование как инструмент подтверждения гипотезы и выбирать визуализацию в качестве единственного инструмента для генерации гипотез. Модели зачастую используется для формулирования гипотез, и с определенной осторожностью можно использовать визуализацию для их подтверждения. Ключевое различие заключается в том, насколько часто воспроизводится каждое наблюдение: если только один раз, это подтверждение; если больше, чем один раз, это уже исследование.

Чтобы получить максимальную отдачу от этой книги, предварительно стоит изучить вводный курс численных методов и определенно полезным иметь некоторый опыт в программировании. Если никогда не программировали ранее, до прочитайте базовый курс по программированию, как полезное дополнение к этой книге.

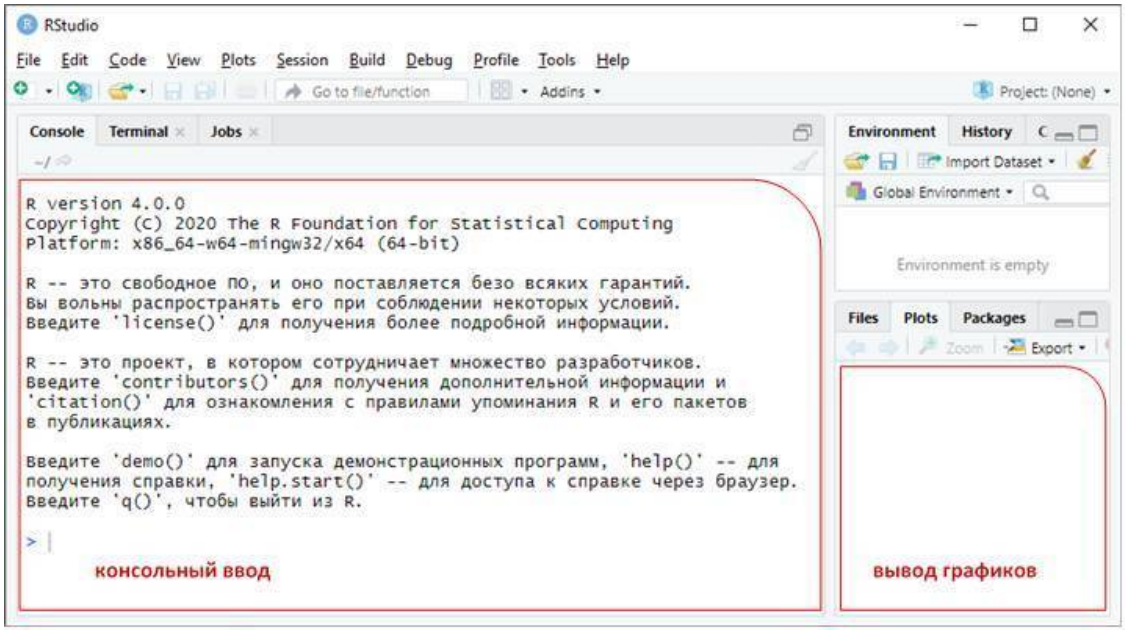
Для запуска примеров кода из этой книги вам понадобятся: R, RStudio, коллекция пакетов R под названием Tidyverse и несколько других пакетов. Пакеты являются основной единицей воспроизводимого кода R. Они включают в себя функции, документацию, и тестовые наборы данных.

Чтобы загрузить R, если не сделали этого ранее, перейдите на сайт CRAN, который состоит из зеркалируемых серверов, распределенных по всему миру, и используется для распространения R. Не пытайтесь выбрать зеркало, которое находится географически рядом, вместо этого используйте зеркало облака, https://cloud.r-project.org, которое автоматически выяснит оптимальное расположение серверов для загрузки.

Новая основная версия R выходит один раз в год, кроме того, выпускается два или три незначительных обновления каждый год. Положите за правило регулярно обновлять установку. Да, обновление может принести немного хлопот, особенно основных версий, которые требуют переустановки всех ваших пакетов, но откладывание только ухудшает ситуацию.

RStudio это интегрированная среда разработки, или IDE, для программирования в R. Загрузите и установите его с официального сайта http://www.rstudio.com/download. RStudio обновляется несколько раз в год. Когда новая версия будет доступна, RStudio сообщит об этом. Опять же, хорошей привычкой является регулярное обновление установки, чтобы воспользоваться преимуществами актуальных сборок. Для воспроизведения примеров из этой книги, убедитесь, что установлена актуальная версия RStudio.

Когда запустите RStudio, увидите поля в интерфейсе:



На данный момент все, что нужно знать, это то, что набирается R код на панели консоли, а запускается нажатием на клавиатуре клавиши Enter.

Хрестоматийный пример, после выполнения серии команд:

library(ggplot2)

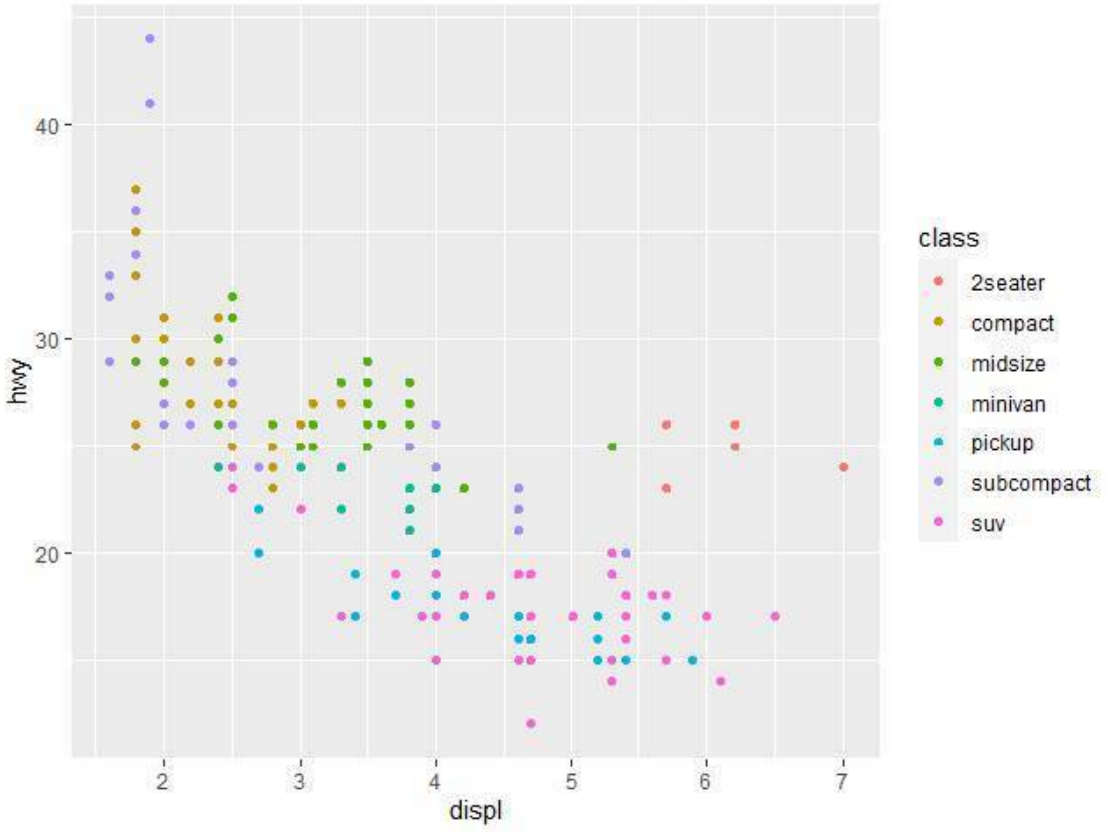
ggplot(mpg, aes(x = displ, y = hwy))+geom_point(aes(colour = class))

Система запишет лог выполнения

```
> library(ggplot2)
> ggplot(mpg, aes(x = displ, y = hwy))+geom_point(aes(colour = class))
```

подключится библиотека для работы с графиками (ggplot2) и визуализируются данные об объеме двигателя автомобиля, в литрах (displ) в зависимости от топливной экономичности автомобиля на шоссе, в милях на галлон (hwy) из предустановленной базы автомобильных производителей (mpg) выделив цветом автомобили разных классов (class).

В соответствующей главе разберем аналогичный пример на педагогических данных подробнее.




Также потребуется установить некоторые дополнительные пакеты R. Пакет R представляет собой набор функций, данных и документацию, что расширяет возможности базовой установки. Использование пакетов является ключевым для успешного использования R. большинство пакетов, которые используются в этой книге есть часть так называемого коллекции пакетов Tidyverse. Пакеты Tidyverse разделяют общую философию программирования и данных в R, а также предназначены для совместной работы. Вы можете установить полный комплект Tidyverse одной строкой кода:

install.packages("tidyverse")

На компьютере введите эту строку в поле консоли, а затем нажмите клавишу Enter, чтобы запустить её. R загрузит приложение пакеты с серверов CRAN и установит их на компьютер. Если возникнут проблемы при установке, убедитесь, что есть подключение к интернет, и что https://cloud.r-project.org/ не блокируется с помощью брандмауэра или на стороне прокси-сервера.

Вы не сможете использовать функции, данные и файлы справки из пакета, пока не подключите его с помощью команды library(). После установки пакета можете загрузить его библиотечной функцией:

library(tidyverse)

В логе консоли отобразится процесс загрузки необходимых пакетов, являющихся ядром tidyverse, потому что используются они практически в каждом анализе.

Пакеты tidyverse достаточно часто обновляются. Можете посмотреть, доступны ли обновления, и при необходимости установить их, запустив

tidyverse_update()

Есть много разных пакетов, которые не являются частью tidyverse, они либо решают задачи статистического анализа немного иначе, либо предназначены для использования других парадигм представления информации. Это не делает их лучше или хуже, они просто иные. По мере ознакомления с R, обязательно узнаете про новые пакеты и новые способы представления данных. В книге будем использовать как правило три вспомогательных пакета из-за ограничений tidyverse. Так как в них предоставлены данные о мировом развитии, рейсах авиакомпаний и бейсболе, то присутствуют некоторые сведения об успеваемости, развитии, различные оценки и тесты, поэтому мы будем использовать их для иллюстрации ключевых идей науки о педагогических данных.

Выше было показано несколько примеров выполнения кода на языке R. Код в книге выглядит так:

```
1 + 2
#[1] 3
```

Если запустите тот же код в своей локальной консоли, он будет выглядеть так:

```
> 1 + 2
[1] 3
```

Как видим, в консоли код вводится после приглашения, символа «>», его не будем дублировать в книге. Выходные же данные порой закомментированы с помощью «#»; а в консоли они появляется непосредственно после кода. Эти два различия означают, что если работаете с электронной версией книги, то сможете легко копировать код из книги и вставлять его в консоль. Кроме того, на протяжении всей книги будем пользоваться следующими договоренностями в тексте кода:

1) Функции находятся в шрифте кода и завершаются скобками, например sum().

2) Другие объекты R, например, данные или аргументы функции, записываются без скобок.

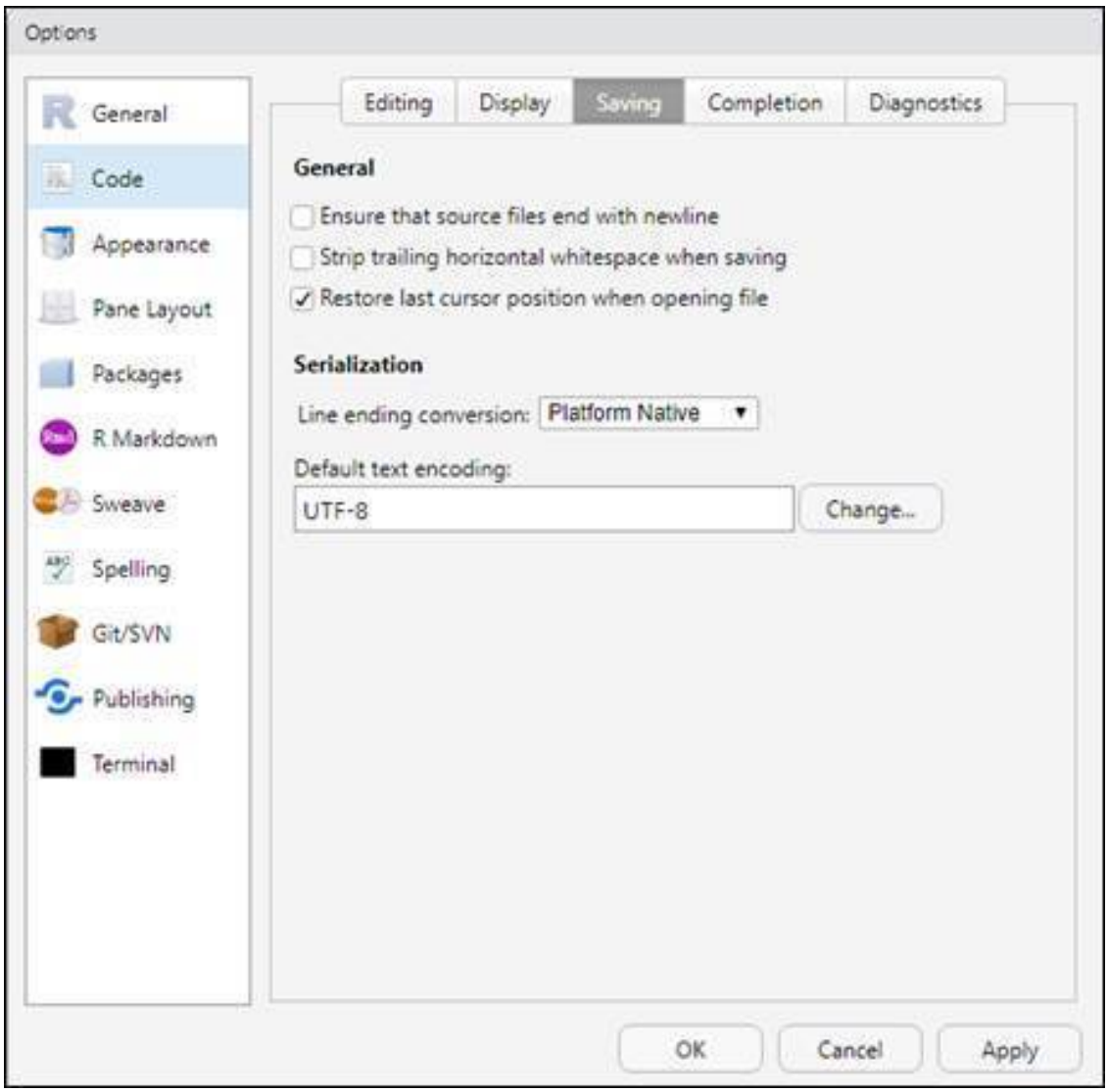
3) Если из контекста не ясно, какой объект из какого пакета, то будем использовать имя пакета, за которым следуют два двоеточия и имя объекта. Этими допущениями изобилует код R.

С другой стороны, эта книга не единственна в своём роде, есть много людей и онлайн-ресурсов, помогающих освоить R. Как только начнете применять техники, описанные в этой книге к вашим данным, наверняка возникнут вопросы, на которые здесь нет ответа. Приведём несколько советов, как получить помощь, чтобы продолжить обучение. Если не знаете, с чего начать, начните с Yandex. Как правило, добавления «R» к поисковому запросу достаточно, чтобы повысить релевантность поисковой выдачи: если поиск не удался, это часто означает, что нет доступных для R результатов, но Yandex особенно полезен для поиска сообщений об

ошибках. Если же получаете сообщение об ошибке и Yandex понятия не имеет, что это значит, попробуйте погуглить. Скорее всего, кто-то ранее уже сталкивался с подобным, и обращался за помощью где-нибудь в интернете. Если сообщение об ошибке не отображается на русском, то в консоли введите

Sys.setenv(LANGUAGE = "ru")

и повторите запуск кода; с большей вероятностью найдете справку для сообщения об ошибке на русском языке. При этом, удастся избежать многих проблем совместимости кода, написанного членами международных исследовательских команд, если настроите RStudio через меню Tools/Global Options на использование UTF-8 в качестве кодировки по умолчанию, как это показано на следующей иллюстрации:




Если Yandex не помогает, то попробуйте ru.stackoverflow.com. Начните с того, чтобы потратить немного времени на поиск существующего ответа, в том числе по тэгу [R], чтобы ограничить ваш поиск вопросами и ответами, которые используют R. Если не нашли ничего полезного, то подготовьте краткий воспроизводящий ошибку пример. Удачный пример делает его более доступным для других людей, чтобы они смогли помочь вам, и часто искореняет проблему в процессе его подготовки.

Есть три вещи, которые нужно описать, чтобы сделать ваш пример воспроизводим другими: необходимые пакеты, используемые данные и код.

1) Пакеты должны загружаться в самом начале скрипта, чтобы можно было увидеть, какие из них понадобятся в примере. При этом, хорошо не лишним будет проверить, что используете последнюю версию каждого пакета; возможно, разработчики обнаружили ошибку ранее и она уже была исправлена с момента установки пакета. Для пакетов из комплекта tidyverse самый простой способ – запустить tidyverse_update().

2) Самый простой способ включить свои данные в вопрос, это использовать dput () при создании кода R. Например, чтобы воссоздать набор данных mtcars в R, достаточно выполнить следующее шаги:

а) Запустите dput (mtcars) в R

б) Скопируйте выходные данные

в) В демонстрационном скрипте введите mtcars<– и вставьте ранее скопированное. Попробуйте найти наименьшее подмножество ваших данных, которое приводит к появлению проблемы.

3) Потратьте немного времени на то, чтобы проверить, что ваш код легко читается другими участниками. Для этого: убедитесь, что использовали пространство и ваш имена переменных лаконичны, но информативны; используйте комментарии, чтобы указать, где именно возникла проблема; сделайте все возможное, чтобы удалить все, что не связано с демонстрируемой проблемой. Чем короче ваш код, тем проще его использовать, понять, и тем легче его исправить.

4) Закончите, проверив, что действительно сделали воспроизводимый пример путем запуска нового сеанса R, копирования и вставки своего скрипта внутрь.

В любом случае, необходимо потратить некоторое время на подготовку, чтобы можно было решать проблемы до их возникновения. Инвестирование времени в обучение R каждый день на долгосрочной перспективе окупится сторицей. Принимайте активное участие в обсуждениях перспективных проектов на блоге RStudio, там размещаются объявления о новых пакетах, публикуются новые возможности интегрированной среды разработки и разрабатываются индивидуальные курсы. Чтобы идти в ногу со временем и сообществом R в более широком смысле, рекомендуется чтение http://www.r-bloggers.com: оно объединяет более 700 блогов пользователей R со всего мира. Если являетесь активным участником социальной сети Twitter, то подпишитесь на обновления по хэштегу #rstats, так как Twitter является одним из ключевых инструментов, который используют разработчики на R. Эта книга не просто вольный пересказ новостей компании занимающейся активной разработкой R, а результат долгой и плодотворной самостоятельной работы. С публикациями автора, в которых оказались применены описываемые инструменты статистической обработки информации в педагогических, биологических и химических областях на базе научных и исследовательских лабораторий ОмГПУ, можно ознакомиться на сайте https://www.researchgate.net.

# §2. Визуализация и преобразование данных

Цель первой главы состояла в том, чтобы получить быстрое знакомство с основными инструментами исследования данных. Ведь исследование данных это в первую очередь искусство просмотра ваших данных, быстрая генерация гипотез, быстрая их проверка, а затем повторение этого процесса снова, и снова. Цель предварительного исследования данных заключается в том, чтобы сгенерировать как можно больше многообещающих идей, которые можете будет развить позднее.

