МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«СИБИРСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИНЖЕНЕРИИ И
БИОТЕХНОЛОГИЙ»

ИНСТИТУТ ЦИФРОВЫХ ТЕХНОЛОГИЙ

Кафедра прикладной биоинформатики

ОСНОВЫ РАБОТЫ С ЯЗЫКОМ ПРОГРАММИРОВАНИЯ R В КАЧЕСТВЕ ИНСТРУМЕНТА ОБРАБОТКИ ПЕРВИЧНЫХ ДАННЫХ В ЗООТЕХНИИ, БИОЛОГИИ И ПИЩЕВОЙ ПРОМЫШЛЕННОСТИ

Учебное пособие

Новосибирск, 2025

УДК 57.087.1; 519.87 (075); 636
ББК 28.0 + 40 : 87.256.631, Я 73

АВТОРЫ:

Шатохин Кирилл Сергеевич, канд. биол. наук, доцент кафедры прикладной биоинформатики

Норкина Виолетта Михайловна, ассиситент кафедры прикладной биоинформатики

Нарожных Кирилл Николаевич, канд. биол. наук, доцент кафедры прикладной биоинформатики, доцент

Петров Алексей Фёдорович, канд. биол. наук, старший преподаватель кафедры прикладной биоинформатики

Чечушкова Марина Анатольевна, канд. биол. наук, доцент кафедры прикладной биоинформатики, доцент

Камалдинов Евгений Варисович, доктор биол. наук, заведующий кафедрой прикладной биоинформатики, доцент

РЕЦЕНЗЕНТЫ:

Алетдинова Анна Александровна, доктор экономических наук, профессор кафедры информационных технологий и моделирования ФГБОУ ВО Новосибсрский ГАУ, доцент;

Казанцев Федор Владимирович, кандидат биологических наук, старший научный сотрудник сектора компьютерного анализа и моделирования биологических систем ИЦиГ СО РАН;

Богомолов Антон Геннадьевич, кандидат биологических наук, научный сотрудник сектора компьютерного анализа и моделирования биологических систем ИЦиГ СО РАН

Для цитирования: Основы работы с языком программирования R в качестве инструмента обработки первичных данных в зоотехнии, биологии и пищевой промышленности: учебное пособие. К.С. Шатохин, В.М. Норкина, К.Н. Нарожных, А.Ф. Петров, М.А. Чечушкова, Е.В. Камалдинов. Новосиб. гос. аграр. ун-т. – Новосибирск, 2025. – 141 с.

В учебном пособии представлен материал, предназначенный для первичного знакомства со средой программирования R. Представлены примерные задания для выполнения контрольной работы. Пособие предназначено для студентов по направлениям подготовки (уровни бакалавриата и магистратуры): Прикладная биоинформатика, Зоотехния, Биология, Технология производства и переработки сельскохозяйственной продукции, Продукты питания животного происхождения, Технология продукции и организация общественного питания всех форм обучения. Пособие рекомендовано для освоения следующих дисциплин: Компьютеризация производства, Информационные технологии в науке, образовании и производстве, Прикладное программирование в R, Биометрический анализ на языке R, Прикладное программирование.

Оглавление

ВВЕДЕНИЕ

Современное животноводство тесно сопряжено с обработкой колоссального объёма данных. Одной из частых задач является осуществление закрепления быка-производителя за маточным поголовьем. Эта операция предполагает анализ родословной, оценку родителей по комплексу линейных признаков экстерьера (либо схожих методов) и генетического потенциала продуктивности обоих родителей, что требует как минимум задействовать базу данных имеющегося маточного поголовья хозяйства и доступных быков-производителей с последующим прогнозированием необходимых качеств потомства. Одно из крупнейших хранилищ данных, Canadian Dairy Network[1] (CDN), включает в себя записи более чем о 800 тыс. быков-производителей одной только голштинской породы.

Наиболее информативным способом является прогнозирование племенного потенциала потомства при помощи регрессионных моделей [Кузнецов, 2013; Петров и др., 2021; Mrode, Pocrnic, 2023]. Быстрая и качественная обработка таких массивов данных практически невозможна без использования специализированного программного обеспечения. Широкое распространение метода BLUP [Кузнецов, 1995; Cole, VanRaden, 2018; Meher, Rustgi, Kumar, 2022; Mrode, Pocrnic, 2023] стало возможным только после 1990 года благодаря массовому внедрению компьютеров в процесс оценки скота в странах с развитым молочным скотоводством [Miglior, Muir, Van Doormaal, 2005]. В России наблюдается дефицит на рынке свободно распространяемых средств хранения и обработки данных первичного зоотехнического учёта. Имеющиеся средства (DairyComp 305, СЕЛЭКС и т.п.) как правило узкоспециализированы и работают только в течении срока и набора функциональных опций, предусмотренных лицензией. В производственной практике имели место случаи запроса сверхлицензионных денежных средств за дополнительное

1 https://www.cdn.ca/home.php

предоставление даже самых элементарных опций, например, распечатывание отчётов. В условиях отключения российских пользователей от программного обеспечения из недружественных стран актуальна подготовка специалистов, умеющих осуществлять анализ данных в области биологии, зоотехнии и пищевой промышленности используя свободнораспространяемое программное обеспечение.