Во второй части книги изложены некоторые полезные инструменты, которые дают немедленную отдачу по следующим причинам:

1) Визуализация прекрасна для начала работы в R, потому что выигрыш очевиден, научитесь делать элегантные и информативные графики, которые помогут понять собранные данные. Погрузитесь в визуализацию изучая основное содержимое библиотеки ggplot2, и узнаете мощные методы превращения табличных данных в графики.

2) Визуализация сама по себе, как правило, не является достаточной для полноценного исследования, потому что в последующей трансформации данных ключевое место занимают визуально обнаруживаемые тренды, наглядная фильтрация наблюдений, создание новых переменных и вычисление сводных данных.

3) Наконец, в исследовательском анализе данных, приходится сочетать визуализацию и преобразования с вашим любопытством и скептицизмом, чтобы задать и ответить на интересующие вопросы о данных.

Моделирование является важной частью исследовательской работы, но порой не хватает навыков, чтобы эффективно этому обучиться для многократного применения. Вернемся к моделированию, как только освоим большое количество инструментов для обработки и программирования данных.

Среди последующих глав, сконцентрированных на изложении инструментов исследования, присутствует описание рабочих процессов. В соответствующем разделе разбираются основы рабочего процесса, автоматизация сценариями, на примере готовых решений иллюстрируются ведущие практики написания и организации R-кода. Это настроит на успех в долгосрочной перспективе, так как даёт инструменты для реализации конкретных проектов.

Как было показано во введении, простой график приносит больше информация для ума аналитика, чем любое другое представление данных. Покажем, как визуализировать данные с помощью ggplot2. В R имеется несколько систем для построения графиков, но ggplot2 является одним из самых элегантных и самых универсальных, так как ggplot2 реализует графический язык, схожий в системе описания и построения графиков. С ggplot2, многое делается быстрее, изучив одну систему команд можно применять её в самых неожиданных местах.

Если хотите узнать больше о теоретической основе ggplot2, то прежде, чем продолжить, рекомендуется прочитать специализированную учебную литературу по компьютерной графике. А в данной главе сфокусируемся на ggplot2, как одном из основных членов библиотеки tidyverse. Для доступа к наборам данных, справке и функциям, которые мы будем использовать в этой главе, загрузите tidyverse запустив следующую строку кода на исполнение:

```
library(tidyverse)
```

Эта одна строка кода загружает ядро tidyverse, пакеты, которые будут использоваться практически при каждом анализе данных. После её выполнения в консоли показывается, какие функции из tidyverse конфликтуют с функциями в базе R (или из других пакетов, которые могли быть загружены). Если запустите этот код и получите сообщение «Ошибка в library(tidyverse) :нет пакета под названием ‘tidyverse’», то нужно будет сначала установить его, а затем снова запустить library() следующим образом:

```
install.packages("tidyverse")
library(tidyverse)
```

Достаточно однократно установить пакет, но необходимо подгружать его каждый раз, когда открываете новую рабочую сессию. Если потребуется в явном виде указать из какого пакета вызывается функция (или набор данных), то будем использовать специальную нотацию с двойным двоеточием, например, ggplot2::ggplot() сообщает явным образом, что мы используем функцию ggplot() из пакет ggplot2. Давайте разберем первый график из предыдущей главы, чтобы ответить на а вопрос: используют ли автомобили с большими двигателями больше топлива, чем автомобили с маленькими двигатели? Аналогично риторическому: лучше ли осваивают математику ученики в специализированных физико-математических классах, чем ученики обучающиеся в классах с минимальным количеством уроков математики? Вы, вероятно, уже знаете ответ, но попробуйте конкретизировать. Какова взаимосвязь между размером двигателя и топливная эффективность, либо взаимосвязь между количеством учебного времени, выделяемого на элементарную математику, и успехами страны в космической отрасли, как она выглядит: положительно? отрицательно? линейно? нелинейно?

Вы можете проверить свой ответ с помощью базы данных mpg хранящейся в ggplot2 (она же ggplot2::mpg). База данных представляет собой таблицу переменных (в столбцах) и наблюдаемых значений (в строках). База mpg содержит наблюдения, собранные американскими агентством по охране экологии на 38 моделях автомобилей.

Среди прочих переменных в базе mpg хранятся:

1. displ, – объем двигателя автомобиля, в литрах;
2. hwy, – топливная экономичность автомобиля на шоссе, в милях на галлон (mpg).

Автомобиль с низкой топливной экономичностью потребляет больше топлива, чем автомобиль с высокой топливной эффективностью, когда они проедут одно и то же расстояние. Чтобы узнать больше о содержимом mpg, откройте ее страницу в справке.

Чтобы визуализировать mpg, запускается следующий код, который отобразит displ на ось *x* и hwy на ось *y*:

```
ggplot (data = mpg) +
geom_point (mapping = aes (x = displ, y = hwy))
```

По графику становится очевидной отрицательная связь между размером двигателя (displ) и топливной экономичностью (hwy). Другими словами, автомобили с большими двигателями использует больше топлива, равно как и большее количество учебного времени приводит к заведомо лучшим результатам обучающихся. Может ли это подтвердить или опровергнуть гипотезу о топливной экономичности и размере двигателя?

С помощью ggplot2 можно начать построение графика с помощью функции ggplot(). ggplot() создает систему координат, которую можно наполнить слоями. Первый аргумент функции ggplot() это набор данных, используемый в диаграмме. Таким образом, ggplot(data=mpg) создает пустой график, но это не очень информативно. Вы завершите построение графика, добавив один или несколько слоёв в ggplot(). Функция geom_point() добавляет слой точек.

В общем случае, ggplot2 используется со многими функциями категории geom, каждая из которых добавляет отдельный слой к графику, на протяжении этой главы они еще будут упомянуты.

Каждая функция geom в ggplot2 принимает аргумент, который определяет, как переменные в наборе данных сопоставляются с визуальными свойствами. Аргумент mapping всегда сопряжен с функцией aes(), а аргументы x и y в функции aes () определяют, какие переменные нужно сопоставить осям x и y соответственно. ggplot2 ищет сопоставленные переменные в данных аргумента, например, в таблице mpg.

Перепишем код в виде многоразового шаблона для построения графиков с помощью ggplot2. Чтобы построить график, достаточно заменить заключенные в угловые скобки фрагменты в коде ниже на наборы данных, функцию geom, либо на соответствия по осям:

ggplot(data = <данные>) +

<функция geom>(mapping = aes(<функция geom ><соответствие>))

В дальнейшем разберем детально, как заполнить и расширить этот шаблон, чтобы построить графики различных типов. Начнем с компоненты <соответствие>.

**Упражнения**

1. Запустите ggplot(data=mpg). Что получится?

2. Сколько строк находится в mpg? Сколько там колонок?

3. Что описывает переменная drv? Прочитайте справку по mpg, введя в консоли ?mpg, чтобы выяснить это.

4. Сделайте диаграмму рассеяния hwy относительно cyl.

5. Что произойдет, если попытаться создать диаграмму рассеяния переменной class относительно drv? Почему такого графика нельзя построить?

С эстетической точки зрения, картина тем ценнее, чем больше она заставляет нас замечать то, чего мы никогда не ожидали увидеть. На анализируемом графике обнаруживается одна группа точек (выделенная оранжевым цветом для автомобилей класса 2-Seater, «двуместные») в которой кажется, что они выходят за пределы линейного тренда. Эти автомобили имеют более высокий пробег, чем можно было бы ожидать. Как объяснить подобное? Давайте предположим, что данные автомобили являются гибридами. Один из способов проверить эту гипотезу – посмотреть на значение переменной class для каждого автомобиля. Переменная class из набора данных mpg разделяет автомобили на такие группы, как compact, midsize и SUV. Если выделяющиеся точки являются гибридными автомобилями, их следует классифицировать как компактные автомобили или, возможно, малолитражные (имейте в виду, что эти исторические данные были собраны раньше, чем стали популярными гибридные грузовики и внедорожники).

Можно добавить третью переменную, например class, к двумерному графику исходя из эстетических соображения. Эстетика – это визуальное свойство объектов в вашей диаграмме. Эстетика включает в себя такие вещи, как размер, форма или цвет точек. Изобразим точки различными способами путем изменения перечисленных значений свойств их оформления. Так как мы уже используем это слово "значение" для описания числовых данных, то будем использовать слово "уровень" для описания художественных, нечисловых, эстетических свойств. Мы меняем уровни размера, формы и цвета точки, чтобы сделать точка маленькая, треугольная или синяя.

В результате, появляется возможность передать информацию о данных с помощью сопоставление эстетики на графике с переменными в наборе данных. Например, можно сопоставить цвета точек (color) с переменной класса автомобиля (class), чтобы выявить класс каждого автомобиля. Для этого, после x = displ, y = hwy в список аргументов функции aes() через запятую необходимо добавить color = class.

Чтобы отобразить настройки форматирования в переменную, сопоставляется имя настраиваемого параметра, например цвета (color), с именем переменной внутри aes(). ggplot2 автоматически присвоит уникальный цвет для каждого уникального значения переменной, а также добавит объяснение, какие уровни каким значениям соответствуют.

Цветом показано, что многие из необычных точек охватывают двухместные автомобили. Эти автомобили не похожи на гибриды, и выглядят, по сути, как спортивные автомобили. Спортивные автомобили имеют большие двигатели, такие как внедорожники или пикапы, но небольшие кузова, такие как средние и компактные автомобили, что улучшает их экономичность. В ретроспективе, эти автомобили вряд будут гибридами, так как у них есть большие двигатели.

В приведенном выше примере сопоставлен класс с цветом, но можно сопоставить класс с размером точки точно так же. В этом случае размер каждой точки будет демонстрировать классовую принадлежность. Достаточно лишь заменить color = class на size = class, но будет получено предупреждение от интерпретатора, так как сопоставление неупорядоченной переменной (class) с упорядоченной категорией размера (size) не самая лучшая идея.

#> Предупреждение: использование параметра size для дискретной переменной не рекомендуется.

Кроме того, можно сопоставить класс с уровнем прозрачности точек (alpha), либо с их формой (shape). Для этого достаточно заменить color = class на alpha = class, либо на shape = class соответственно. Но в последнем случае ggplot2 может использовать только до шести фигур одновременно, по умолчанию все остальные группы будут отключены.

Для каждой эстетики используется aes(), чтобы связать имя эстетического объекта с переменной для отображения. Функция aes() собирает вместе каждое из эстетических отображений, используемых слоем и передает их в аргумент отображения слоя. Синтаксис выделяет полезную информацию об осях *x* и *y*: расположение объектов *x* и *y*, точки сами по себе являются эстетикой, визуальными свойствами, которые можно отобразить к переменным для демонстрации данных. После того, как настроена эстетика, ggplot2 заботится обо всём остальном. Выбирается оптимальный масштаб для использования, строится легенда, которая объясняет условные обозначения. Для координатных осей *x* и *y* функция ggplot2 не создает легенду, но будут построены осевые линии с делениями и метками. Линия оси сама по себе выступает в качестве легенды, так как она объясняет связь между расположением и координатами точек.

По аналогии можно задавать свойства объекта geom вручную, например, можно сделать все точки на диаграмме зелеными, если использовать следующий синтаксис:

geom_point(mapping = aes(x = displ, y = hwy), color = "green")

Здесь цвет не передает информацию о переменном, он только меняет внешний вид графика. Устанавливая параметры вручную, можно регулировать общий стиль диаграмм. В частности, форму точек можно задавать порядковыми номерами, например, 0, 15 и 22 – это квадраты, разница между ними заключается в том, что некоторые залиты сплошным цветом. Полые формы (0-14) имеют границу, определяемую значением параметра color; сплошные формы (15-18) заполнены цветом указанным в color; а заполненные формы (21-24) имеют границу, совпадающую с цветом заливки.

**Упражнения**

1. Как сделать цвет всех точек графика синим?

2. Какие переменные в базе mpg являются категориальными? Который переменные являются непрерывными? (Подсказка: найдите в документации описание типов полей таблицы mpg). Где найти эту информацию при открытии справки по mpg?

3. Сопоставьте непрерывную переменную с цветом, размером и формой. Как такие настройки эстетики поведут себя для категориальных в отличии от непрерывных переменных?

4. Что произойдет, если сопоставить одну и ту же переменную с несколькими эстетиками?

5. Что делает эстетика stroke? В каких случаях она применима? (Подсказка: в документации найдите описание функции geom_point, для этого в консоли можно ввести ?geom_point)

6. Что произойдет, если сопоставите эстетику с чем-то другим, не являющимся именем переменной, например color = displ < 5? Как и прежде предварительно нужно будет указать значения параметров *x* и *y*.

Когда начнете выполнять код R, возможны некоторые затруднения. Не волнуйтесь, это случается со всеми. Автор книги тоже писал код R, в течение многих лет, который работает не сразу. Начните с тщательного сравнения кода, который используете, с кодом из книги. R чрезвычайно придирчив. Убедитесь, что каждая открывающаяся скобка «(«соответствует закры-

вающейся «)», а каждая кавычка «"» имеет парную «"». Иногда запускаете код, но ничего не происходит. Проверьте содержимое левого нижнего угла консоли: если там «+», это означает, что R ничего не делает, просто набрали не полное выражение, и он ждет завершения ввода. В этом случае можно начать с начала, нажав клавишу ESC, чтобы прервать обработку текущей команды.

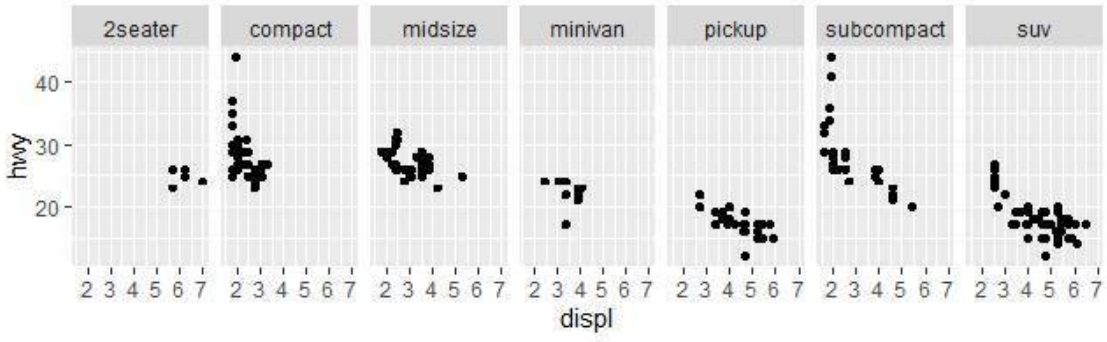
Частая ошибка при создании графиков ggplot2 это расположение «+» в неправильном месте, он должен находиться в конце строки, а не в начале. Убедитесь, что не сделали этого случайно. Если всё ещё в тупике, попробуйте обратиться к справочной информации. Для получения развернутой справки о любой функции R достаточно ввести команду ?имя_функции в консоли, или выделить имя интересующей функции и нажать клавишу F1 в RStudio. Не волнуйтесь, если справки не кажется, бывает полезным вместо этого перейти к примерам и найти код, который соответствует тому, что пытаетесь сделать. Если и это не помогает, то внимательно перечитайте сообщение об ошибке. Иногда ответ содержится именно там. Просто для начинающих пользователей R, ответ может находиться в сообщении об ошибке, но не приходит его понимания. Еще одним отличным инструментом является Yandex, попробуйте поискать сообщение об ошибке в интернете, так как возможно, у кого-то была аналогичная проблема, и её решение описали на специализированных онлайн-форумах.

Как было показано выше, один из способов добавить дополнительные измерения на графике, это художественные вариации эстетических параметров. Но есть ещё один способ, особенно полезный для категориальных переменные, – это разбивка графика на фрагменты, подзадачи, каждая из которых заключается в отображении некоторого подмножества анализируемых данных.

Чтобы собрать свой график из нескольких фрагментов от одной одной переменной, используйте facet_wrap(). Первым аргументом функции facet_wrap() должна быть именем структуры данных в R, которое начинается с символа «~», за которым следует имя переменной. Переменная, передаваемая в функцию facet_wrap(), должна быть дискретной. Например, следующая команда:

```
ggplot(data = mpg) +
geom_point(mapping = aes(x = displ, y = hwy)) +
facet_wrap(~ class, nrow = 1)
```

разобьет общее изображение данных известной уже нам базы на фрагментарные части, расположив их одной строкой, так как указано nrow = 1:




Чтобы расположить группы данных фрагментами на сетке, можно использовать комбинацию из двух переменных, добавив функцию facet_grid() к вызову графопостроителя. Первый аргумент в этом случае на этот раз будет содержать два имени переменных, разделенных знаком «~»:

```
ggplot(data = mpg) +
```

geom_point(mapping = aes(x = displ, y = hwy)) +
facet_grid(drv ~ cyl)

Если необходимо расположить группы только по одной переменной, в одну строку, либо в столбец, то достаточно поставить «.» вместо имени второй переменной. Так, например окончание +facet_grid(cyl ~ .) предпишет расположить графики вертикально, сгруппировав автомобили по числу цилиндров.

**Упражнения**

1. Что произойдет, если будете группировать данные по непрерывной переменной?

2. Что означают пустые ячейки на графике с facet_grid(drv~cyl)?

3. Каковы преимущества использования группирования данных частями по сравнению с цветовым выделением точек на одном графике? Каковы их недостатки? Как соблюдать баланс достоинств и недостатков этих подходов на больших объемах данных?

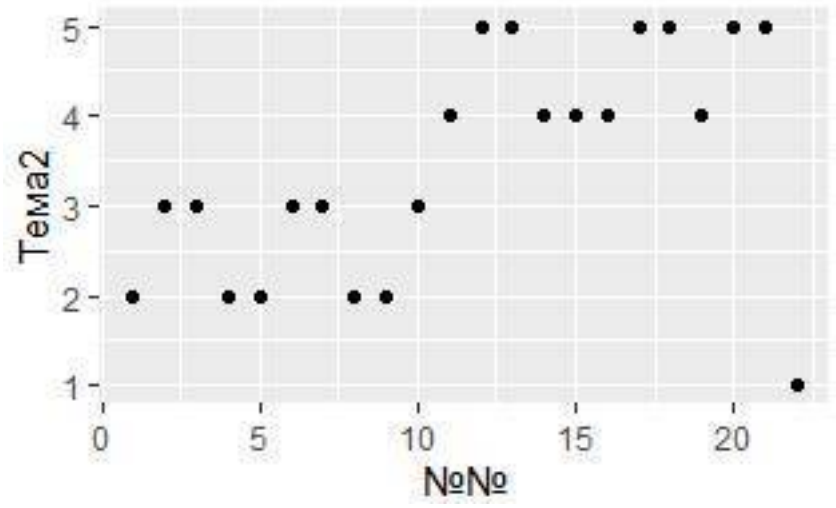
4. Прочитайте справку по ?facet_wrap. Что регулируется параметрами nrow, ncol? Какие ещё параметры управляют компоновкой? Почему функция facet_grid() не имеет аргументов nrow и ncol?

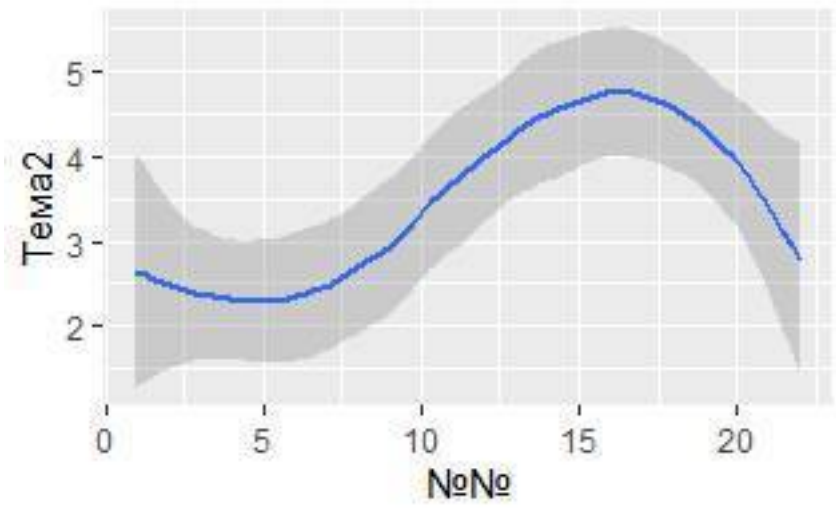
5. При использовании facet_grid() рекомендуется в столбцах располагать переменные с большим количеством уникальных значений. Почему?

Вернёмся к данным о результатах обучения. Во введении использовалась таблица с оценками успеваемости, которую можно воспроизвести следующей командой консоли:

My_table <– structure(list(Класс = c("7а", "7а", "7а", "7а", "7а", "7а", "7а", "7а", "7а", "7а", "7б", "7б", "7б", "7б", "7б", "7б", "7б", "7б", "7б", "7б", "эталон", "отстающий"), `Фимилия Имя` = c("Иванов Иван", "Петров Петр", "Сидоров Сидор", "Егоров Егор", "Романов Роман", "Николаев Николай", "Григорьев Григогий", "Викторов Виктор", "Михайлов Михаил", "Тимуриев Тимур", "Ульянова Ульяна", "Ольгина Ольга", "Людмилова Людмила", "Дарьева Дарья", "Кристинина Кристина", "Натальина Наталья", "Глафирова Глафира", "Янина Яна", "Иринова Ирина", "Валентинова Валентина", "Идеальный ученик", "Другая крайность"), Тема1 = c(5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 1, 1), Тема2 = c(2, 3, 3, 2, 2, 3, 3, 2, 2, 3, 4, 5, 5, 4, 4, 4, 5, 5, 4, 5, 5, 1), Тема3 = c(1, 2, 2, 1, 1, 2, 2, 1, 2, 2, 1, 2, 2, 1, 1, 2, 2, 2, 1, 2, 5, 1), Тема4 = c(4, 5, 5, 4, 4, 4, 5, 5, 5, 4, 5, 5, 4, 4, 5, 5, 4, 4, 5, 4, 5, 1), `Тема 5` = c(1, 2, 2, 2, 1, 2, 1, 1, 2, 2, 1, 2, 2, 1, 1, 2, 2, 1, 2, 5, 5, 1), `№№` = c(1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22)), row.names = c(NA, -22L), class = c("tbl_df", "tbl", "data.frame"))

Представим успеваемость графически:






Насколько похожи эти две иллюстрации?

Оба графика содержат одну и ту же переменную *x*, один и тот же *y*, оба визуализируют одни и те же данные. Но их сюжет не идентичен. Каждая иллюстрация описается на свои визуальные образы для представления данных. В синтаксисе ggplot2 они используют разные геометрические объекты (geom). Geom – это геометрический объект, который применяет графопостроитель для представления данных. Например, линейные диаграммы используют линейные геометрические объекты, прямоугольные диаграммы используют геометрические объекты прямоугольной формы и так далее. Диаграммы рассеяния нарушают этот тренд, они используют точечное представление данных. Как видели выше, можно использовать разные геометрические объекты для визуализации одних и тех же данных. На левом графике используется точечная геометрия, а в правом – гладкая линия, усредняющая данные. Чтобы изменить геометрические примитивы на вашем чертеже, измените функцию geom_, которую добавляете к ggplot (). Например, чтобы воспроизвести вышеприведенные рисунки, выполните код:

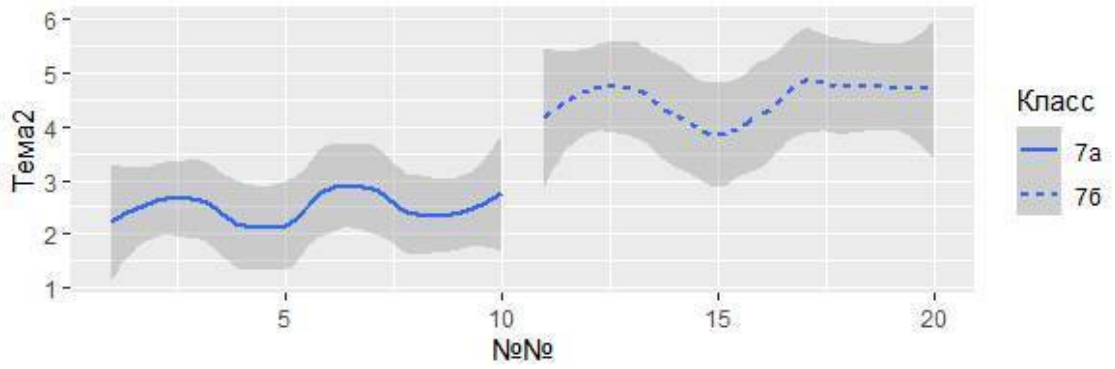
```
# левый график
ggplot (data = My_table) +
geom_point (mapping = aes (x = `№№`, y = Тема2))
# правый график
ggplot (data = My_table) +
geom_smooth (mapping = aes (x = `№№`, y = Тема2))
```

Каждая функция geom в ggplot2 принимает аргумент mapping, однако не каждая настройка эстетики работает с любой функцией geom. Можно было бы установить форму точки, но нельзя установить форму линии. С другой стороны, можно установить параметр linetype, тогда geom_smooth() нарисует линии разного типа для каждого уникального значения переменной, которая сопоставлена с типом линии.

Например, функция geom_smooth() может разделить обучающихся по классам:

```
ggplot (data = My_table) +
geom_smooth (mapping = aes (x = `№№`, y = Тема2, linetype = Класс))
```

Одна линия описывает успехи в освоении Темы2 для всех одноклассников из «7а», а другая из «7б»:




Покажется немного странным, эклектикой, но можно выполнить наложение всех линий поверх необработанных данных с последующим их окрашиванием в соответствии с успеваемостью класса. Заметим, что этот график потребует два вызова geom_ для построения, но как разместить несколько геометрических объектов разного типа на одном и том же графике.

ggplot2 обеспечивает более 40 вариантов функции geom_, пакеты расширений предоставляют ещё больше возможностей. Лучший способ получить исчерпывающий обзор, используйте справку: ?geom_smooth.

Многие варианты функции geom_, такие как geom_smooth(), например, используют один геометрический объект для отображения нескольких строк данных. Для этих функций, можно выносить эстетику группы в категориальную переменную для рисования нескольких объектов в едином стиле, так как ggplot2 нарисует отдельный объект для каждого уникального объекта значение группирующей переменной. На практике ggplot2 будет автоматическая группировка данных для этих функций всякий раз, когда сопоставляется эстетика для дискретной переменной (как было в примере с linetype). Удобно использовать эту особенность, потому что в таком случае группа эстетических параметров оказывается самой по себе, она не выносится на поле легенды или в настройки каждого объекта. К слову, показ легенды можно запретить вовсе, установкой значения параметра show.legend = FALSE, как это показано в примере кода ниже:

```
ggplot (data = My_table) + geom_smooth (mapping = aes (x = `№№`, y = Тема2))
ggplot (data = My_table) +
geom_smooth (mapping = aes (x = `№№`, y = Тема2, group = Класс))
ggplot (data = My_table) +
geom_smooth( mapping = aes(x = `№№`, y = Тема2, color = Класс),
show.legend = FALSE)
```

Чтобы изобразить несколько графиков на одном чертеже, добавьте несколько вызовов функции geom к ggplot():

```
ggplot (data = My_table) +
geom_point (mapping = aes (x = `№№`, y = Тема2)) +
geom_smooth (mapping = aes (x = `№№`, y = Тема2))
```

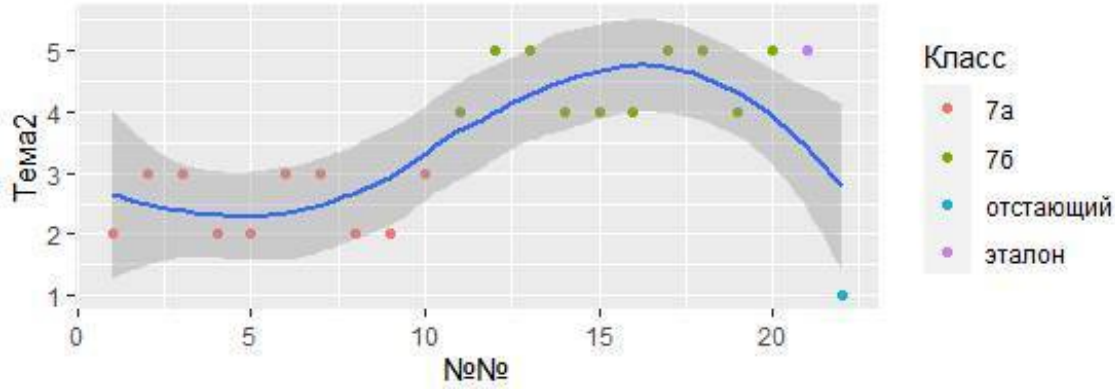
Это, однако, вносит некоторое дублирование в код. Представите, что хотите изменить ось *y* для отображения успеваемости по теме 3 вместо темы 2, нужно будет менять переменную в дух местах, при этом можно забыть про обновление в одном из них.

Дабы избежать подобного сценария, набор значений аргумента mapping передается непосредственно в функцию ggplot(). ggplot2 будет рассматривать эти значений как глобальные и применять их к каждой функции вызываемой внутри. Другими словами, следующий код создаёт ту же иллюстрацию, что и предыдущий, но более лаконичен:

```
ggplot (data = My_table, mapping = aes (x = `№№`, y = Тема2)) +
geom_point() + geom_smooth()
```

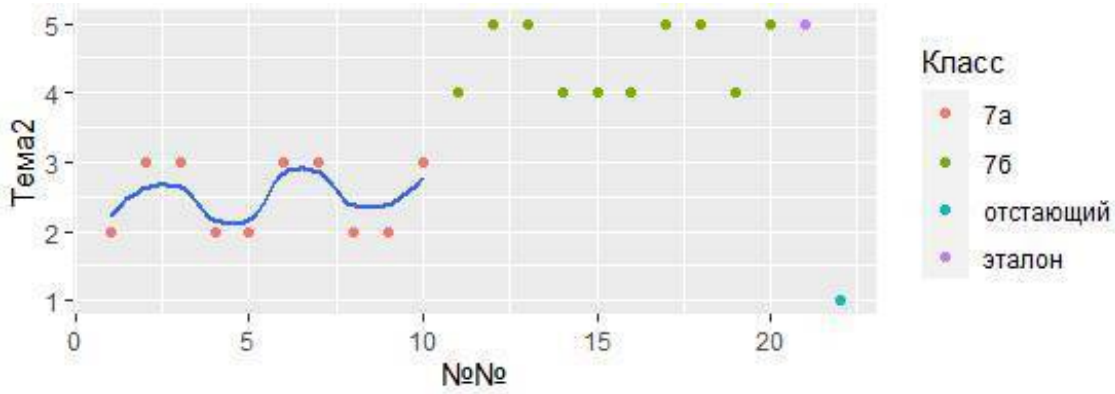
Если же размещаете параметры mapping внутри каждой функции geom, то ggplot2 будет рассматривать их как локальные настройки для слоя. Будет использоваться параметр mapping для расширения или перезаписи глобальных настроек слоя. Это позволяет настраивать различную эстетику внутри индивидуальных слоёв:

```
ggplot (data = My_table, mapping = aes (x = `№№`, y = Тема2)) +
geom_point (mapping = aes (color = Класс)) + geom_smooth()
```




Можно использовать подобную идею, чтобы выбирать разные данные для каждого слоя:

```
ggplot (data = My_table, mapping = aes (x = `№№`, y = Тема2)) +
geom_point (mapping = aes (color = Класс)) +
geom_smooth (data = My_table[My_table$Класс == "7а", ], se = FALSE)
```




В приведенном примере, гладкая линия охватывает только подмножество исходного набора данных. Локальный аргумент в geom_smooth() переопределяет глобальный аргумент отбора данных в ggplot().

Разберем как работает фильтрация чуть позже, на данный момент достаточно понять, что эта команда выбирает только учеников 7а класса, а опция se = FALSE отключает подсветку доверительного интервала.

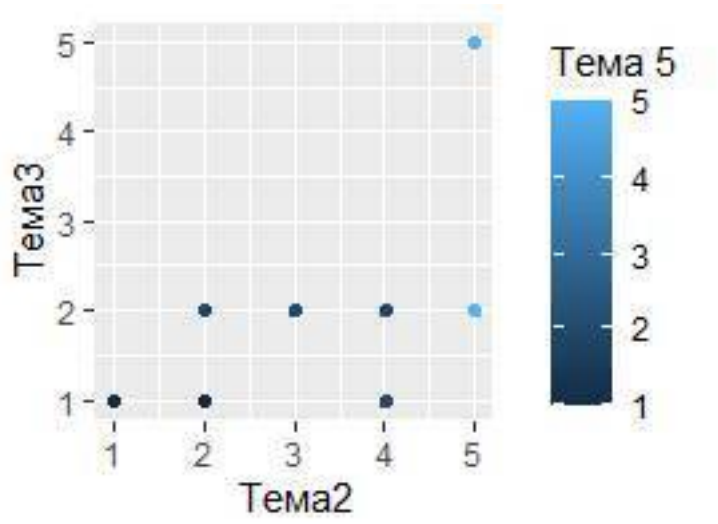
**Упражнения**

1. Какую функцию из категории geom_ вы бы использовали для построения линейного графика? А для круговой, лепестковой диаграммы, гистограммы?

2. Что меняет опция show.legend = FALSE? Что происходит если её убрать? Как думаете, почему она использовалась ранее в примере?

3. Что делает аргумент se для функции geom_smooth ()?

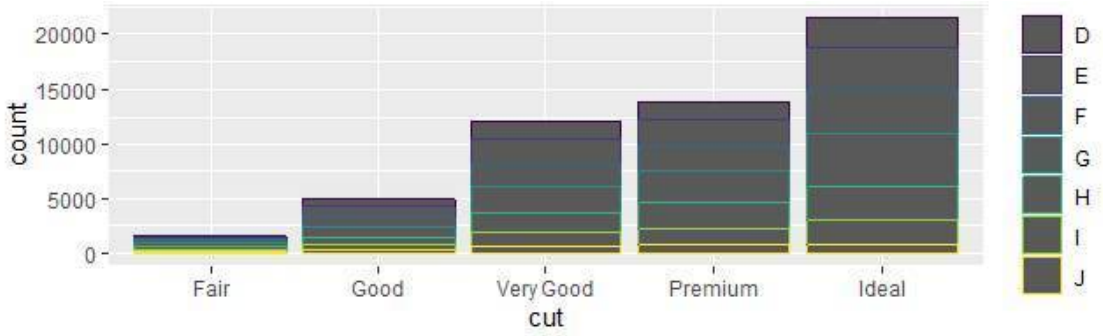
4. Воссоздайте код R, необходимый для создания следующего рисунка и дайте ему соответствующую интерпретацию:




Подробнее остановимся на гистограммах, – так называемых прямоугольных диаграммах. Они кажутся простыми, но интересны тем, что открывают потенциальные закономерности в наблюдаемой статистике. Рассмотрим базовую линейчатую диаграмму, построенную следующим образом с помощью функции geom_bar(). Принимая во внимание, как Роберт Грин Ингерсолл (1833-1899) за оффлайн-школой закрепил хлёсткое определение: «Школа – это место, где шлифуют булыжники и губят алмазы», – медленно, но верно приобщаясь к принципиально иной онлайн-школе попробуем всё же научиться правильному обращению с алмазами. На диаграмме ниже будет показано общее количество обработанных алмазов – бриллиантов, хранящихся в предустановленной с пакетом ggplot2 базе данных, сгруппированных по огранке.

База данных о бриллиантах (diamonds) поставляется в комплекте ggplot2 и содержит информацию о ~54 000 дорогостоящих украшениях, включая цену, размер в каратах, цвет, прозрачность и огранку каждого из них. Несомненно, онлайн-учителю любой по карману. Диаграмма показывает, что бриллиантов с идеальной огранкой имеется гораздо больше, чем с черновой обработкой:

```
ggplot (data = diamonds) +
geom_bar (mapping = aes (x = cut, colour = diamonds$color))
```




На оси *x* диаграмма показывает огранку (cut) алмазов. На оси *y* с учетом цвета отображается их общее количество (count), но в базе данных не хранится поле count. Откуда же берется информация о количестве? Одни алгоритмы графопостроителей, например диаграммы рас-

сеяния, формируют изображение по необработанным значениям исходного набора данных. Другие, например гистограммы, вычисляют новые вспомогательные значения при построении. Гистограммы, как частотные диаграммы, преобразуют ваши данные, осуществляют подсчеты числа записей определенного типа, будто раскладывая их по ящикам. При масштабировании последних диаграмма адаптируется к объему исходных данных, а затем строятся прямоугольники нужного размера. Вычисляется статистическая сводка выборки и после этого рисуется специально отформатированный прямоугольник. Алгоритм, используемый при вычислении новых значений для графиков, определяется параметром stat, сокращенно от «статистическое преобразование». В примере ниже показано, как это работает с geom_bar(). Вы можете узнать, какое статистическое преобразование использует та или иная функция, проверив значение по умолчанию аргумента stat. Например, в документации по функции ?geom_bar сказано, что её значение по умолчанию для аргумента stat это count, то есть geom_bar() использует функцию stat_count(), описанную на той же странице, что и geom_bar(), и если прокрутить вниз, то можно найти раздел «вычисляемые переменные», в котором сказано, что вычисляются две новые вспомогательные переменные: count и prop.