Одним из наиболее доступных, высокопроизводительных и простых средств обработки первичных данных является язык программирования R [Barry, 2018]. Важными преимуществами языка программирования R являются:

1. Открытый исходный код, что даёт возможность совершенствовать существующие пакеты и функции, а так же разрабатывать собственные.
2. Кроссплатформенность — на официальном репозитории размещены установочные пакеты для Windows 10, 11 и macOS, а так же инструкции по установке R на ряд операционных систем Linux с использованием командной строки.
3. Доступность — R - это не лицензионный продукт, а потому не является чей-либо собственностью, что позволяет свободно ссылаться на решения, написанные на R в своих научных работах не опасаясь претензий со стороны правообладателей.

На русском языке имеется не так уж много литературы по использованию R [Мастицкий, Шитиков, 2015; Уикем, Гроссер, Буманн, 2024; Шипунов и др., 2017], которая не раскрывает частные вопросы использования R применимо к задачам зоотехнии, биологии и пищевой промышленности. Целью настоящего пособия является погружение студентов, магистрантов, аспирантов, преподавателей и научных сотрудников научных организаций и учебных ВУЗов сельскохозяйственной и биологической направленности в среду статистического программирования R. Настоящее пособие предполагает прикладной подход к изучению языка R без углубления в программирование.

Книга в первую очередь предназначена для начинающих, но некоторые частные вопросы могут быть интересны и продвинутым пользователям.

В учебном пособии приведены практические задания, для решения которых необходимо воспользоваться данными в открытом доступе. Для облегчения работы с пособием, все данные были помещены в электронное хранилище на яндекс диске[2]. Список источников данных представлен в прил. 2.

2 https://disk.yandex.ru/d/N9xkmdsklgCbWw

1. История языка программирования R. Основные репозитории. Установка и начало работы с RStudio.

1.1. Краткая история языка программирования R.

Язык программирования R был создан сотрудниками Оклендского университета в 1993 году Россом Ихакой и Робертом Джентльманом на базе другого языка программирования — S, откуда был унаследован ряд базовых функций. По одной из версий название R было выбрано потому, что с этой буквы начинаются имена его первых разработчиков. Данный программный продут изначально задумывался как средство обработки и статистического анализа исходных данных. Основными его пользователями планировались не программисты, то есть люди, занимающиеся разработкой программных продуктов, а аналитики, то есть люди работающие с данными. Основной целью разработчики ставили создать решение, позволяющее из большого массива данных получить максимум полезной информации. R является программным продуктом с открытым исходным кодом, то есть свободно распространяемым ПО [Giorgi, Ceraolo, Mercatelli, 2022; Ihaka, 2022; Singh, Garg, 2019; Venables, Ripley, 2002]. В 2009 году была разработана графическая оболочка для работы с R — RStudio. Считается, что удобство работы в RStudio стало «отправной точкой» роста популярности R как средства обработки данных[3]. Начиная с 2000-х годов R рассматривается как один из основных методов обработки данных и статистического анализа во всём мире [Martin, Quinn, Park, 2011]. В 2009 году начал издаваться специализированный научный журнал, посвящённый использованию R - *The R Journal*[4], который является преемником журнала *R News*, что издавался с 2001 по 2008 годы [Chambers, 2009]. Импакт-фактор SJR у данного журнала за 2024 год составляет 0.949 (Q1)[5]. Язык R

3https://varmara.github.io/proteomics/01_introduction_to_r.

4 https://digitalcommons.unl.edu/r-journal/

5 https://www.scimagojr.com/journalsearch.php?q=21100255423&tip=sid&exact=no

оказался единственным инструментом, способным обрабатывать данные с андронного коллайдера [Pfeiffer, Pia, 2013; Zhao, 2013].

1.2. Основные репозитории R.

Для хранения исходного кода R в 1997 году Куртом Хорником и Фрицем Лейшем был создан специальный репозиторий - Comprehensive R Archive Network (CRAN; https://cran.r-project.org/). Там же размещена сопроводительная документация для дополниельных пакетов [Hornik, 2012]. По состоянию на 08.08.2024 на официальном репозитории CRAN размещено более 21 тыс. дополнительных библиотек.

Существуют и другие популярные репозитории. В частности, под началом Роберта Джентльмена в 2001 году был заложен проект Bioconductor (https://www.bioconductor.org/) [Gentleman и др., 2004], специализированный на хранении дополнительных пакетов языка программирования R, ориентированных на работу с данными в области биоинформатики, включая операции с данными о геноме, протеоме и транскриптоме [Gauthier и др., 2019; Lawrence и др., 2013; Sepulveda, 2020; Stanstrup и др., 2019].