Как правило, префиксы geom_ и stat_ взаимозаменяемы. Например, можно запустить предыдущий пример с использованием stat_count() вместо geom_bar(). Это работает, потому что каждая функция категории geom_ имеет параметр stat по умолчанию, а каждая функция категории stat_ имеет двойственный параметр geom по умолчанию. Это означает, что можно используйте функции построения графиков, не беспокоясь о лежащих в их основе статистических преобразованиях данных. Есть три причины, по которым может потребоваться использовать параметр stat в явном виде:

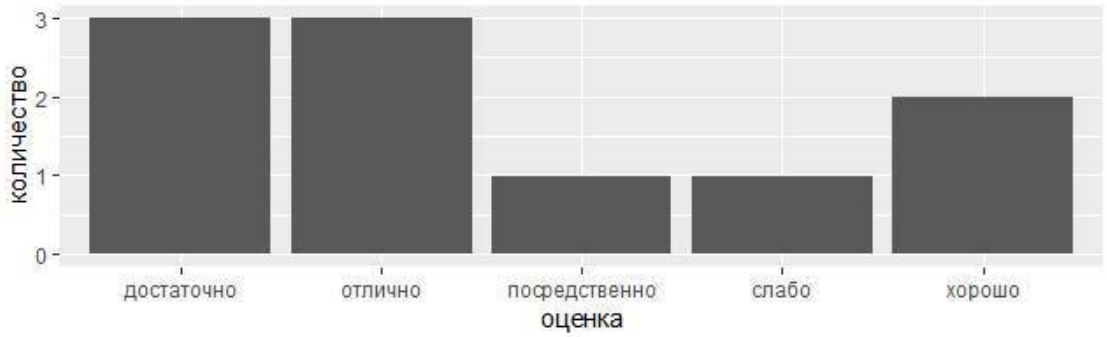
1) Возможно, захотите переопределить используемое по умолчанию статистическое преобразование. В коде ниже, заменено значение аргумента stat в geom_bar() с count (принятого по умолчанию) на identity. Это позволяет сопоставить высоту баров с необработанным значением переменной. Когда говорят о столбцевой диаграмме, можно иметь ввиду такой тип гистограммы, в котором высота столбика уже присутствует в данных, либо предыдущую диаграмму, на которой высота генерируется с помощью подсчет строк.

*Историческая справка.*

Как известно, из всех систем оценивания знаний в России поныне жива 5-балльная, которая была в 1837 году официально установлена Министерством народного просвещения. Поло-

жим, что продемонстрированные воспитанницами на одном из уроков математики в Серпуховской женской гимназии результаты были занесены в следующую демонстрационную таблицу.

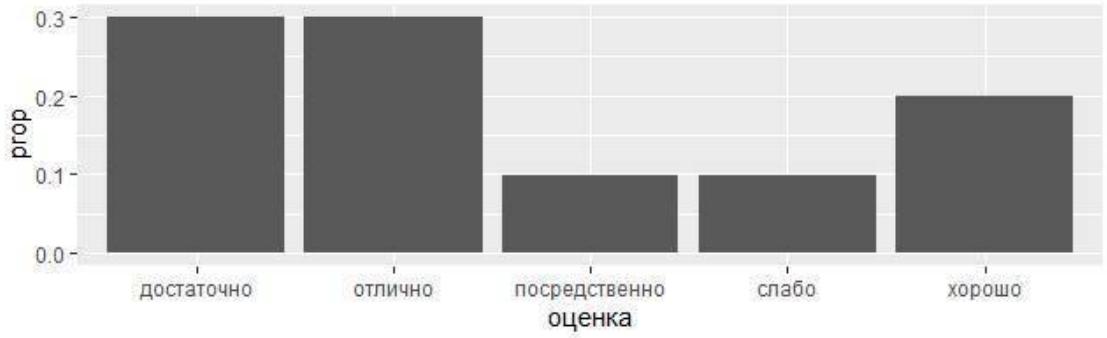
```
library(tidyverse)
demo <– tribble( ~оценка, ~количество,
"слабо", 1,
"посредственно", 1,
"достаточно", 3,
"хорошо", 2,
"отлично", 3 )
ggplot(data = demo) +
geom_bar(mapping = aes(x = оценка, y = количество), stat = "identity")
```




Не волнуйтесь, что не видели <– tribble раньше. Из контекста понятно назначение этих операторов, но что именно они делают в общем случае, будет подробно рассказано чуть позже.

2) Возможно, потребуется переопределить сопоставление по умолчанию от трансформированных переменных. Например, можете чтобы отобразить линейчатую диаграмму частот, а не количества:

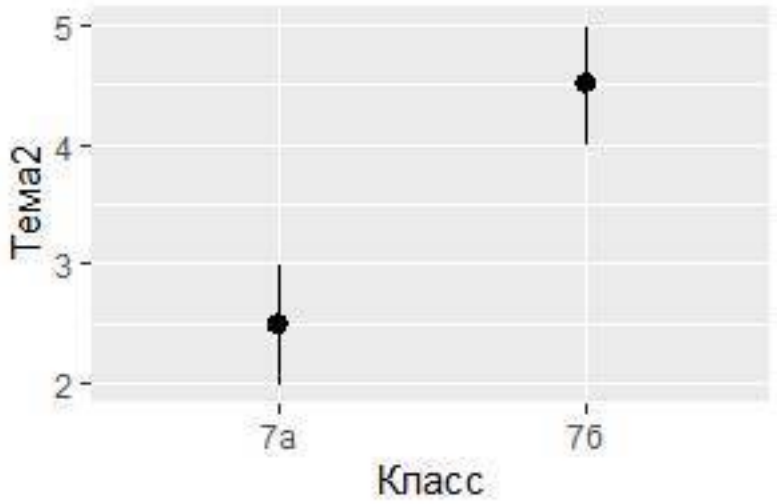
```
library(tidyverse)
demo <– tribble( ~оценка, "слабо", "посредственно",
"достаточно", "достаточно", "достаточно",
"хорошо", "хорошо",
"отлично", "отлично", "отлично" )
ggplot (data = demo) +
geom_bar (mapping = aes (x = оценка, y = stat (prop), group = 1))
```




Чтобы найти полный список переменных, вычисляемых в статистике, достаточно заглянуть в раздел справки, озаглавленный как «вычисляемые переменные».

3) Возможно, захотите извлечь больше статистической информации в вашем коде. Например, если использовать функцию stat_summary(), то будет получена дополнительная описательная статистика, которую можно показать на диаграмме. Следующий фрагмент кода выберет из тестовой базы успеваемость обучающихся в 7а или 7б классах по теме 2, найдет наименьшую оценку в каждом классе, наибольшую и медианное значение. После этого найденные статистики будут отображены на диаграмме соответствующими линиями:

```
ggplot(data = My_table[My_table$Класс == "7а" | My_table$Класс == "7б",]) +
stat_summary(
mapping = aes(x = Класс, y = Тема2),
fun.ymin = min,
fun.ymax = max,
fun.y = median
)
```




На данном этапе развития проекта, пакет ggplot2 предоставляет пользователям более 20 статистик. Каждое значение параметра stat является функцией, поэтому получить справку по ним можно обычным способом, например, введя ?stat_bin в консоли.

**Упражнения**

1. Что такое geom по умолчанию, связанный с stat_summary()? Как переписать код из примеров, чтобы использовать функцию начинающуюся с geom_ вместо stat_?

2. Что делает функция geom_col()? Чем она отличается от geom_bar()?

3. Большинство значений параметров geom и stat парные, и почти всегда используется вместе. Ознакомьтесь с документацией и составьте список всех пар, что у них общего?

4. Какие вспомогательные переменные вычисляет функция stat_smooth()? Какие параметры контролируют её поведение?

5. В диаграмме частот из примера установлено значение group = 1. Зачем? Другими словами, что будет нарисовано без указания этого параметра?

Есть еще одна интересная опция, связанная с гистограммами. Можно раскрасить её элементы с помощью любого цвета, указав значения параметров цвета границы (color) и заливки (fill). Обратите внимание, что произойдет, если сопоставите настройки заливки с отдельной переменной: каждый цветной прямоугольник будет представлять комбинированную информацию из двух параметров.

Регулировка положения прямоугольников задается соответствующим аргументом (position). Если его не менять, то построится столбчатая диаграмма, но можете использовать один из трех других вариантов: используемый по умолчанию (identity), развернутый по горизонтали (dodge) или с заполнением прямоугольников до равной высоты (fill). Указание position = "identity" будет размещать каждый объект ровно там, где он попадает в контекст графика. Это не очень полезно в случае детализированных прямоугольников, потому что фрагменты могут перекрываться между собой внутри одного прямоугольного столбика. Чтобы увидеть это перекрытие, можно сделать заливку полупрозрачной, придав уровню прозрачности (alpha) небольшое значение, либо использовав настройку fill = NA. Такое расположение прямоугольников полезно для 2d-примитивов, в виде точек. Указание position = "fill" работает как штабелирование, оно сделает каждый набор прямоугольников одинаковой суммарной высоты. Такой подход значительно облегчает сравнение пропорций внутри групп. И наконец position = "dodge" нарисует перекрывающиеся объекты непосредственно рядом друг с другом, что облегчает сравнение индивидуальных значений.

Заключительный тип регулировки является не очень полезным для гистограмм, но может быть очень полезен для диаграмм рассеяния. Вспомните примеры из первой главы. неужели не заметили, что график отображает только 126 точек, хотя в базе данных об автомобилях записано 234 значения. Как в известном письме на Балабановскую спичечную фабрику: «Я 11 лет считаю спички у вас в коробках – их то 59, то 60, иногда 58. Вы там сумасшедшие что ли все???». Источник обозначенной проблемы в том, что значения *x* и *y* округлены. В результате, многие точки появляясь на сетке перекрывают друг друга. Эта проблема известна как «overplotting». Такое расположение делает график трудным для понимания, когда на нём находится много данных. Распределены ли точки данных поровну на всем графике, или есть комбинация координат *x* и *y*, которая содержит 109 значений одновременно? Проблемы можно избежать, переключив регулировку положения в режим дрожания (jitter). Настройка position = "jitter" добавляет небольшое количество случайных шумов в каждую точку. Это распространяется на всю поверхность и поэтому не окажется двух точек, которые, вероятно, получат одинаковое количество случайных шумов. Добавление случайности кажется странным способом улучшения изображения, но несмотря на то, что график получится менее точным на малом масштабе, в больших масштабах график становится более иллюстративным. Поскольку это такая полезная опция, в ggplot2 внесена отдельная краткая форма записи выражения geom_point(position = "jitter"), вместо него лучше использовать geom_jitter().

Чтобы узнать больше о регулировке положения, загляните в раздел справки, посвященный каждой из перечисленных настроек.

**Упражнения**

1. Какие параметры функции geom_jitter() регулируют количество дрожаний?

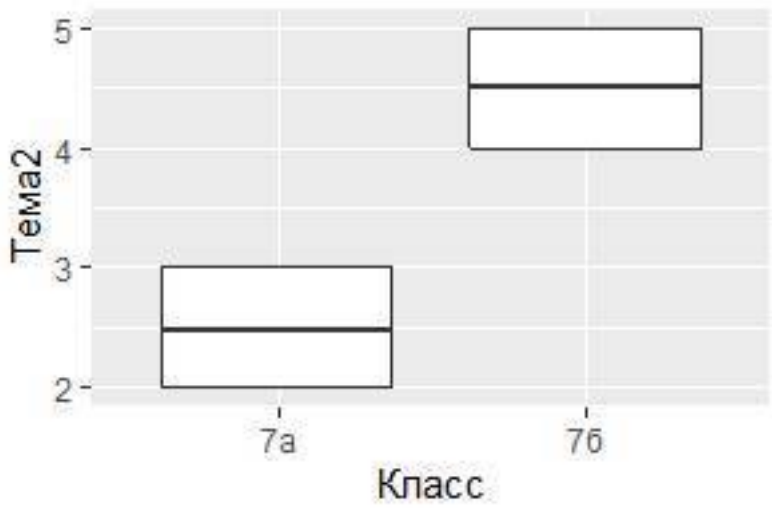
2. Примените geom_jitter() и geom_count(), сравните полученные результаты.

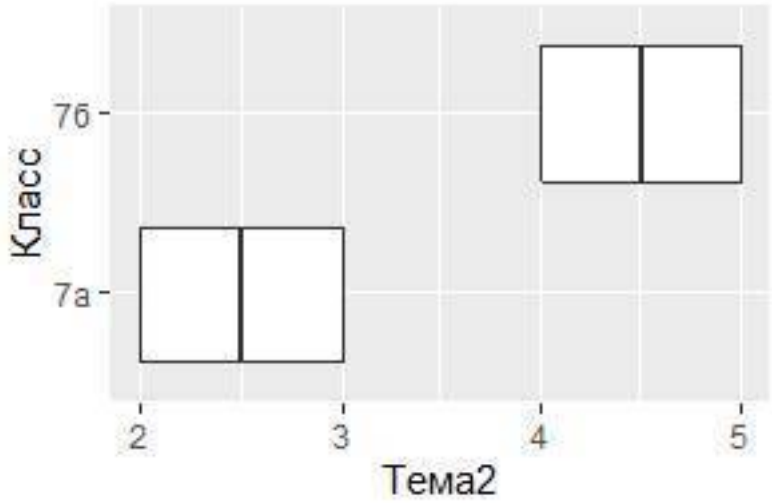
3. Какая настройка положения используется в функции geom_boxplot() по умолчанию? Создайте на её основе визуализацию своего набора данных.

Заключительной частью настоящей главы рассмотрим настройку систем координат для построения графиков. Система координат, пожалуй, имеет самый сложный функционал в ggplot2. Естественно, по умолчанию используется прямоугольная декартова система координат, в которой значения *x* и *y* позволяют однозначно определить местоположение каждой точки. Но есть и другие системы координат, которые иногда полезны. Функция coord_flip() меняет местами оси *x* и *y*. Это пригодится, если хотите нарисовать горизонтальные боковые диаграммы,а также полезно для длинных графиков, которые трудно подгонять без перекрытия по оси *x*.

```
# левый график
ggplot(data = My_table[My_table$Класс == "7а" | My_table$Класс == "7б",],
```

```
mapping = aes(x = Класс, y = Тема2)) +
geom_boxplot()
# правый график
ggplot(data = My_table[My_table$Класс == "7а" | My_table$Класс == "7б",],
mapping = aes(x = Класс, y = Тема2)) +
geom_boxplot() +
coord_flip()
```




Функция coord_quickmap() устанавливает соотношение сторон правильным для карт. Это очень важно, если строите планы карт местности с помощью ggplot2. Например:

1) Установите пакет карт, если не использовали его ранее.

```
install.packages("maps")
```

2) Подключите соответствующую библиотеку.

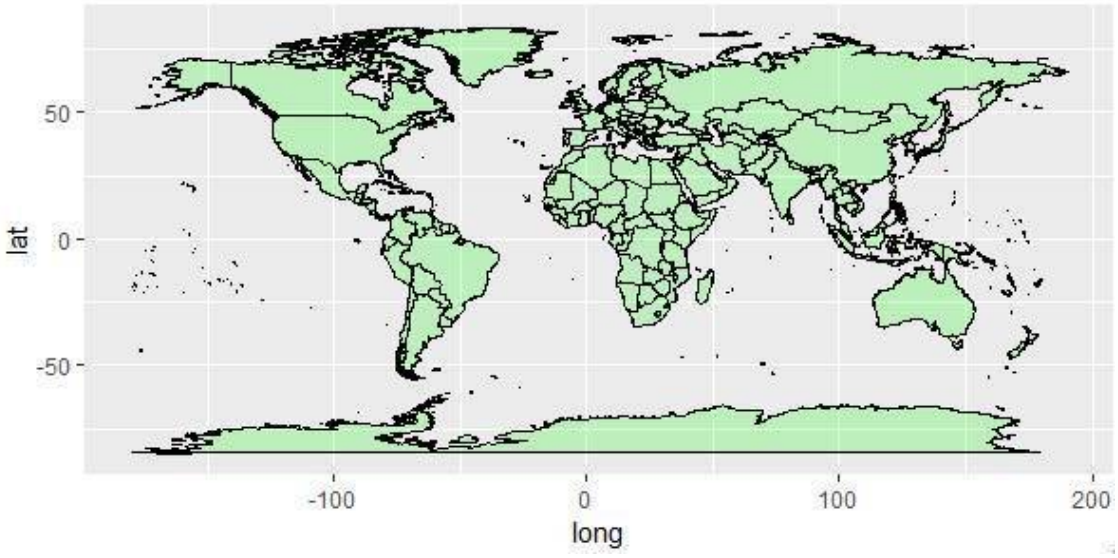
```
library(maps)
```

3) Заполните переменную картографическими данными.

```
ru <– map_data("world")
```

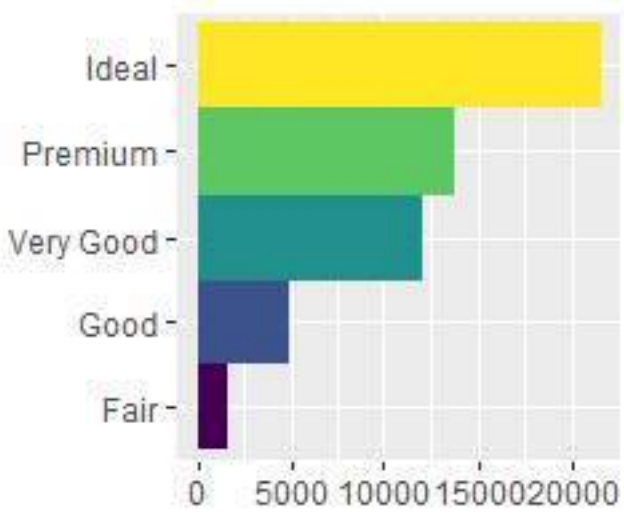
4) Теперь можно получить изображение карты в корректном масштабе

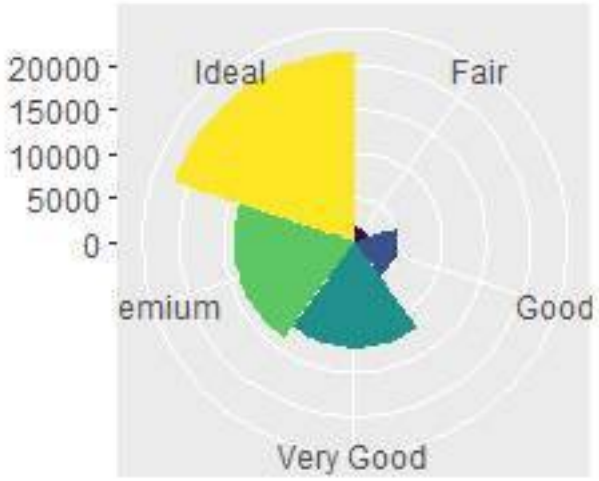
```
ggplot(ru, aes(long, lat, group = group)) +
geom_polygon(alpha=1/5, fill = "green", color = "black") +
coord_quickmap()
```




Функция coord_polar() переключает графопостроитель в режим полярных координат. Полярный координаты позволяют визуализировать интересную связь между линейчатой и круговой диаграммами. Напоследок вернёмся к тому, с чего начинали, – алмазам и их популярности в зависимости от качества. В следующем примере переменная bar заполняется вызовом процедуры формирования блоков данных для изображения. Далее, диаграмма транспонируется, тем самым приводя к линейчатому виду, и изображается в полярной системе координат отдельно:

```
bar <– ggplot(data = diamonds) +
geom_bar(
mapping = aes(x = cut, fill = cut),
show.legend = FALSE,
width = 1
) +
theme(aspect.ratio = 1) +
labs(x = NULL, y = NULL)
bar + coord_flip()
bar + coord_polar()
```




**Упражнения**

1. Преобразуйте линейчатую диаграмму с накоплением в круговую диаграмму с помощью coord_polar().

2. Где и как используется функция labs()? Ознакомьтесь с документацией.

3. В чем разница между coord_quickmap() и coord_map()?

4. Почему важно применение coord_fixed()? Что делает функция geom_abline()?

5. Выполните аналогично разобранной визуализацию успеваемости учеников своего класса.

Выше было показано как создавать диаграммы рассеяния, гистограммы и прямоугольные-диаграммы. После закрепления на практике сформировался навык, легко применимый к освоению диаграмм ggplot2 любого типа. Чтобы закрепить изученное, добавим настройки положения, статистическую обработку, настройки системы координат и разбиение данных к исходному шаблону кода:

ggplot(data = <данные>) +

<geom_основная функция графопостроителя>(

mapping = aes(<сопоставления с координатными осями и эстетикой>),

stat = <сбор дополнительной статистики>,

position = <позиция фрагментов диаграммы>
) +
<настройка координатной системы> +
<функция группирования данных>

Новый шаблон принимает семь параметров (заключенные в угловые скобки), которые применяются для описания желаемой визуализации данных. На практике редко приходится заполнять их все, чтобы построить график, так как в ggplot2 используются оптимальные значения по умолчанию для всего кроме данных, сопоставлений с осями и выбора функции geom.

Семь параметров в шаблоне составляют грамматику графопостроителя, формальную систему визуализации изображений. Грамматика основана на понимании того, что можно однозначно описать любой участок кода как комбинацию набора данных, функции графопостроителя, набора соответствий, статистической обработки, настройки положения фрагментов чертежа, системы координат и схемы группирования подмножеств исходных данных.

Чтобы понять, как это работает, вспомните, как строился простейший график с самого начала: фиксировался набор данных, затем выполнялась статистическая обработка для извлечения вспомогательной информации. Далее, выбирался способ представления каждого исходного значения и новых данных. При этом настраивались эстетические свойства геометрических примитивов, чтобы сопоставление значений каждой переменной с положением, цветом или формой объекта несло определенную смысловую нагрузку. Затем выбирали систему координат, чтобы в ней наглядно разместить полученное изображение, это само по себе тоже несет определенную эстетику, сопоставляя значения переменных с *x* и *y*. В результате получался график, но опционально ещё настраивалось местоположение объектов внутри системы координат (корректировка положения) и разбиение графика на подграфики (фасетирование). Также можно было улучшить изображение, добавив один или несколько дополнительных слоёв, на каждом из которых использовался свой набор данных, функция графопостроителя, набор сопоставлений, собиралась дополнительная статистика и регулировалось положение.

При помощи описанного метода строятся графики практически любой сложности. Другими словами, выкристаллизовавшийся в главе шаблон кода охватывает сотни тысяч уникальных графиков.

Перейдем ко второй части, анонсированной в названии раздела. Да, визуализация является важным инструментом понимания, но считается большой удачей получить исходные данные сразу в пригодном для визуализации формате. Часто приходится создавать новые переменные или сводные таблицы, переименовывать переменные или изменять порядок следования наблюдений, чтобы сделать данные немного проще для повышения наглядности их визуализации. Рассмотрим, как сделать все это и многое другое, как преобразовывать данные с помощью пакет dplyr на примере обширного набора данных о рейсах, вылетающих из Нью-Йорка.


*Историческая справка.* На прилагаемом фото запечатлён трансарктический самолёт АНТ-25 в ангаре аэропорта Флойд Беннет, февраль 1939 года, ознаменовавший успешное участие СССР в Нью-Йоркской выставке 1939-1940 годов. Нью-Йорк, Бруклин.

Сосредоточимся на том, как использовать пакет dplyr, – один из базовых инструментов tidyverse. Проиллюстрируем ключевые идеи, используя данные из базы nycflights13, и пакета ggplot2, чтобы визуализировать эти данные.

```
library(nycflights13)
library(tidyverse)
```

Обратите внимание на сообщение о возможных конфликтах, которое выводится при загрузке tidyverse, так как dplyr перезаписывает некоторые функции R. Если хотите использовать эти функции после загрузки dplyr, то нужно будет вводить их полные имена через два двоеточия, например, stats::filter().

Чтобы изучить основные способы работы с данными из dplyr, будем пользоваться базой данных nycflights13::flights, она содержит информацию по всем 336 776 рейсам, вылетевшим из Нью-Йорка. Данные поступают из Бюро статистики транспорта США, и вы можете с ними ознакомиться в любое время, просто введя в консоли:

```
flights
```

Заметим, что эта база данных при выводе в консоль отличается от вывода из других баз данных, которые применяли ранее. Показаны лишь первые несколько строк и столбцы, которые поместились на экране. Чтобы просмотреть весь набор данных, необходимо запустить:

```
view(flights)
```

Откроется таблица средствами просмотра RStudio, в слегка упрощенном виде, чтобы легче было применять инструментарий tidyverse. На данный момент не нужно беспокоиться о нюансах, позже вернемся к табличному представлению данных в соответствующей главе. Ряд из нескольких буквенных сокращения под названиями столбцов описывает тип каждой переменной: int означает целые числа; dbl означает действительные числа; chr означает символьные строки; dttm означает дату-время (дата + время). Существуют и другие распространенные типы переменных, они не используются в данном наборе, но будут рассмотрены отдельно: lgl означает логические значения, которые содержат только TRUE или FALSE; fctr означает факторы, которые R использует для представления категориальных переменных с фиксированными возможными значениями; date означает данные.

Следующие пять ключевых функций dplyr позволяют решить подавляющее большинство задач обработки данных: filter() отфильтрует наблюдения по заданным условиям; arrange() меняет порядок строк; select() выберет переменные по их именам; mutate() создаёт новые переменных со свойствами существующих переменных; summary() сворачивает множество значений до одного. Перечисленные функции можно использовать совместно с group_by(), которая изменяет область действия каждой функции со всего набора данных на определенные группы. Собственно перечисленные шесть функции и предоставляют собой команды языка обработки данных.

Все функции работают по общей схеме:

1) Первый аргумент – фрагмент данных.

2) Последующие аргументы описывают, что нужно делать с выбранными данными, используя имена переменных без кавычек, либо в одиночных апострофах кавычках «'», если имена содержат пробел « ».

3) Результатом является новый фрагмент данных.

Перечисленные свойства делают легко реализуемой последовательность из нескольких простых шагов к достижению желаемого результата. Разберем на примерах, как это работает. filter() позволяет выбирать подмножество наблюдений на основе определенных условий. Первый аргумент содержит имя базы данных. Второй и последующие аргументы являются выражениями, фильтрующими данные. Например, выберем все рейсы на 5 мая следующей командой:

```
filter(flights, month == 5, day == 5)
```

Когда запускаете эту строку кода, dplyr выполняет операцию фильтрации и возвращает новый блок данных. Функции dplyr никогда меняют входные данные, поэтому, если понадобится сохранить результат, то придется использовать оператор присваивания:

may5 <– filter(flights, month == 5, day == 5)

R либо распечатывает результаты, либо сохраняет их в переменную. Когда нужно сделать и то, и другое, команда заключается в круглые скобки:

(may5 <– filter(flights, month == 5, day == 5))

Чтобы эффективно использовать фильтрацию, нужно знать, как выбрать наблюдения, используя операторы сравнения. R предоставляет стандартный набор операторов: > (больше), >= (больше или равно), < (меньше), <= (меньше или равно), != (не равны), == (равны). Начинающие пользователи R зачастую ставят = вместо == при проверке равенства. Если допустить такое, то возникнет предупреждение об ошибке. Есть еще одна распространенная проблема, с которой сталкиваются при использовании ==, это числа с плавающей запятой. Поистине альтернативная арифметика:

```
sqrt (4) ^ 2 == 4
# > [1] TRUE
sqrt (5) ^ 2 == 5
# > [1] FALSE
1 / 50 * 50 == 1
# > [1] TRUE
1 / 49 * 49 == 1
# > [1] FALSE
```

Дело в том, что в R используется арифметика конечной точности, так как затруднительно хранить бесконечное количество цифр, либо реализовывать алгебраический подход. Поэтому каждое число в R является приближением, а вместо оператора == нередко используется функция near(), позволяющая сравнивать приближенные величины:

```
near(sqrt(5) ^ 2, 5)
# > [1] TRUE
```

Несколько аргументов функции filter() перечисленные через запятую равносильны объединению условий союзом «и», при этом, каждое выражение должно оказаться истинным, чтобы из входных данных соответствующая запись была сохранена в выходные данные. Для остальных логических связок можно использовать булевы операторы: & это «и», | это «или», ! это отрицание «не», xor(*x*, *y*) это исключающее или с аргументами *x*, *y*.

Следующий код находит все рейсы, которые вылетели в феврале или марте:

```
filter(flights, month == 2 | month == 3)
```

Если попытаться ввести команду буквально

```
filter(flights, month == (2|3))
```

то вместо желаемого будут найдены все месяцы равные результату булевой операции 2|3, значение которой обращается в TRUE. В числовом контексте TRUE становится равным единице 1, поэтому будут найдены все январские вылеты, что вовсе не соответствует задуманному.

Полезным клавиатурным сокращением для решения обозначенной проблемы является *x* %in% *y*. Это позволит выбрать каждую строку, где *x* является одним из значений в *y*. Можно было бы использовать следующую альтернативу для кода выше:

```
filter(flights, month %in% c(2, 3))
```

Иногда можно упростить сложное выражение вспомнив законы де Моргана из курса математической логики: !(x & y) == !x | !y, и !(x | y) == !x & !y. Например, если нужно найти рейсы, которые не задерживались (по прилету или отправлению) более чем на час, можно воспользоваться любым из следующих фильтров:

```
filter(flights,!(arr_delay > 60 | dep_delay > 60))
```

filter(flights, arr_delay <= 60, dep_delay <= 60)

Кроме & и |, в R есть && и ||, но не используйте их сейчас, позже узнаете, при каких условиях уместно их применение.

Всякий раз, когда используется сложное составное выражение в filter(), предпочтительнее разбить выражение на несколько вспомогательных, это значительно упрощает последующую проверку работы. Вскоре узнаем, как быстро создать новые переменные. Одна важная особенность R, которая может затруднить фильтрацию, это пропущенные значения, или недоступные (NA), которые представляют собой неизвестное значение, поэтому пропущенные значения являются изгоями, практически любая операция с участием NA приведет к NA.

```
NA > 1
#> [1] NA
2 == NA
#> [1] NA
NA + 3
#> [1] NA
NA / 4
#> [1] NA
```

Самым алогичным результатом может показаться следующий:

```
NA == NA
#> [1] NA
```

Но его легко понять в конкретном контексте: совпадает ли содержимое двух ящиков, внутри которых неизвестно что? Мы не знаем! Если хотите определить, отсутствует ли значение конкретной переменной, можно воспользоваться функцией is.na(), в качестве аргумента задав интересующее имя. Функция filter() отбирает только те строки, для проверяемые условия обращаются в TRUE, при этом исключаются как значения FALSE, так и NA. Если хотите сохранить пропущенные значения, то запрашивайте их в явном виде:

```
filter (flights, is.na(month) | month > 1)
```

**Упражнения**

1. Найти все рейсы, которые: имели задержку прибытия на два и более часа; прилетели в Хьюстон; управлялись компанией Delta; улетели летом; прибыл с опозданием более чем на два часа; задержались они как минимум на час, но наверстали более 30 минут в полете; отбыли между полуночью и 6 утра (включительно).

2. Функция between() из пакета dplyr тоже полезна для фильтрации. А что она делает? Можно ли использовать её для упрощения кода, необходимого для получения ответов в предыдущем задании?

3. Сколько рейсов имеет отсутствующее значение dep_time? Какие еще переменные у них отсутствуют? Что могут представлять собой эти записи в базе данных?

4. Почему значение NA^0 определено, NA / TRUE не определено, а FALSE & NA определено? Можете ли сформулировать общее правило, охватывающее и случай NA * 0?

Функция arrange () работает аналогично функции filter(), за исключением того, что вместо выбора строк, сортирует их. На вход принимаются данные и набор имен столбцов (или более сложных выражений), чтобы задать отношение порядка по возрастанию. Если укажете более одного имени столбца, то каждый последующий столбец будет сортировать значения строк с равными значениями из предыдущих столбцов:

```
arrange(flights, year, month, day)
```

Используя desc() можно переданный в аргументе столбец упорядочить по убыванию значений:

```
arrange(flights, desc(dep_delay))
```

Пропущенные значения (NA) всегда оказываются в конце сортировки.

**Упражнения**

1. Как использовать функцию arrange() для переноса всех пропущенных значений в начало списка? (Подсказка: применимо is.na()).

2. Сортировка рейсов позволяет найти самые задерживаемые рейсы. Найдите рейсы, которые вылетали пунктуальнее всех.

3. Отсортируйте рейсы так, чтобы найти самые скоростные перелёты.

4. Какие рейсы летали дальше всех? Какой маршрут был самым коротким?

Нередко формируемые наборы данных содержат сотни или даже тысячи записей. В таком случае проблематично даже просто найти интересующую переменную. Функция select() позволяет быстро сузить поле зрения исследователя, сконцентрировав его на нужных именах переменных. Конечно, select() не очень полезна для базы авиаперелётов, так как здесь лишь 19 переменных, но продемонстрируем общую идею:

```
# поимённый выбор столбцов «месяц», «день»
select(flights, month, day)
# выбор всех столбцов между «месяц» и «день» включительно
select(flights, month:day)
# выбор всех столбцов, кроме тех, что лежат между «месяц» и «день» включительно
select(flights, -(month:day))
```

Существуют вспомогательные функции, которые уместно вызывать внутри select(): функция starts_with("абв") выбирает имена столбцов начинающихся с «абв»; функция ends_with("эюя") выбирает имена столбцов заканчивающиеся на «эюя»; функция contains("клм") выбирает имена содержаие подстроку «клм»; функция matches("(.)\\1") выбирает переменные, имена которых соответствуют заданному регулярному выражению, конкретно в данном случае магическим образом выбираются переменные, содержащие повторяющиеся символы, подробнее о регулярных выражениях в строках расскажем в соответствующей главе; вызов num_range("m", 2:4) соответствует набору m2, m3, m4. Всегда можно заглянуть в ?select для получения более подробной информации.

А еще, select() можно использовать для переименования переменных, но это редко когда бывает полезным, так как отбрасывает не упомянутые явно переменные. Вместо этого для переименования используется функция rename(), который является вариантом select(), но сохраняет все переменные, которые не указаны явно:

```
rename(flights, год = year)
```

Другой вариант использования select(), совместно со вспомогательной функцией everything(), бывает необходим если есть несколько переменных, которые нужно переместить в начало базы данных. Например, месяц (month) и день (day) вылета будут показаны первыми при выводе данных из таблицы, содержащей информацию обо всех авиаперелётах (flights) по команде:

```
select (flights, month, day, everything ())
```

Аналогично запланированную дату и время полёта (time_hour), и время, проведенное в воздухе, выраженное в минутах (air_time) можно перекинуть в начало.

**Упражнения**

1. Примените мозговой штурм, чтобы найти как можно больше способов выборки значений переменных содержащих информацию о времени из базы данных flights.

2. Что произойдет, если имя одной переменной использовать несколько раз при вызове функции select()?

3. Что делает функция one_of()? Насколько полезно её применение в сочетании с вектором c("month", "day")?

4. Является ли результатом выполнения следующего кода неожиданным? Что вспомогательные функции выбора переменных в нём возвращают по умолчанию? Как изменить их значение по умолчанию?

select(flights, -contains("TIME"))

Помимо выборки существующих столбцов, полей таблицы базы данных, переменных, бывает необходимым добавление новых столбцов, которые хранят значения, являющиеся функциями от существующих. Это выполняется путём обращения к функции mutate(), которая всегда добавляет новые столбцы в конце имеющегося набор данных. Поэтому создадим более узкий набор данных, чтобы видеть новые переменные. Помните, что в RStudio самый простой способ увидеть все столбцы таблицы это вызов функции view(). Создадим укороченный_вариант_таблицы, содержащий все поля между «год» (year) и «день» (day) включительно, плюс поля, содержащие информацию о задержках (заканчивающиеся на delay), покрытом расстоянии (distance) и времени полёта (air_time) в минутах:

укороченный_вариант_таблицы <– select(flights,
year:day,
ends_with("delay"),
distance,
air_time)

Теперь добавим вычисляемые поля с информацией об опоздании, – задержке вылета минус задержка прилета, в минутах, и о средней скорости полёта. Обратите внимание, что можно ссылаться на столбцы, которые уже созданы. Если вдруг захотите сохранить только новые переменные, то используйте transmute() вместо mutate():

mutate(укороченный_вариант_таблицы,
опоздание = dep_delay – arr_delay,
скорость = distance / air_time * 60,
часы_полёта= air_time / 60,
опоздание_в_каждом_часе = опоздание / часы_полёта )

Существует много функций для создания новых переменных, которые можно комбинировать с mutate(). Ключевое их свойство заключается в том, что функция должна быть пригодной для обработки векторов, то есть она должна принимать вектор значений на входе и возвращать вектор с тем же количеством значений на выходе. Нет возможности перечислить все такие функции, но приведём некоторые из реально используемых.

Арифметические операторы: +, -, *, /, ^. Все они работают с векторами используя так называемые «правила рециркуляции», заключающиеся в том, что если один параметр короче другого, то произойдет автоматическое удлинение до равного размера путём клонирования короткого вектора достаточное количество раз. Это полезно, когда один из аргументов – число. В примере выше так были вычислены часы_полёта делением вектора на скаляр, а скорость умножением вектора на скаляр. Арифметические операторы также полезны в связке с агрегирующими функциями, о которых узнаете позже. Например, x / sum(x) вычисляет долю от общей суммы значений переменной, а y – mean(y) вычисляет отклонение величины от среднего.

Модулярная арифметика: %/% (целочисленное деление) и %% (остаток), здесь x == y * (x %/% y) + (x %% y). Модулярная арифметика очень удобный инструмент, потому что позволяет представлять большие целые числа сравнительно небольшими остатками. Например, в наборе данных flights можно выделить полные часы и оставшиеся минуты из общей продолжительности полёта, представленной в формате ЧЧММ или ЧММ (dep_time). Тогда вместо хранения и выполнения различных операций над одним большим числом, можно будет хранить и выполнять операции над двумя маленькими:

transmute(flights,

dep_time,
час = dep_time %/% 100,
минута = dep_time %% 100)

Логарифмические функции: log(), log2(), log10(), являются невероятно полезным преобразованием при работе с данными, диапазон которых охватывает несколько порядков наблюдаемой величины. Они также преобразуют мультипликативные операции в аддитивные, к этой особенности вернемся в разделе, посвященном моделированию. При прочих равных условиях, рекомендуется использовать функцию log2() так как её значения легко интерпретировать: разница в 1 на логарифмической линейке соответствует удвоению в исходном масштабе, а разница в -1 соответствует делению пополам.

Смещения: вперёд lead() и назад lag() позволяют просматривать последующие и предыдущие значения списка. Бывают необходимо вычислить приращение аргумента, например, x – lag(x), или проверить неизменность его значений, выражением x != lag(x). Смещения особенно полезны в сочетании с group_by(), но не будем забегать вперёд.

Накопительные и скользящие агрегаторы: R предоставляет функции для вычисления накапливаемой суммы cumsum(), произведения cumprod(), минимума cummin() и максимума cummax() элементов списка; кроме того, dplyr имеет функцию cummean() для вычисления среднего значения. Если нужны скользящие агрегаторы, когда сумма вычисляется по скользящему окну, то обращаются к функционалу пакета RcppRoll.

Логические сравнения: < (меньше), <= (не больше), > (больше), >= (не меньше), != (не равны), и == (равны), о них мы узнали ранее. Напомню лишь, если осуществляется сложная последовательность логических операций, то настоятельно рекомендуется сохранять промежуточные значения в отдельных вспомогательных переменных, чтобы проверить значение выражения на каждом шаге вычислений.