GitHub (https://github.com). Был запущен в 2008 году командой разработчиков (Крис Ванстрас, Пи Джей Хайетт и Том Престон-Вернер) из Лос-Анджелеса. С 2008 по 2021 год количество зафиксированных ссылок на использование продуктами проекта GitHub увеличилось со 16 до 76746 или почти в 480 раз [Escamilla и др., 2022]. Основная цель GitHub — обеспечить удобную и мощную платформу для совместной разработки программного обеспечения. Он решает несколько ключевых проблем разработчиков:

1. Хранение кода: Ваш код не пропадет, если сломается компьютер.
2. Версионность: Вы можете видеть всю историю изменений, возвращаться к старым версиям и понимать, кто, что и когда поменял.

3. Совместная работа: Несколько человек могут работать над одним проектом, не мешая друг другу.
4. Демонстрация и портфолио: Для многих разработчиков GitHub — это их профессиональное резюме.

   В условиях отключения России от международных репозиториев актуально будет приведение ссылки на зеркало R в Новосибирске:

   https://mirror.truenetwork.ru/CRAN/

1.3. Особенности проекта GitHub

Представьте, что над одним сложным документом — например, научной статьей или техническим проектом — работает целая команда авторов. Если каждый будет редактировать один файл, постоянно пересылая его по почте, рано или поздно версии перепутаются, а нужное изменение будет невозможно найти. Именно эту проблему решает GitHub — платформа, которая стала стандартом для совместной разработки программного обеспечения. По своей сути, это мощный инструмент управления версиями, объединенный с социальной сетью для миллионов разработчиков [Cosentino, Cánovas Izquierdo, Cabot, 2017].

Система контроля версий. В основе GitHub лежит система Git, функционирование котрой построено на так называемых репозиториях. Репозиторий — это полноценный «архив проекта», который хранит не только текущее состояние кода, но и всю историю его изменений: кто, когда и какую строку исправил. Если новая функция нарушила работу программы, разработчик всегда может совершить «откат» назад к стабильной версии [Jiang и др., 2017].

Коммиты — это ключевые моменты в истории проекта. Каждый коммит представляет собой законченный набор изменений, снабженный комментарием

разработчика (например, «исправлена ошибка расчета»). Коммиты можно сравнить с контрольными точками в видеоигре — сохранившись, вы всегда можете к ним вернуться [Gousios, Pinzger, Deursen, 2014].

Ветвление и слияние — механизм, который позволяет вести параллельную работу. Главная, стабильная версия проекта хранится в ветке `main`. Чтобы добавить новую функцию или исправить ошибку, разработчик создает свою собственную ветку — изолированную «песочницу». В ней можно экспериментировать, не рискуя нарушить работу основного кода. Когда работа завершена и протестирована, изменения из этой ветки «сливаются» обратно в *`main`*. Этот подход делает процесс разработки предсказуемым и безопасным [Zou и др., 2019].

Механизмы совместной работы. GitHub вышел за рамки простого хранения кода, став платформой для глобальной коллаборации. Это особенно важно для открытого программного обеспечения, где тысячи незнакомых друг с другом разработчиков со всего мира могут сообща работать над одним проектом [Zou и др., 2019] [Ren, Zhou, Kästner, 2018].

Автоматизация и управление проектами. GitHub Actions — это «виртуальный помощник» для разработчика. Он позволяет автоматизировать практически любой процесс: запускать тесты при каждом новом изменении, автоматически собирать приложение или развертывать его на сервере. GitHub имеетв своём арсенале гибкие канбан-доски для визуального отслеживания задач, похожие на Trello. Так же имеется встроенная систеа для ведения документации, что позволяет хранить всю справочную информацию прямо в репозитории [Zou и др., 2019].

1.4. Начало работы с R. Интерфейс RStudio.

Перед началом работы необходимо установить R на ваш ПК либо ноутбук. Если у вас операционная система Windows, то нужно просто скачать клиент с

официального репозитория[6], а затем установить как стандартное декстопное приложение. Если у вас Linux, то R рекомендуется устанавливать при помощи командной строки, набор команд будет зависеть от конкретной версии операционной системы.

После установки языка программирования R в системе Windows пользователю будет предоставлена стандартная оболочка, в Linux программирование можно осуществлять в терминале. Оба способа являются не очень удобными. В частности, автоматический вывод доступных функций и других объектов при вводе команд недоступен, так что пользователю приходится вручную полностью их вводить, что учащает количество ошибок. Наиболее комфортной средой для работы с языком программирования R представляется RStudio. Существует мнение, что рост популярности R связан с простой работы в RStudio[7].