Ранжирование: объединяет в себе целый ряд функций, начиная с min_rank(), которая осуществляет вычисление простого порядкового номера (например, 1-й, 2-й, 3-й, 4-й). По умолчанию присваиваются меньшие номера меньшим значениям, но можно воспользоваться функцией desc() для обращения порядка значений аргумента, чтобы придать наибольшие порядковым номера наименьшим значениям элементов исходного списка. Если min_rank() не делает то, что нужно, загляните в описание функций ранжирования на страницах справки для получения более подробной информации.

**Упражнения**

1. На переменные, хранящие длительность перелёта, удобно смотреть, но трудно выполнять операции над ними, так как они не совсем порядковые числа, за 159 (которое символизирует 1 час, 59 минут) идет сразу 200 (2 часа ровно). Конвертируйте их в более удобное представление, чтобы хранилось общее количество минут начиная с полуночи.

2. Сравните значения часы_полёта с опоздание_в_каждом_часе. Что надеялись увидеть и что увидели? Что нужно сделать, чтобы исправить ошибку?

3. Найдите 10 самых задерживаемых рейсов, используя функции ранжирования. Как это связано? Внимательно прочитайте текст документация по min_rank().

4. Что возвращает 2:4 – 5:8 и почему?

5. Какие тригонометрические функции определены в R?

Последняя ключевая функция summary(), – она собирает сводную статистику по переменным при помощи вспомогательных функций. Например, среднее значение (mean) переменной dep_delay посчитается в переменную средняя_задержка_рейсов:

summarise(flights, средняя_задержка_рейсов = mean(dep_delay, na.rm = TRUE))

Ниже объясним подробно, что значит последний параметр na.rm = TRUE. Функция summary() не очень полезна, если используется без вспомогательной функции group_by(), которая переключает разбивает анализ всего набора данных на отдельные группы. Когда вызы-

вается функция из пакета dplyr на сгруппированных данных, автоматически подключается group_by() для распараллеливания вычислений в целях повышения производительности и дробления информации. Например, если применить точно такой же вызов как в предыдущем примере, но для сгруппированных по дате записях, то на выходе получится средняя задержка по дням:

```
сгруппированные_по_дням <– group_by(flights, year, month, day)
summarise(сгруппированные_по_дням,
средняя_задержка_рейсов_по_датам = mean(dep_delay, na.rm = TRUE))
```

Вызов функций group_by() совместно с summary() чаще всего используется при работе в пакете dplyr для получения статистических отчетов по группам. Но прежде, чем погрузиться в детали, дополнительно изложим одну техническую идею, касающуюся обработки информации путём её направления по специальным каналам. Представьте, что хотим исследовать закономерность между расстоянием и средней задержкой рейса для каждого пункта назначения. Опираясь на имеющиеся знания о возможностях dplyr, для этого достаточно использовать такой код:

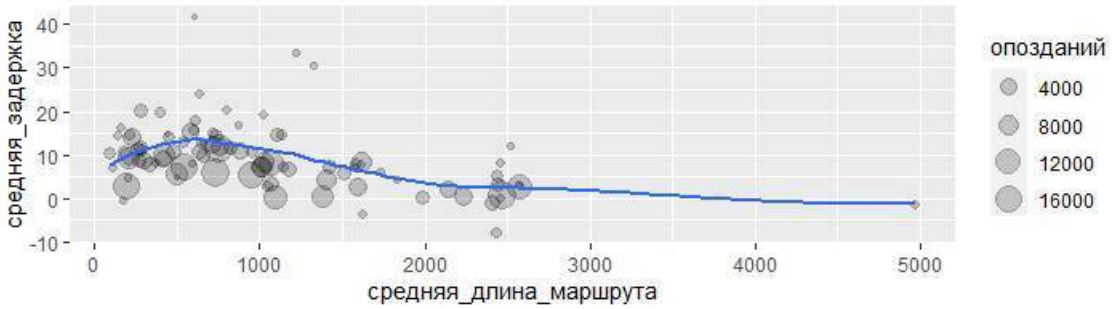
```
группы_рейсов_по_месту_назначения <– group_by(flights, dest)
задержки <– summarise(группы_рейсов_по_месту_назначения,
опозданий = n(), средняя_длина_маршрута = mean(distance, na.rm = TRUE),
средняя_задержка = mean(arr_delay, na.rm = TRUE))
```

Оставим в выборке рейсы имеющие более сотни регулярных опозданий и, например, не на московских направлениях:

```
задержки <– filter(задержки, опозданий > 100, dest != "MSK")
```

Визуализируем оставшиеся записи:

```
ggplot(data = задержки, mapping =
aes(x = средняя_длина_маршрута, y = средняя_задержка)) +
geom_point(aes(size = опозданий), alpha = 1/5) +
geom_smooth(se = FALSE)
```




Похоже, что задержки растут с увеличением расстояния до ~750 миль, а затем сокращаются. Неужели, когда рейсы становятся длиннее, появляется возможность компенсировать опоздание находясь в полёте?

Предварительно было пройдено три вспомогательных этапа подготовки данных:

1. Сгруппированы рейсы по направлениям.

2. В каждой из групп усреднены расстояния, длительность задержки и вычислено количество опоздавших рейсов.

3. Отфильтрованы шумы и аэропорт, который не подчиняется законам логики.

Этот код немного перегружен, так как каждому промежуточному блоку данных присвоено имя. Вспомогательные таблицы сохранялись, даже когда их содержимое не востребовано на заключительном этапе, и замедляли анализ. Но есть отличный способ справиться с обозна-

ченной проблемой посредством настройки каналов передачи данных служебным оператором %>%:

```
задержки <– flights %>%
group_by(dest) %>%
summarise(
опозданий = n(),
средняя_длина_маршрута = mean(distance, na.rm = TRUE),
средняя_задержка = mean(arr_delay, na.rm = TRUE) ) %>%
filter(опозданий > 100, dest != " MSK ")
```

Такой синтаксис фокусирует внимание исследователя на выполняемых преобразованиях, а не на том, что получается на каждом из вспомогательных этапов, и делает код более читаемым. Это звучит как ряд предписаний: сгруппируй, после этого подведи итоги, после этого отфильтруй полученное. Как подсказывает здравый смысл, можно читать %>% в коде как «после этого». По сути же, формируется информационный канал последовательной передачи данных на обработку от одной функции через другую к третьей. Технически, x %>% f(y) превращается в f(x, y), а x %>% f(y) %>% g(z) превращается в композицию функций g(f(x, y), z) и так далее, что позволяет использовать канал для объединения нескольких операций в одну, которую можно читать слева направо, сверху вниз. Будем часто пользоваться каналами, так как это значительно упрощает читаемость кода, разберём их более подробно в соответствующем разделе.

Работа с каналами это одна из ключевых особенностей tidyverse. Единственным исключением является ggplot2, так как библиотека была написано до появления такой возможности в R. К сожалению, являющаяся наследником ggplot2 библиотека ggvis хотя и поддерживает работу с каналами, но пока еще не в полной мере.

Внимательный читатель наверняка задавался вопросом о смысле и предназначении аргумента na.rm. Что будет, если его не писать? Получим много пропущенных значений! Дело в том, что агрегационные функции подчиняются обычному правилу пропущенных значений: если на входе есть какое-либо отсутствующее значение, то выход будет отсутствующим значением. К счастью, все функции агрегации имеют аргумент na.rm, который удаляет пропущенные значения перед началом вычислений. В том случае, где пропущенные значения представляют отмененные рейсы, мы также могли бы решить эту проблему, сначала удалив отмененные рейсы. Сохраним этот набор данных, чтобы использовать его повторно в нескольких следующих нескольких примерах:

```
неотмененные <– flights %>%
filter(!is.na(dep_delay), !is.na(arr_delay))
```

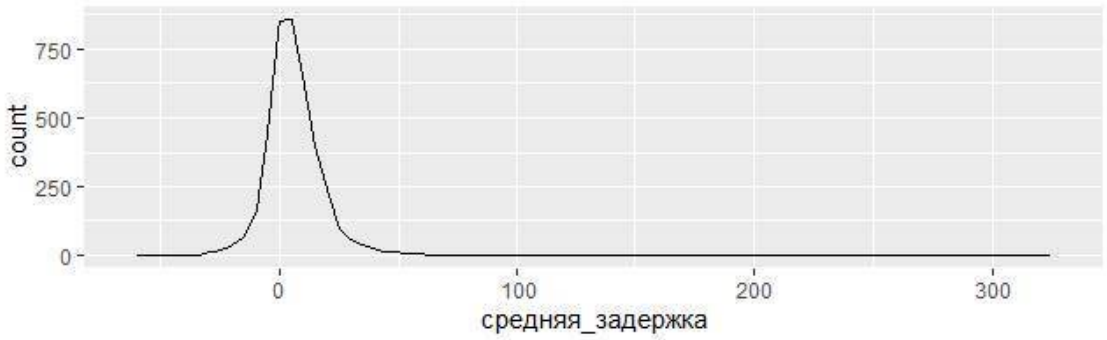
Сгруппируем получившиеся данные о неотмененных рейсах по датам и посчитаем среднюю задержку на каждую дату в отдельности:

```
неотмененные %>%
group_by(year, month, day) %>%
summarise(средняя_задержка = mean(dep_delay))
```

Всякий раз, когда осуществляется подобная агрегация, правилом хорошего тона является добавление счетчика числа учтенных значений функцией n(), либо путём подсчета используемых непустых значений командой sum(!is.na(x)). Таким способом можно удостовериться, что не делается поспешных выводов на основании выборок очень малых объемов. Например, сгруппировав рейсы по бортовому номеру, хранящемуся в переменной tailnum из таблицы неотмененных рейсов, на графике посмотрим каковы самые высокие задержки в среднем на борт:

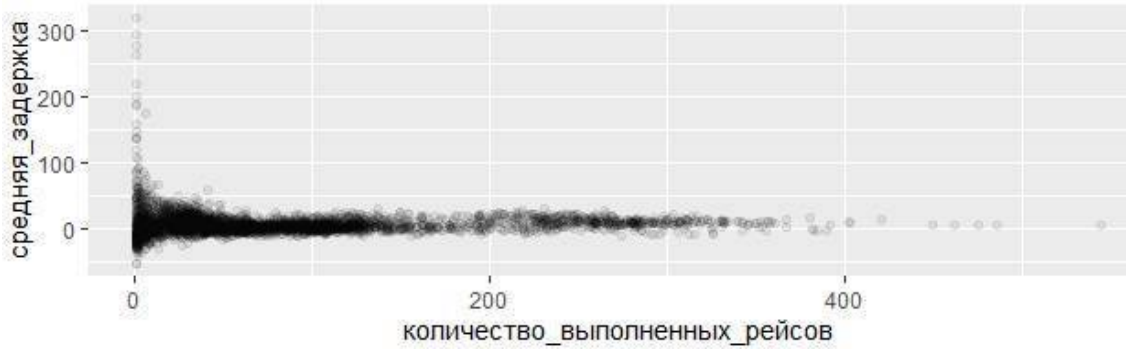
```
задержки <– неотмененные %>%
group_by(tailnum) %>%
```

```
summarise(
средняя_задержка = mean(arr_delay)
)
ggplot(data = задержки, mapping = aes(x = средняя_задержка)) +
geom_freqpoly(binwidth = 5)
```




Неужели много самолетов со средней задержкой рейса более 5 часов (300+ минут)? На самом деле не всё так печально, как могло показаться при поверхностном ознакомлении. Можно получить более глубокое представление об опозданиях, если нарисовать диаграмму рассеяния количества рейсов относительно средней задержки:

```
задержки <– неотмененные %>%
group_by(tailnum) %>%
summarise(
средняя_задержка = mean(arr_delay, na.rm = TRUE),
количество_выполненных_рейсов = n()
)
ggplot(data = задержки, mapping = aes(x = количество_выполненных_рейсов,
y = средняя_задержка)) +
geom_point(alpha = 1/15)
```

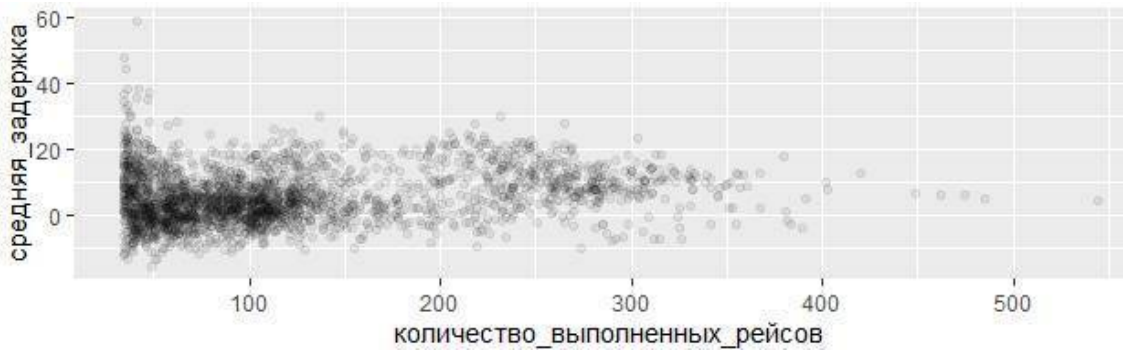



Неудивительно, что на частых рейсах задержек практически не наблюдается, а в основном задерживаются те борта, чьих рейсов мало. Что характерно, и в принципе соответствует статистическому закону больших чисел: всякий раз, когда ищется среднее значение (или другая сводка) в сравнении с размером группы, приходят к выводу, что вариативность вычисленного значения уменьшается по мере увеличения объема выборки.

Именно поэтому, когда решаете аналогичные задачи, полезно отфильтровывать группы с наименьшим количеством наблюдений, тогда можно будет увидеть общие закономерности и

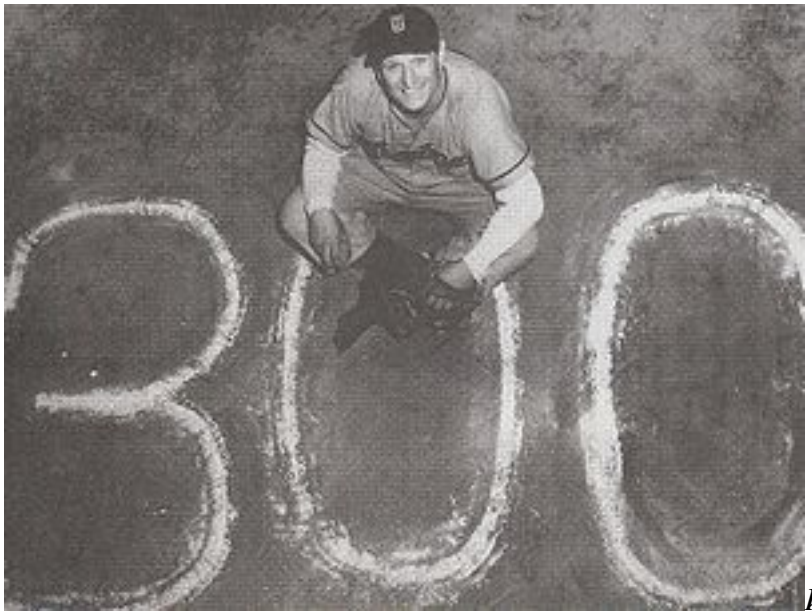
уменьшить выбросы значений на малых группах. На примере следующего кода будет демонстрирован удобный шаблон интеграции ggplot2 с каналами в dplyr. Немного странным может показаться смешение стилей %>% и +, дело привычки, со временем это станет естественным. Отфильтруем на предыдущем графике экспериментальные самолёты с малым количеством вылетов, не превышающим 33:

```
задержки %>% filter(количество_выполненных_рейсов > 33) %>%
ggplot(mapping = aes(x = количество_выполненных_рейсов,
y = средняя_задержка)) +
geom_point(alpha = 1/15)
```




Полезным сочетанием клавиш RStudio является Ctrl + Shift + P, для повторной отправки ранее отправленного фрагмента из редактора в консоль. Это очень удобно, когда экспериментируете с граничным значением 33 в приведенном выше примере: отправляете весь блок в консоль нажатием Ctrl + Enter, а затем изменяете значение границ фильтрации на новое и нажимаете Ctrl + Shift + P, чтобы повторно отправить весь блок в консоль.

Есть еще одна хрестоматийная иллюстрация к применению изложенного метода. Рассмотрим среднюю эффективность бейсбольных игроков относительно количества подач, когда они находятся на базе. Воспользуемся данными из пакета Lahman для вычисления среднего показателя эффективности (количество попаданий / количество попыток) каждого ведущего игрока бейсбольной лиги.



*Историческая справка.* Не сказать чтобы бейсбол был нашей национальной забавой, но многие в России знают «сколько пинчеров на базе», и именно россиянин, уроженец Нижнего Тагила, Виктор Константинович

Старухин стал первым питчером, кто одержал 300 побед в японской бейсбольной лиге, являясь одним из лучших бейсболистов мира в 20 веке.

Когда строится график визуализирующий уровень мастерства игроков, измеряется среднее значение эффективных попаданий по мячу по отношению к общему количеству предпринятых попыток, возникает две статистические предпосылки:

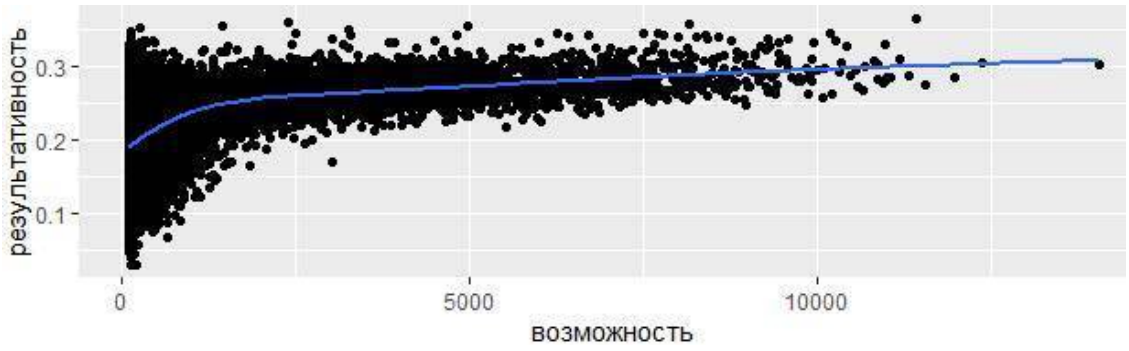
1. Как было в примере с самолётами, вариативность показателей уменьшается при увеличении количества наблюдений.

2. Существует положительная корреляция между результативностью и элементарно предоставляемой возможностью бить по мячу. Дело в том, что команды контролируют свой состав, поэтому очевидно, что на поле выходят только лучшие игроки из лучших.

Предварительно преобразуем сведения об ударах игроков в табличную форму, так они легче воспринимаются:

```
удары <– as_tibble(Lahman::Batting)
эффективность <– удары %>%
group_by(playerID) %>%
summarise(
результативность = sum(H, na.rm = TRUE) / sum(AB, na.rm = TRUE),
возможность = sum(AB, na.rm = TRUE)
)
эффективность %>%
filter(возможность > 100) %>%
ggplot(mapping = aes(x = возможность, y = результативность)) +
geom_point() +
geom_smooth(se = FALSE)
```

Функция geom_smooth() здесь формирует график методом обобщенных аддитивных моделей с интегрированной оценкой гладкости (method = "gam") рассчитывая значения по формуле formula = y ~ s(x, bs = "cs"), так как имеется более 1 000 наблюдений.