С установкой RStudio у пользователей Windows 10/11 не должно возникнуть проблем, так как клиент для установки доступен для бесплатного скачивания с официального репозитория[8]. Для пользователей Windows 8 и более ранних версий клиент для установки RStudio придётся искать на сторонних ресурсах. Для пользователей Linux рекомендуется устанавливать RStudio при помощи набора команд. После установки и запуска RStudio пользователю откроется рабочее пространство Рисунок 1

6https://cran.r-project.org/

7https://varmara.github.io/proteomics/01_introduction_to_r.

8https://posit.co/download/rstudio-desktop/#download

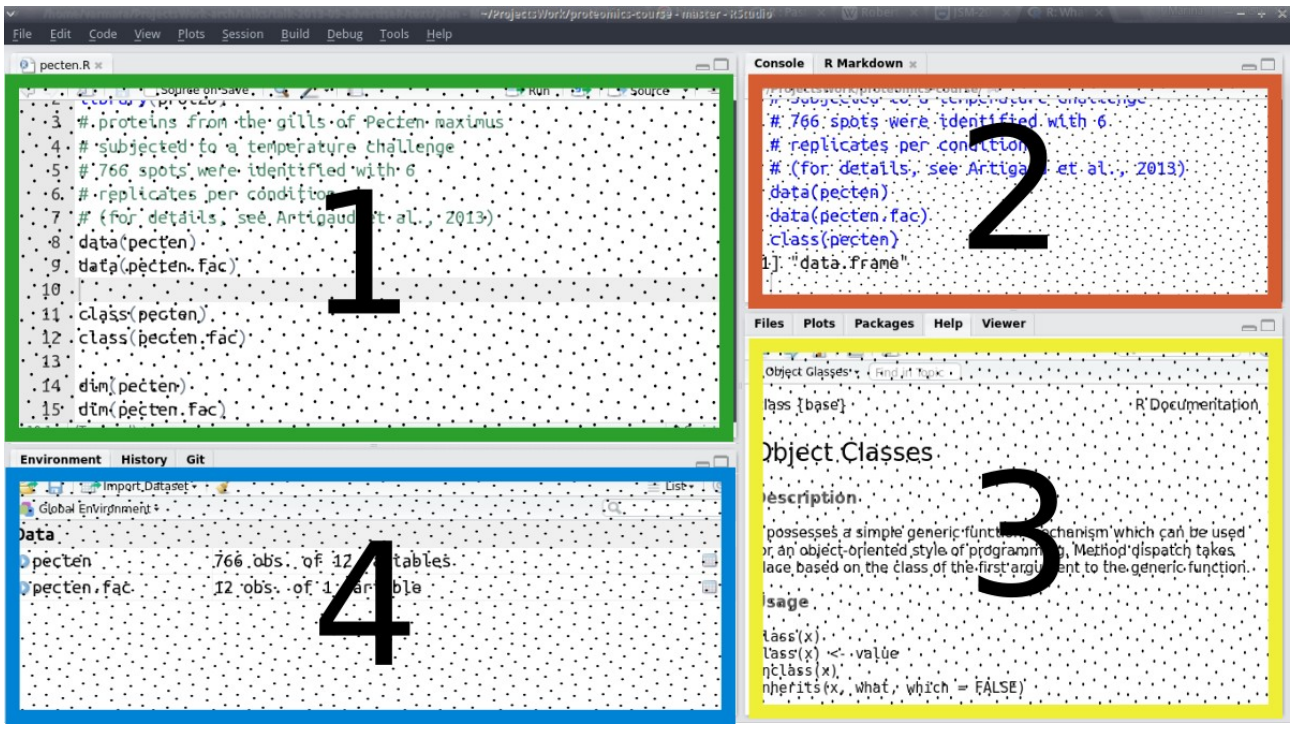


Рисунок 1. Схема рабочего пространства в RStudio.[9]

Среда разработки RStudio предоставляет структурированное рабочее пространство, разделённое на несколько функциональных окон Рисунок 1.Окно для написания скриптов (1, зелёная область) является основным для ввода программного кода. Выполнение команд осуществляется с помощью клавиши «Run» или комбинации клавиш Ctrl + Enter. Для последовательного запуска блока кода его необходимо предварительно выделить. Существует также возможность выполнить весь код от начала до текущей позиции курсора, используя комбинацию Ctrl + Alt + B. Консоль (4, синяя область) служит для вывода результатов выполнения команд, а также сообщений об ошибках и предупреждений, формируемых интерпретатором R. Окно «Environment» (2, красная область) отображает перечень всех объектов (функций, таблиц и т.п.), созданных пользователем в текущей рабочей сессии, что позволяет отслеживать состояние рабочего пространства. Многофункциональная панель (3, жёлтая область) объединяет несколько вкладок. На вкладке «Files» представлена информация о файловой системе, что облегчает навигацию по проектам.

9 https://varmara.github.io/proteomics/01_introduction_to_r

Вкладка «Help» предоставляет доступ к встроенной документации по функциям и пакетам, которую также можно вызвать командой *help()*. Следует отметить, что в некоторых конфигурациях RStudio данное окно также используется для визуализации и отображения графиков Рисунок 1.

Что касается настройки интерфейса, панель управления «Tools» предоставляет ограниченный набор параметров для персонализации, таких как изменение шрифта, его размера и цветовой схемы редактора. Важной особенностью RStudio, характерной для сред программирования в целом, является ориентация на командный, а не графический интерфейс. В качестве примера найдём сумму всех записей о длине листа ирисов [Fisher, 1936]. Для этого необходимо ввести следующую команду[10]:

```
sum(iris$Sepal.Length)
```

Запись до скобок представляет собой ***функцию***[11]**.** Обратите внимание, что результат будет выведен в консоль. Запись внутри скобок со ссылкой на объект представляет собой переменную. В программировании **переменная** - это именованная ячейка памяти компьютера, в которой хранится значение. Это значение может быть изменено во время выполнения программы. Переменные имеют фундаментальное значение для программирования, поскольку они позволяют хранить, обрабатывать и повторно использовать данные в программе. При необходимости, результат действия функции можно сохранить в виде пользовательского объекта:

```
a <- sum(iris$Sepal.Length)
```

Представленный выше фрагмент кода означает присвоение объекту его свойств[12]. В языке R действует важное правило управления объектами. Создание нового объекта и присвоение ему уже существующего названия, приводит к полному замещению предыдущего объекта. Данное свойство

10 В примере представлена адресация на столбец таблицы, подробнее см. раздел 6
11 Подробнее см. в разделе №3
12 Подробнее см. в разделе №2.1

системы требует от пользователя осознанного подхода к именованию во избежание непреднамеренной потери данных. Для минимизации рисков рекомендуется следовать следующим практикам:

- Если требуется сохранить предыдущее состояние объекта, новый следует наделить уникальным именем.
- Для временных объектов, используемых в рамках одноразовых операций, рекомендуется применение простых, стандартизированных обозначений (например, *df* для временного фрейма данных, *data* для входного набора данных).

Теперь мы познакомим читателя с понятием термина «объект». **Объект** в большинстве языков программирования - это фундаментальное понятие, представляющее реальную сущность или абстракцию в программе. Оно объединяет данные (атрибуты или свойства) и код (методы или поведение), которые работают с этими данными [MacLennan, 1982; Martin, 2012]. В зависимости от происхождения, все объекты в R подразделяются на три класса:

- Базовые — написаны на уровне ядра, существуют в базовой комплектации языка программирования R, доступны с момента его установки.
- Дополнительные — созданы сторонними разработчиками, входят в состав пакетов или библиотек. Становятся доступными только после загрузки и активации дополнительных библиотек.
- Пользовательские — созданы или загружены в среду R самим пользователем.

Примером встроенных объектов является таблица *iris* [Fisher, 1936], часто используемая в различных примерах по использованию R для решения тех или иных задач. Пользовательские объекты создаются непосредственно пользователем в процессе работы. Ими бывают исходные (входные) данные,

промежуточные результаты и финальные таблицы либо рисунки. Под входными данными понимаются исходные необработанные данные наблюдений либо экспериментов, зачастую представленные в виде таблиц. Примеров исходных данных в зоотехнии могут быть данные зоотехнического учёта, в биологии — наблюдения за популяциями животных, в пищевой промышленности — результаты химического анализа сырья. В RStudio пользовательские объекты отображаются в правой верхней части рабочего пространства Рисунок 1.

Промежуточные таблицы или иные объекты создаются в тех случаях, когда получение финальных таблиц (рисунков) невозможно или сложно получить в одно действие. Примером промежуточной таблицы будет подсчёт долей цыплят, кормление которых осуществляется разными рационами по данным встроенной таблицы *ChickWeigt*, которая в данном случае будет являться примеров входных данных. Исходные данные для таблицы *ChickWeigt* были собраны, чтобы проверить влияние содержания белка в 4 видах кормов на раннее развитие цыплят. 4 корма являются контрольными (обычный рацион) и 3 тестовых корма (10%, 20% и 40%-ная замена белка) [Crowder, Hand, 1990].

```
df
  Var1 Freq Prop
1    1  220 0.38
2    2  120 0.21
3    3  120 0.21
4    4  118 0.20
```

Выходные (финальные, итоговые) результаты представляют собой обобщённую информацию о входных данных. Обычно представлены в виде рисунков или таблиц. Примером выходного рисунка будет круговая диаграмма показывающая долю цыплят (*df$Prop*), кормление которых осуществляется разными рационами (*df$Var1*) .