Особый интерес вызывает ранжирование результатов. Если наивно отсортировать показатели эффективности по убыванию результативности, то первыми с самой лучшей результативностью окажутся скорее везучие, а не квалифицированные игроки, за всю карьеру сделавшие лишь 1 удар, но при этом попавшие по мячу:

```
эффективность %>%
arrange(desc(результативность))
```

Можно найти хорошее объяснение этого парадокса в пословице «новичкам везёт». Используя простые инструменты, подсчет количества одинаковых значений, их суммирование, можно долго искать любопытные закономерности, но R предоставляет и много других полезных функций для генерации статистических отчетов:

Выше использовалась функция, вычисляющая среднее значение mean(*x*), но вычисляющая медианное значение функция median(*x*) тоже бывает полезна. Ведь среднее как 36.6° по больнице, а медиана – это величина, относительно которой 50% значений *x* находится выше, и 50% находится ниже, что гораздо информативнее. Иногда полезно комбинировать подобные функции с логическим условием. Мы еще не говорили о таких вещах как подмножество значений, этому можно посвятить целый раздел, пока лишь приведем наглядный пример, на тех же неотмененных авиарейсах, сгруппированных по дате вылета.

Отрицательные значения «задержки» рейса символизируют прибытие с опережением графика, оказывается, такое тоже бывает:

```
неотмененные %>% group_by(year, month, day) %>%
summarise(
средняя_задержка = mean(arr_delay),
средняя_положительная_задержка = mean(arr_delay[arr_delay > 0])
)
```

Особый интерес вызывают функции вычисления стандартного отклонения sd(x), меры разброса наблюдаемой величины, вычисления интерквартильного размаха IQR(x) и вычисления медианы абсолютного отклонения mad(x), которые являются надежными эквивалентами друг друга и могут быть полезны, если у данных есть выбросы. Любопытно, почему расстояние до одних пунктов назначения варьируются сильнее, чем до других, являя собой не иначе как чудеса телепортации:

```
неотмененные %>% group_by(dest) %>%
summarise(среднеквадратическое_отклонение_дистанции = sd(distance)) %>%
arrange(desc(среднеквадратическое_отклонение_дистанции))
```

Функции поиска минимального значение min(x), первого квантиля quantile(x, 0.25), вычисления максимума max(x), неизменные спутники при построении ранжирования. Квантили являются обобщением медианы. Так, например, quantile(x, 0.25) найдет значение *x*, которое больше чем 25% значений из всех возможных значений анализируемой переменной, и меньше чем остальные 75%.

Найдем время отправления первого и последнего рейса каждый день:

```
неотмененные %>% group_by (year, month, day) %>%
summarise( первый_рейс = min (dep_time),
последний_рейс = max (dep_time) )
```

Измерение позиции указателя на элементах списка осуществляется функциями first(*x*) для выбора первого элемента переменной *x*, nth(*x*, *n*) для выбора *n*-ного, last(*x*) для выбора последнего. Они работают аналогично адресации массивов в нотации *x*[1], *x*[*n*] и *x*[length(*x*)], но возвращают значение аргумента default, если запрошенная позиция не существует. Например, не увенчается успехом попытка получить значение такого элемента, как неотмененные$dep_time[length(неотмененные$dep_time)+1], вернув NA, неопределенное значение переменной, но при этом на выходе даст «Бинго!» вызов nth(неотмененные$dep_time,length(неотмененные$dep_time)+1, default = "Бинго!").

Следующая функция range() дополняет фильтрацию. Приведём пример, в котором сначала все записи группируются по датам и ранжируются, а потом фильтрация оставляет в строках значения, имеющие наибольший и наименьший из рангов в группе. Для сравнения, вызов функции range(неотмененные$dep_time) вернёт список, состоящий из наибольшего и наименьшего значений переменной dep_time:

```
неотмененные %>% group_by (year, month, day) %>%
mutate(ранжирование = min_rank(desc(dep_time))) %>%
filter(ранжирование %in% range(ранжирование) )
```

Ранее в вычислениях уже использовалась функция n(), которая вызывается без аргументов, и возвращает размер текущей группы. Чтобы посчитать количество непустых значений в группе *x*, используется конструкция sum(!is.na(*x*)), а чтобы подсчитать число различных (уникальных) значений вызывается n_distinct(*x*). Например, вычислим, какие направления имеют наибольшее количество перевозчиков:

```
неотмененные %>% group_by(dest) %>%
summarise(перевозчики= n_distinct(carrier)) %>%
arrange(desc(перевозчики))
```

Подсчеты значений настолько востребованы, что в пакете dplyr выделена отдельная функция count() для этого. Подсчитаем число повторений каждого направления, хранящихся в переменной dest таблицы неотмененных авиарейсов:

```
неотмененные %>% count(dest)
```

При необходимости указывается параметр веса каждого слагаемого (wt). Например, это можно использовать для подсчета общей суммы количества миль, которые пролетел самолет с фиксированным бортовым номером, взятым из поля talinum в базе неотмененных рейсов:

```
неотмененные %>% count(tailnum, wt = distance)
```

Подсчет числа значений удовлетворяющих логическому выражению, sum($x > 777$), или их среднее количество, mean($y == 0$), предполагает, что в связке с числовыми функциями TRUE преобразуется в 1, а FALSE в 0. Это делает функции sum() и mean() очень востребованными: sum(*x*) возвращает количество значений TRUE для аргумента *x*, а mean(*x*) возвращает их долю. Вычислим, сколько неотмененных рейсов было до 6 утра по данным за каждые сутки, это обычно указывает на задержку с предыдущего дня:

```
неотмененные %>% group_by(year, month, day) %>%
summarise(утренние_рейсы = sum(dep_time < 600))
```

Какова доля неотмененных рейсов, задержавшихся более часа:

```
неотмененные %>% group_by(year, month, day) %>%
summarise(часовая_задержка = mean(arr_delay >= 61))
```

При группировании по нескольким переменным, каждая новая сводка выносится на новый уровень группировки. Это облегчает восприятие и постепенно упрощает данные:

```
группы_по_дням <– group_by(flights, year, month, day)
(сводка_по_дням <– summarise(группы_по_дням, полётов = n()))
(сводка_по_месяцам <– summarise(сводка_по_дням, полётов = sum(полётов)))
(сводка_за_год <– summarise(сводка_по_месяцам, полётов = sum(полётов)))
```

Будьте осторожны при постепенном сворачивании выборки, это приемлемо для итоговых сумм и счетчиков, но нужно не забывать про такие характеристики, как медиана и отклонение, анализ результатов свёртки принципиально невозможен в ранговой статистике. Другими словами, сумма внутригрупповых сумм является общей суммой, но медиана внутригрупповых медиан не будет равна общей медиане, о последнем свойстве порой сознательно забывают при выведении нужных результатов из различных голосований. Если потребуется отменить группировку, и вернуться к операции с негруппированными данными, то используется функция ungroup():

```
группы_по_дням %>%
ungroup() %>% # разгруппируем данные обратно
summarise(полётов = n()) # подсчитаем все полёты
```

**Упражнения**

1. Примените мозговой штурм чтобы изобрести как минимум 7 различных способов анализа типовых причин серийной задержки рейсов, учитывая следующие сценарии:

а) в 50% случаев вылет осуществляется на 15 минут раньше запланированного, и в 50% рейс задерживается на 15 минут.

б) рейс всегда опаздывает на 10 минут.

в) 50% рейсов вылетает на 30 минут раньше, и 50% на 30 минут опаздывает.

г) в 99% случаев рейс выполняется точно по графику, а в оставшемся 1% происходит опоздание на 2 часа.

Что более важно для пассажира, задержка прибытия или задержка вылета, а для работы аэропортов?

2. Придумайте альтернативный способ решения той же задачи, что и в примерах неотмененные %>% count (dest), неотмененные %>% count(tailnum, wt = distance), но без использования функции count().

3. Следующее определение отмененных рейсов не оптимально:

отмененные <– flights %>%

filter( is.na(dep_delay) | is.na(arr_delay) )

Почему? Какая колонка важнее в этом случае: задержка времени вылета (dep_delay) или задержка времени прилёта (arr_delay)?

4. Посмотрите на ежедневное количество отмененных рейсов. Есть ли здесь закономерность? Связана ли доля отмененных рейсов со средней задержкой?

5. Какой перевозчик (carrier) имеет худшую статистику по задержкам рейсов? Можно ли обнаружить зависимость плохих статистических показателей аэропортов от качества работы перевозчиков? Если да, то как, если нет, то почему?

6. За что отвечает аргумент выбора способа сортировки sort в функции count()? Когда уместно его использование?

Группирование данных бывает полезным в сочетании с функцией подведения итогов summarise(), но есть удобные шаблоны и для операций преобразования mutate() с фильтрацией filter(). Вспомним про укороченный_вариант_таблицы <– select(flights, year:day, ends_with("delay"), distance, air_time), хранящий лишь сведения об опозданиях, и выделим по 5 злостных нарушителей регламента полётов на каждый день:

укороченный_вариант_таблицы %>% group_by(year, month, day) %>%

filter(rank(desc(arr_delay)) <= 5)

Сгруппируем рейсы по направлениям и оставим лишь такие группы, объем которых превышает некоторое пороговое значение:

( популярные_направления <– flights %>% group_by(dest) %>%

filter(n() > 17282) )

Преобразуем их для дальнейшего вычисления метрических характеристик внутри каждой из полученных групп:

популярные_направления %>% filter(arr_delay > 0) %>%

mutate(относительная_задержка = arr_delay / sum(arr_delay)) %>%

select(year:day, dest, arr_delay, относительная_задержка)

Как видим, группирующая фильтрация приводит к изменениям, за которыми следует обычная фильтрация. Желательно избегать подобного, за исключением поверхностного анализа данных, в противном случае бывает трудно проверить корректность выполненных манипуляций. Функции, которые наиболее естественно работают со сгруппированными данными, например mutate() и filter(), называются оконными функциями, в отличие от резюмирующих функций типа summary(). Узнать больше об оконных функциях можно вызвав соответствующий раздел справки, введя консольную команду vignette("window-functions").

**Упражнения**

1. Вернитесь к примерам использования функций mutate() и filter() со списками. Как меняется результат каждой операции при создании промежуточных групп данных?

2. Какой самолет (бортовой номер) имеет рекордно худшее время вылета?

3. В какое время суток нужно лететь, чтобы максимальное избежать задержек?

4. Для каждого пункта назначения вычислите суммарное время задержек. Для каждого рейса вычислите долю его задержек в общей сумме.

5. Задержки обычно коррелируют по времени: после того как вызвавшая первоначальную задержку проблема была решена, более поздние рейсы задерживаются, чтобы разрешить ранним покинуть аэропорт. При помощи функции lag() исследуйте, как задержка каждого рейса связана с задержкой непосредственно предшествующего.

6. Просмотрите каждый пункт назначения. Можно ли найти рейсы, долетевшие подозрительно быстро? То есть те полеты, которые потенциально представляют собой ошибку ввода данных. Вычислите долю воздушного времени полета относительно самого скоростного рейса до выбранного пункта назначения. Какие рейсы были дольше всего задержаны в воздухе?

7. Найдите все пункты назначения, в которые одновременно следовало как минимум два перевозчика. Используйте эту информацию для ранжирования перевозчиков.

8. У каждого самолета найдите количество рейсов до первой задержки более 1 часа.

# §3. Организация рабочего процесса

Теперь у читателя есть некоторый опыт выполнения кода R, без погружения в детали, но достаточный, чтобы разобраться в основах, иначе эта книга уже была бы закрыта. Разочарование естественно, когда начинаете программировать на R, потому что этот язык требователен к пунктуации, и даже один неуместно поставленный символ может заставить крепко попотеть в поисках допущенной ошибки. Но пока разочарование не зашло слишком далеко, можно утешиться тем, что описанное неудобство типично и главное – временно, такое происходит со всеми, а чтобы преодолеть его, нужно продолжать программировать на R. Прежде чем мы пойдем дальше, убедимся в наличии прочных навыков выполнения кода R, и узнаем о некоторых наиболее полезных функциях редактора RStudio. Рассмотрим некоторые базовые соглашения языка, которые до сих были пропущены в интересах скорейшего погружения в тематику. Во-первых, можно использовать R в качестве калькулятора:

```
2*2
#> [1] 4
sin(pi/2)
#> [1] 1
```

Но только в поле действительных чисел, что естественно для языка программирования статистической обработки данных и построения графиков:

```
(-1)^(1/2)
#> [1] NaN
(-1+0i)^(1/2)
#> [1] 0+1i
```

Можно создавать новые объекты с помощью оператора <-:

```
x <– 2*2
```

Все команды R, которыми создаются объекты путём присваивания, имеют одинаковую форму:

```
имя_объекта <– получаемое_значение
```

При чтении этого кода в вашей голове может прозвучать: объект «имя_объекта» получает значение «получаемое_значение». В дальнейшем понадобится решать много интересных задач, набирая большие тексты при этом. Не ленитесь, используя знак =, он тоже будет работать, но позднее приведет к путанице. Вместо этого используйте клавиатурное сокращение RStudio: Alt + – (знак минус) для быстрого набора <-. Обратите внимание, что RStudio автоматически окружает <– пробелами. Приятно читать хорошо оформленный код, поэтому недайсвоимглазамсломаться, используйте пробелы.

Имена объектов должны начинаться с буквы и могут содержат только буквы, цифры, нижнее подчеркивание «_» и точку «.», вы ведь хотите, чтобы имена объектов были информативны, поэтому понадобится соглашение для склеивания нескольких слов. Мне привычнее змеиный_стиль, в котором строчные слова разделяются нижним подчеркиванием «_».

```
НекоторыеЛюдиИспользуютВерблюжийСтиль,
третьи.Вовсе_ОТКАЗЫАВЮТСЯОтусловностей.
```

Вернемся к стилю позже, в разделе посвященном описанию функций. А пока, можно проверить содержимое ранее созданного объекта, введя его имя:

```
x
#> [1] 4
```

Выполните эксперимент, введите:

```
это_действительно_длинное_имя <– 777
```

Чтобы проверить значение этого объекта, попробуйте в RStudio средство завершения строки: введите «это» и нажмите клавишу Tab, либо Ctrl + Space (пробел), добавятся недостающие символы, так как пока этот префикс уникален, а затем нажмите клавишу Enter. А что, если это_действительно_длинное_имя должно было иметь значение 1234, а не 777. Можно использовать другое сочетание клавиш, чтобы исправить его. Введите «это» и нажмите Ctrl + ↑. Появится список всех ранее набранных команд, которые начинаются с таких букв. Воспользовавшись стрелками навигации и нажав Enter, выбранная команда наберется повторно. Тогда можно будет изменить значение параметра с 777 на 1234 и выполнить ввод.

Еще один поучительный эксперимент, вместо x введём

```
икс
#> Ошибка: объект 'икс' не найден
X
#> Ошибка: объект 'X' не найден
```

Существует негласная договоренность между пользователем и R: за вас будут делать все рутинные вычисления, но взамен, нужно быть абсолютно точным в своих инструкциях. Как видим, система чувствительна к регистру.

R имеет обширную коллекцию встроенных функций, которые вызываются так:

имя_функции(аргумент1 = значение1, аргумент2 = значение2, …)

Давайте попробуем вызвать функцию seq(), которая генерирует регулярные последовательности чисел, и на её примере узнать больше полезных особенностей RStudio. Введите se и нажмите Tab. Всплывающее окно покажет возможные завершения. Укажите seq(), введя дополнительное «q», чтобы снять двусмысленность, или используйте стрелки ↑/↓ для выбора из предложенных вариантов. Обратите внимание на всплывающую подсказку, в ней перечислены аргументы функции и их назначение. Если хотите открыть справку, то нажмите клавишу F1, чтобы получить все подробности на соответствующей вкладе в правой нижней части панели управления. Нажмите клавишу Tab еще раз, когда выбрали функция, которую хотите вызвать. RStudio добавит соответствующие открывающиеся «(» и закрывающиеся «)» скобки. Введите аргументы 1, 5 и нажмите Enter.

```
seq(1, 5)
#> [1] 1 2 3 4 5
```

Введите следующий код и обнаружите, что RStudio также помогает с кавычками:

```
x <– "привет мир!"
```

Дело в том, что кавычки и круглые скобки всегда должны следовать в паре. RStudio делает всё возможное, чтобы помочь с их расстановкой. Если произойдет какое-либо несоответствие, то R подскажет:

```
> x >
+
```

Символ «+» говорит о том, что консоль R ожидает продолжения ввода. Обычно это означает, что забыли закрыть кавычки «"», либо скобки «)». Добавьте недостающие символы, либо нажмите клавишу Esc, чтобы начать ввод заново.

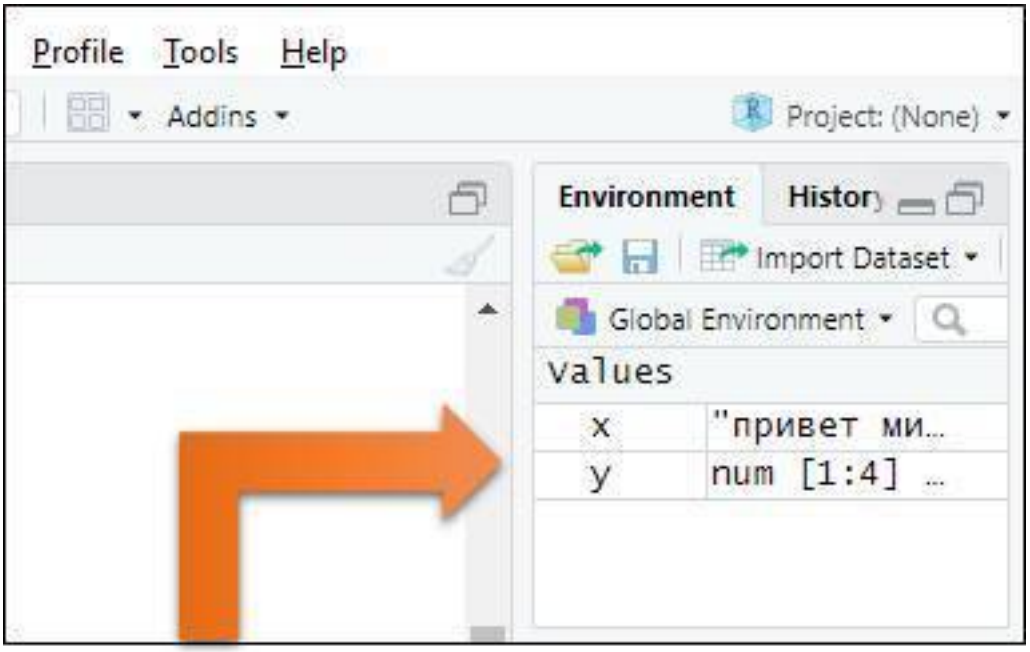
При решении задач не терпится узнать результат вычислений, поэтому вводят имя объекта, чтобы увидеть его содержимое:

```
y <– seq(1, 5, length.out = 4)
y
#> [1] 1.000000 2.333333 3.666667 5.000000
```

Это частое действие сокращается путём заключения команды в круглые скобки:

```
(y <– seq(1, 5, length.out = 4))
#> [1] 1.000000 2.333333 3.666667 5.000000
```

Теперь посмотрим на панель в правом верхнем углу окна RStudio:




Здесь можно увидеть все объекты, которые были созданы ранее.