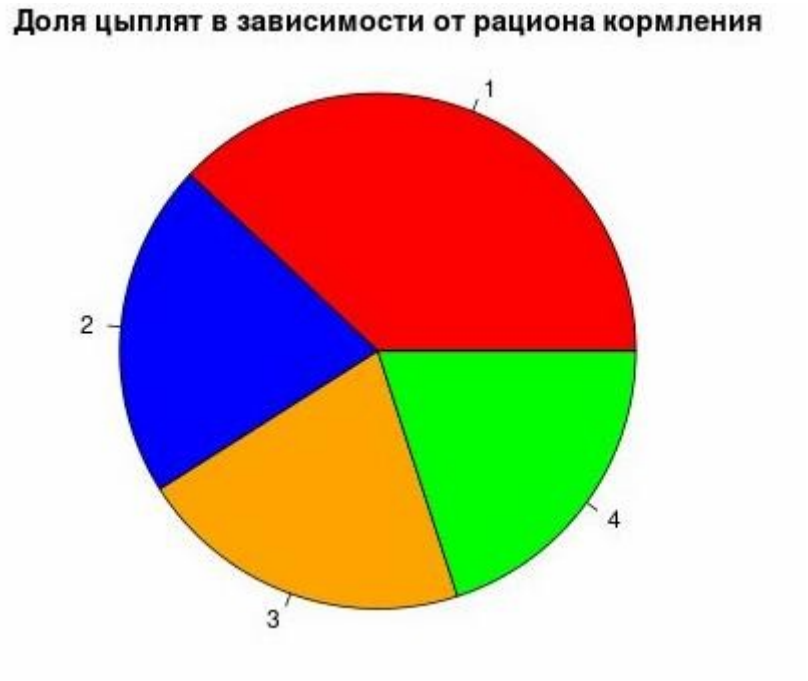


Рисунок 2 Доля цыплят разных рационов кормления по данным таблицы *ChickWeight.*

Контрольные вопросы к разделу

1. В каком году появился язык программирования R?
2. Кто считается его авторами?
3. Какой язык является предшественником R?
4. Что такое RStudio?
5. Какая вкладка интерфейса RStudio шрифт и фон рабочего пространства?
6. Что называют входными данными?
7. Что называют итоговыми результатами?

2. Типы данных и операторы

2.1. Операторы

Оператор в программировании — это конструкция языка, которая выполняет определённое действие над данными, являясь минимальной самостоятельной единицей программы. Язык программирования R характеризуется уникальной системой операторов, которая отражает его ориентацию на статистические вычисления и работу с данными. Эти операторы служат фундаментальным инструментом для выполнения широкого спектра операций — от базовых арифметических действий до сложных манипуляций с структурами данных. Для систематизации первоначального знакомства с ними основные операторы и их функциональное назначение представлены в таблице 1. Подробное описание по использованию операторов кода в R написано в электронном учебнике «Введение в R»[13].

Таблица 1 Основные операторы кода в R

Символ	Описание	Пример
Присвоение		
<-	Основной оператор присваивания	`a <- c("apple", "cherry", "banana")`
=	Оператор присваивания аргументов	`mean(df, na.rm = TRUE)`
Арифметические действия		
%/%	Деление на целое	`10 %/% 2` `# [1] 5`
+	Сложение	`1 + 1` `#[1] 2`
*	Умножение	`2 * 2` `#[1] 4`
^	Возведение в степень	`2 ^ 2` `#[1] 4`
%%	Остаток при делении на	`7 %% 3` `# [1] 1`

13 https://textbook.rintro.ru/syntax.html

	целое	
-	Вычитание	`3 - 1` `#[1] 2`
/	Деление	`8 / 4` `#[1] 2`
^(1/n)[14]	Извлечение корня n-й степени	`16 ^ (½)` `#[1] 4`
Доступ		

[ ]	доступ к элементу объекта по индексу	`Milk <- c(3.89, 3.98, 3.96, 4.12, 4.03, 2.50, 7.00, 4.23, 3.78, 3.94)`
		`Milk[Milk > 3.50 & Milk < 5.00]`
		`[1] 3.89 3.98 3.96 4.12 4.03 4.23 3.78 3.94`

$	доступ к элементу объекта по названию	`head(iris$Sepal.Length, 1)` `#[1] 5.1`
%in%	проверка принадлежности	см. ниже
Логические операторы		
==	Равно	`"green" == "red"` `#[1] FALSE`
!	Отрицание	`!is.na(1)` `#[1] TRUE`
!=	Не равно	`"green" != "blue"` `#[1] TRUE`

&	Логическое И	`animals <- c("Корова стада", "Архивная корова", "Тёлочка", "Бык", "Бычок", "Бычок")`
		`animals[animals != "Бык" & animals != "Бычок"]`
		`# [1] "Корова стада"  "Архивная корова" "Тёлочка"`

|	Логическое ИЛИ	`animals[animals == "Бык" | animals == "Бычок"]` `# [1] "Бык"  "Бычок" "Бычок"`
<=	Меньше либо равно	`5 <= 7` `#[1] TRUE`
>=	Больше либо равно	`5 >= 7` `#[1] FALSE`
>	Больше	`8 > 6` `#  [1] TRUE`
<	Меньше	`8 < 6` `#[1] FALSE`
Общий синтаксис		
.	Разделение десятичных знаков	`4.2`
~	Знак формулы	`lm(y ~ x)`
:	оператор последовательности	`1:5` `#[1] 1 2 3 4 5`

14 Выражение не является оператором

,	Разделение аргументов функции	`a <- c("apple", "cherry", "banana")`
#	Комментарий. Всё после # игнорируется	`# 17 ÷ 5 = 3 целых и 2 в остатке`
" "	Выделение строковых записей	`a <- c("apple", "cherry", "banana")`

В языке R операторы играют ключевую роль в выполнении различных операций. Для присвоения объектам их свойств обычно используется оператор `<-`, в то время как `=` применяется в основном для передачи аргументов в функции. Арифметические действия поддерживаются стандартными операторами: `+` для сложения, `-` для вычитания, `*` для умножения, `/` для деления, `^` для возведения в степень. Для специфических вычислений существуют специальные операторы: `%%` для получения остатка от деления, `%/%` для целочисленного деления, а корень n-й степени вычисляется через `^(1/n)` Таблица 1.

В предыдущем разделе был рассмотрен пример создания нового объекта с использованием ***оператора присвоения*** (символ `<-`). В стандартном коде R запись означает, что объекту слева от оператора присвоения необходимо присвоить свойства, которые описываются функцией справа от него. Так объект ***a*** в данном случае предствляет собой сумму всех записей о длине листа. Следует помнить, что объект следует называть строчными либо заглавными латинскими буквами (*object, Object, OBJECT*), либо сочетанием латинских букв с цифрами (*object1, object2, object3*). При этом название объекта всегда должно начинаться с латинской буквы, неважно, заглавной или строчной. Важно, что R является регистрзависимым языком, воспринимающим заглавные и строчные буквы как разные символы. Поэтому *object, Object, OBJECT* будут приняты системой как разные объекты. Если название объекта будет начинаться с цифры, система воспримет это как ошибку.

```
> 1a <- sum(iris$Sepal.Length)
Ошибка: неожиданный символ в "1a"
```

В названиях объектов нежелательно присутствие специальных символов (*, #, !, ? и т.д.). Если название объекта содержит более 2-х слов, рекомендуется в качестве разделителя использовать нижнее подчёркивание (*My_object*).

Для работы с данными используются операторы доступа: `[ ]` позволяет извлекать элементы по индексу или условию, а `$` обращается к элементам по имени. Логические операторы включают == (равно), `!=` (не равно), `<`, `>`, `<=`, `>=` для сравнений, а также `&` (И), `|` (ИЛИ) и `!` (отрицание) для комбинирования логических условий.

Синтаксические символы включают `"."` как разделитель десятичных дробей, ~ для создания формул в статистических моделях, `:` для генерации последовательностей, `,` для разделения аргументов функций и `#` для добавления комментариев в код. Каждый из этих операторов выполняет специфическую функцию, обеспечивая гибкость и эффективность программирования на R. `" "` выделяются строковые записи. Числовое значение без `" "` определяется как число, внутри `" "` как строка. Запись, начинающаяся с буквы латинского алфавита без `" "` определяется как название объекта, внутри `" "` как строка.

Оператор `%in%` возвращает логический вектор, показывающий, какие элементы первого вектора присутствуют во втором векторе. Это очень полезно для фильтрации данных и проверки принадлежности объектов к определенным группам в зоотехнических исследованиях.

```
# Создаём вектор с идентификаторами коров в основном стаде
основное_стадо <- c(101, 102, 105, 107, 110, 112)
# Создаём вектор с идентификаторами коров, отправленных на ветеринарный осмотр
на_осмотре <- c(105, 107, 109, 113)

# Проверяем, какие коровы из группы на осмотре находятся в основном стаде
на_осмотре %in% основное_стадо
# [1]  TRUE  TRUE FALSE FALSE
# Получаем конкретные идентификаторы коров, которые и в основном стаде, и на
осмотре
коровы_на_осмотре_из_стада <- на_осмотре[на_осмотре %in% основное_стадо]
коровы_на_осмотре_из_стада
# [1] 105 107
```

2.2. Типы данных в R

В традиционной статистике принято классифицировать данные на качественные (комолость, масть, форма рогов и т.д.) и количественные [Лакин, 1990], которые в свою очередь подразделяются на дискретные (яйценоскость, многоплодие, количество животных в стаде) и непрерывные (живая масса, удой, жирномолочность). Классификация признаков, используемых в традиционной статистике представлена на схеме Рисунок 3.