**Упражнения**

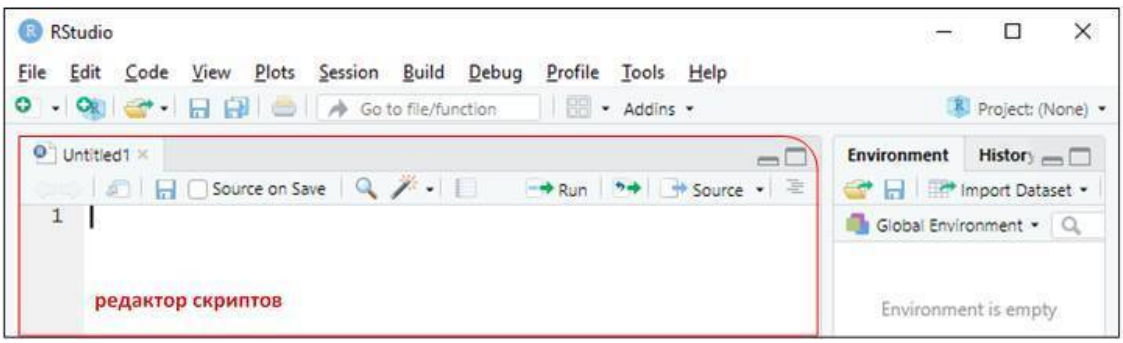
1. Скопируйте и вставьте в консоль:

```
label <– 1
label
# > Ошибка: объект 'label' не найден
```

Почему этот код не работает? Смотрите внимательно! Может показаться бессмысленным, но разница в написании а-латиницей и а-кириллицей для R существенна.

2. Нажмите Alt + Shift + K (латинское). Что произошло? Как добраться до того же экрана из главного меню приложения?

До сих пор использовали консоль для запуска код. Это неплохо для начала, но восприятие его оказывается невозможным довольно быстро, как только создаются более сложные конструкции ggplot2 при построении графиков с применением каналов dplyr. Чтобы использовать больше рабочего пространства, нужно открыть редактор скриптов. Откройте его, выбрав пункт меню «File», а затем «New File» и «R Script», либо с помощью сочетания клавиш Ctrl + Shift + N. Появится новая панель:




Редактор отлично подходит для просмотра кода и его набора. Можно экспериментировать в консоли, но как только написан код, который работает и делает то, что задумывалось,

поместите его в редактор скриптов. RStudio автоматически сохранит содержимое редактора при выходе и будет автоматически загружать его при следующем входе. Тем не менее, чтобы не потерять набранное, лучше сохранять скрипты регулярно.

В редакторе скриптов отлично проходит сборка и отладка сложных графиков ggplot2 из длинных последовательностей команд dplyr. Ключом к эффективному использованию любого редактора является запоминание горячих клавиш и клавиатурных сокращений. Наиболее востребованным сочетанием клавиш редактора скриптов RStudio является Ctrl + Enter, оно выполняет текущее выражение R в консоли. Например, возьмем любой код из предыдущей главы. Если курсор находится внутри некоторой команды, то нажатие Ctrl + Enter запустит команду целиком и переместит курсор к началу следующей команды. Эта возможность используется для пошагового выполнения сценариев, путём многократного нажатия Ctrl + Enter.

Вместо того, чтобы запускать сценарий шаг за шагом, можно выполнить сохраненный сценарий целиком, нажав Ctrl + Shift + S. Рекомендуется делать это регулярно, чтобы проверить, все ли важные части кода сохранены. Начинать свой код желательно с команд подключения необходимых для запуска пакетов. Таким образом, если поделитесь кодом с другими исследователями, то они смогут легко увидеть, какие пакеты предстоит установить при необходимости. Однако никогда не нужно включать команды принудительной установки пакетов install.packages() или изменения рабочего каталога setwd() в распространяемом скрипте, признаком дурного тона является изменение настроек на чужом компьютере.

Работая с фрагментами кода в последующих главах настоятельно рекомендуется открывать их редактором и использовать различные клавиатурные сокращения. Со временем отправка кода на исполнение в консоль станет настолько естественной, что даже не будете думать об этом.

Редактор скриптов хорош тем, что выделяет синтаксические ошибки красной волнистой линией и крестиком на боковой панели. Наведите указатель мыши на крестик, чтобы увидеть, в чем проблема. RStudio укажет и потенциальные ошибки в программе.

**Упражнения**

1. Найти один совет, который выглядит интересный. Практикуйтесь в его использовании.
2. Какие распространенные ошибки показывает RStudio при диагностике кода?

В завершении главы скажем немного слов об организации и структуре хранения данных для крупных проектов. Порой приходится переключаться с R на сопутствующие задачи, возвращаясь к анализу данных на следующий день. Или вовсе работать над несколькими задачами одновременно, каждая из которых задействует R независимо друг от друга. Наступит момент, когда в R нужно будет загрузить данные с внешних источников и отправить вычисленные результаты обратно на внешние носители. Чтобы справиться с этими жизненными ситуациями, нужно ответить для себя на два вопроса:

1. Что является результатом выполненного анализа, то есть что будет сохранено в итоговом отчете о проделанной работе?
2. Где это сохранить?

Для начинающего пользователя R нормально рассматривать окно RStudio как основную рабочую область R, в которой хранятся все необходимые данные и строки кода. Однако в конечном счете гораздо лучше основными считать R-скрипты. С помощью R-скриптов и файлов исходных данных всегда можно восстановить рабочее окружение, но намного сложнее восстановить не сохранившиеся R-скрипты. Для этого либо придется перепечатать код заново, допуская ошибки при наборе, либо тщательно перебирать записи в истории консоли R. Если настроить RStudio так, чтобы рабочее пространство не сохранялось между сеансами, это причинит некоторое кратковременное неудобство, потому что теперь, когда перезапускается RStudio, не будет открываться код, который запустился в прошлый раз. Но это спасёт от мучений в будущем, так как заставляет все важные манипуляции с данными прописывать в

коде и сохранять отдельно. Нет ничего хуже, чем обнаружить через несколько месяцев сохранившиеся лишь результаты важного расчета в рабочем окне RStudio, а не сам расчет.

Существует пара клавиатурных сокращений, используемых чтобы убедиться в сохранности важной части кода в редакторе RStudio:

1. Нажмите Ctrl + Shift + F10, чтобы перезагрузить сеанс RStudio.

2. Нажмите Ctrl + Shift + S, чтобы повторно запустить текущую команду скрипта.

Для R есть понятие «рабочий каталог». Именно в рабочем каталоге ищутся файлы при загрузке, сохраняются по умолчанию файлы, которые отправляют на сохранение. RStudio показывает текущий рабочий каталог в верхней части окна консоли. Можно увидеть путь к рабочему каталогу отдельно, запустив в коде R функцию getwd(). Начинающие пользователи R выбирают в качестве рабочего каталог рабочего стола, каталог документов, или любой другой странный каталог на компьютере. Но очень скоро эволюционируют в плане организации расположения проектов по каталогам и при работе над крупным проектом меняют рабочий каталог R на более подходящий. Можно сменить рабочий каталог из кода R запустив команду setwd("/путь/до/нового/рабочего/каталога"). Но никогда не делайте так, это не тот способ, которым пользуются профессионалы R для настройки пути до рабочего каталога. Адресация путей и каталогов осложнена тем, что в операционных системах Mac, Linux и Windows применяются разные форматы для их записи. Есть три основных отличия:

1) Самое главное отличие заключается в разделителях компонентов путей. Mac и Linux используют косую черту «/», а Windows использует обратную косую черту «\». R может работать с любым типом разделителя, независимо от используемой платформы, но обратные косые черты зарезервированы для ввода управляющих символов, в частности, чтобы получить одну обратную косую черту, в строке пути нужно вводить две обратные косые черты. Поэтому рекомендуется всегда использовать стиль Linux/Mac и разделять фрагменты пути символом «/».

2) Абсолютные пути, то есть пути, которые указывают на фиксированное место, независимо от рабочего каталога, выглядят по-разному. В Windows они начинаются с буквы диска, например «C:», или с двух обратных косых черт, например, «\\имя_сервера», а в Mac/Linux они начинаются с косой черты «/», например, «/users/RStudio». Никогда не используйте абсолютные пути в скриптах, потому что они мешают обмену результатами исследований с другими разработчиками, ведь ни у кого нет в точности совпадающий структур каталогов.

3) Последнее отличие – ссылки на домашний каталог. Символ «~» является удобным обозначением для домашнего каталога, но в Windows на самом деле нет понятия «домашний каталог», поэтому «~» указывает на папку документов текущего пользователя.

Специалисты хранят все файлы, связанные общими целями и задачами (входные данные, R скрипты, аналитические результаты, рисунки) в одном каталоге. Эта житейская мудрость является распространенной практикой, так как RStudio имеет встроенную поддержку для её реализации посредством создания проектов.

Для создания нового проекта, который будет использоваться при работаете с оставшейся частью книги, достаточно выбрать в главном меню приложения пункт «File», а затем «New Project…». Назовите проект «практикум по статистике» и хорошенько подумайте, в какой каталог его поместить. Если не сделаете это сознательно, то потом очень трудно будет его найти. Как только создание проекта завершено, проверьте, что путь до каталога проекта совпадает с рабочим каталогом. Всякий раз, когда в коде ссылаются на файл через относительный путь, он будет искаться в рабочем каталоге.

Теперь введите в редакторе скрипа следующие команды

```
library(tidyverse)
ggplot(diamonds, aes(carat, price)) +
geom_hex()
ggsave("алмазы.pdf")
```

```
write_csv(diamonds, "алмазы.csv")
```

И сохраните файл, назвав его «алмазы.R». Далее, запустите этот скрипт, чтобы создать файлы PDF и CSV в каталоге проекта. Не беспокойтесь о деталях, их разберем подробнее чуть позже. А пока, выйдите из RStudio и откройте папку проекта, обнаружите там файл «практикум по статистике» с расширением «.Rproj». Дважды кликните по нему для повторного открытия проекта. Заметьте, что вернулись на место, где остановились, это тот же самый рабочий каталог и история команд, открылись все файлы, над которыми работали. Следуя описанным инструкциям имейте ввиду, что начинать новый проект лучше с пустого системного окружения, как чистого листа.

Если теперь выполнить поиск файл алмазы.pdf, то найдется PDF рядом со скриптом, который его создал (алмазы.R). Одновременно сохранились график и данные, по которым он строился. Предпочтительно сохранять данные кодом R, а не с помощью мыши или через буфер обмена, чтобы не исказить информацию.

Проекты RStudio формируют крепкую основу рабочего процесса. Для повышения эффективности стоит придерживаться следующих рекомендаций:

- Создавать отдельный проект RStudio для каждого аналитического проекта.
- Хранить файлы данных в папке проекта, для удобной загрузки их в R.
- Храните там же и скрипты, редактируя их, запуская по частям или целиком.
- Сохранять там же и выходных данных (графики, очищенные данные).
- Использовать только относительные пути, а не абсолютные.

В результате, всё необходимое для работы будет находится в одном месте, изолированном от других проектов.

# §4. Статистический анализ данных

Эта глава посвящена освоению основных приёмов статистического анализа информации, полученной средствами визуализации и преобразований, при систематическом изучении педагогических данных. Основная задача отдельной дисциплины, называемой «исследовательский анализ данных», заключается в открытии новых характеристик данных, и решается неоднократным повторением следующих трех шагов:

1) Сформулируйте вопросы о ваших данных.

2) Ищите ответы с помощью визуализации, преобразований и моделирования.

3) Используйте обнаруженные закономерности, чтобы уточнить имеющиеся вопросы и сформулировать новые.

Описанное не является формальным процессом со строгим набором правил, это скорее «состояние ума». Во время первого этапа нужно чувствовать себя свободно, чтобы исследовать каждую идею, что приходить на ум. Некоторые из идей будут реализованы, другие заведут в тупик, но поскольку исследование продолжится, то можно будет сконцентрироваться на нескольких особо продуктивных направлениях, которые в конечном итоге разовьются при общении с другими людьми.

Визуализация и преобразования являются важной частью любого анализа данных, даже если данные представлены «на блюдечке с голубой каёмочкой», всегда нужно исследовать качество исходных данных. Предварительная подготовка является одним из ключевых этапов. Задайте вопросы о том, соответствуют ли имеющиеся данные ожидаемым или нет. Чтобы выполнить грамотную очистку данных, будут использованы все доступные инструменты: визуализация, преобразование и моделирование.

Опыт использования пакетов dplyr и ggplot2 в интерактивном режиме для генерации вопросов, поиска ответов, с последующей формулировкой новых вопросов, показывает, что всегда нужно искать хотя бы примерный ответ на один принципиальный вопрос, чем погрузиться в поиски точных ответов на несколько риторических. Основная цель второго этапа состоит лишь в том, чтобы пришло понимание исходных данных. Самый простой способ достижения этого – использовать вопросы как инструменты для руководства к действиям. Когда спрашиваете, вопрос фокусирует внимание на определенной части набора данных и помогает решить, какие графики, модели или преобразования предстоят. Как любое частное приложение ТРИЗ (теории решения изобретательских задач) это в основном творческий процесс. И как у большинства творческих процессов, ключ к тому, чтобы задавать качественные вопросы, заключается в генерации большого количества вопросов. В любых открытых системах закон перехода количества в качество неизбежно проявляется также, как закон единства и борьбы противоположностей. Да, трудно формулировать вопросы в начале исследования, но лишь до тех пор, пока не станет известно, какая информация содержится в анализируемом наборе данных. С другой стороны, каждый новый вопрос, который задается, откроет новый фрагмент мозаики и увеличить шанс на внеочередное научное открытие. Можно детализировать наиболее интересные моменты и формулировать вопросы, наводящие на размышления, продолжая каждый вопрос новым вопросом, основанным на уточненной информации.

Не существует общего правила, какие вопросы нужно задать, чтобы продвинуться в исследовании. Тем не менее, два типа вопросов всегда полезны для совершения открытия:

1) Какова вариативность значений внутри выборки?

2) Какова ковариация между различными выборками?

Ниже рассмотрим эти два вопроса. Будет объяснено, что такое вариация и ковариация, и показано несколько способов ответа на каждый вопрос. Чтобы обсуждение сделать плодотворным, определимся с терминами:

Переменная – это количество, качество или свойство, которое можно измерить.

Значение – это состояние переменной, полученное в процессе измерения. Значение переменной может изменяться между измерениями.

Наблюдение – это набор измерений, сделанных в аналогичных условиях. Обычно все измерения наблюдений делаются в одно время на одном объекте. Наблюдение может содержать несколько значений, каждое из которых связано с разными переменными, поэтому наблюдение порой считают точкой многомерного пространства данных.

Табличные данные представляют собой набор значений, каждое из которых ассоциируется с переменной и наблюдением. Табличные данные «аккуратны» если каждое значение помещается в отдельной ячейке, а каждая переменная в своей собственной колонке, каждое наблюдение в своей собственной строке.

До сих пор все данные, которые видели, были аккуратны, но в реальной жизни большинство данных не являются аккуратными, достоверными, точными, верными, значащими, поэтому будем возвращаться к идее предварительной очистки снова и снова.

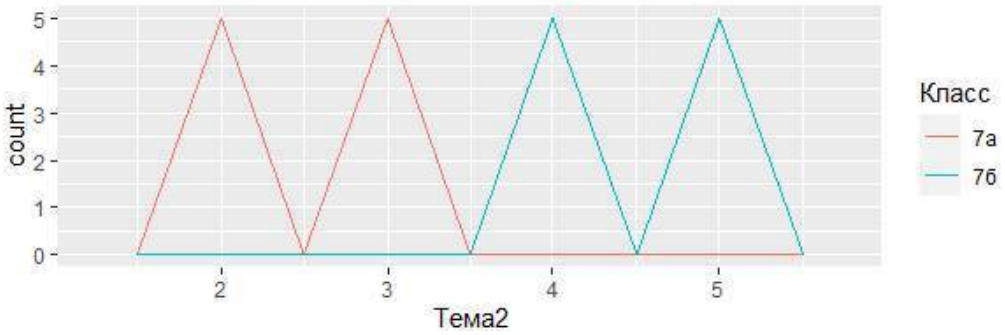
Вариативность данных представляет собой тенденцию в изменениях значений переменной при её изменении от одного измерения к другому. Можно легко наблюдать вариативность данных в реальной жизни. Если измерить любую непрерывную переменную дважды, то получатся два разных результата, даже если измерять величины, которые постоянны, например скорость света. Каждый раз в измерение войдет небольшое количество погрешностей, варьирующихся от измерения к измерению. Категориальные переменные также могут меняться если их измерять на разных предметах (например, цвет глаз у разных людей), или в разное время (например, энергетические уровни электрона в разные моменты времени). Каждая переменная имеет свой диапазон вариации, который помогает извлечь интересную информацию. Самый лучший путь к пониманию вариативности заключается в визуализации распределения значений переменной.

Как именно визуализировать распределение переменной зависит от того, является ли переменная категориальной или непрерывной. Переменная называется категориальной, если она может принимать только одно значение из небольшого набора. В R категориальные переменные обычно сохраняются как факторы или вектора символов. Обычно распределение категориальной переменной демонстрируется с помощью гистограмм, высота прямоугольников которых показывает, сколько наблюдений имело то или иное значение переменной. Переменная является непрерывной, если она может принимать любое значение из потенциально бесконечного множества упорядоченных величин. Действительные числа и время в этом смысле являются примерами непрерывных переменных. Изучить распределение непрерывной переменной тоже можно используя гистограмму, если предварительно разбить данные на непересекающиеся интервалы. Дело в том, что гистограмма поделит ось $x$ на равные промежутки, а затем вычислит высоту прямоугольника для представления числа наблюдений, которые попадают в каждый из отдельных промежутков. Можно установить фиксированную ширину интервалов гистограммы аргументом binwidth, который измеряется в единицах измерения переменной $x$. Всегда стоит проверить несколько разных значений параметра binwidths при работе с гистограммами, так как разная ширина прямоугольников поможет выявить закономерности. Если необходимо наложить несколько гистограмм на один график, то рекомендуется использовать функцию geom_freqpoly() вместо geom_histogram(), так как geom_freqpoly() выполняет тот же подсчет повторений, что и функция geom_histogram(), но вместо прямоугольников для отображения результата вычислений использует линии. Гораздо проще воспринять информацию, когда перекрываются линии разных цветов, а не прямоугольники.

Проиллюстрируем сказанное на примере визуализации сведений об успеваемости по Теме 2 из приведенной в Главе 2 базы данных:

```
ggplot(data = filter(My_table, Класс %in% c("7а","7б")),
```

```
mapping = aes(x = Тема2, colour = Класс)) +
geom_freqpoly(binwidth = 0.5)
```




Есть несколько проблем, связанных с этим методом, но к ним ещё вернёмся, визуализируя категориальные и непрерывные переменные в дальнейшем. Теперь же, когда знаем, как визуализировать вариативность значений переменной, что следует искать на графиках? Какие вопросы задавать?

Ниже собран список типовых сведений, которые можно почерпнуть из графиков и диаграмм, с некоторыми последующими вопросами для каждого информационного блока. Ключом к хорошим формулировкам вопросов, следует считать любопытство (о чем хотите узнать больше?) и скептицизм (как это может ввести в заблуждение?).

# Конец ознакомительного фрагмента.