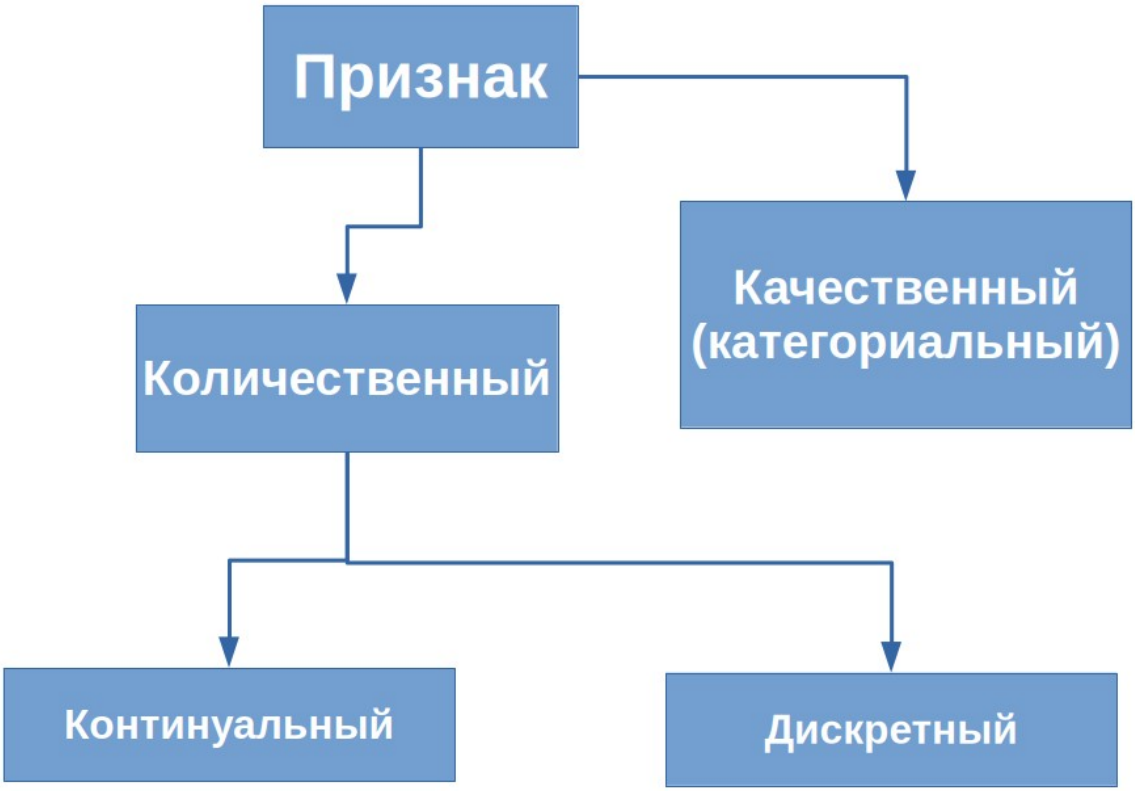


Рисунок 3. Схема классификации признаков в традиционной статистике.

В языке программирования R классификация данных несколько сложнее [Мастицкий, Шитиков, 2015; Уикем, Гроссер, Буманн, 2024; Шипунов и др., 2017]. Чёткого соответствия типов данных, используемых в R и в классической статистике нет, сравнение представлено в таблице 2.

Таблица 2. Соответствие признаков классической статистике и типов данных в R

Тип данных в R	Признак в классической статистике	Пояснение	Пример
Logical(логические)	Качественный	Наличие либо отсутствие	Комолое или рогатое животное, носитель наследственной аномалии либо «свободное» животное
Integer (цельночисельные)	Дискретный	Целые числа	Численность поголовья, яйценоскость
Numeric/double (дробные числа)	Континуальный	Дробные числа	Большинство хозяйственно полезных признаков: удой, жирномолочность, среднесуточный прирост, резвость
Character (строковые)	Качественный, дискретный	Записи о категориях	Масть, генотип, форма гребня, хозяйство
Factor (фактор)	Качественный, дискретный	Записи о категориях, связанных с зависимой переменной	Масть, генотип, форма гребня, хозяство
Date (даты)	–	Записи о датах	Дата рождения, дата взвешивания, дата вакцинации, дата выбытия
Complex (комплексный)	–	Сложный тип данных	–
Raw (бинарные)	–	Бинарный код	–

— представляет собой ответ на логическую операцию (равенство, не равенство, соответствие, не соответствие). Принимает значение TRUE(T) либо FALSE(F).

Integer(цельночисельные) — любое целое число (1, 2, 3, 0, -1). Такой тип данный можно присвоить континуальным (последовательным) признакам, таким как многоплодие, яйценоскость, продолжительность хозяйственного использования, выраженную в лактациях или опоросах. К этому же типу

данных относятся признаки, которые в зоотехнии принято округлять до целых чисел. В качестве разделителя десятичных знаков выступает точка.

Numeric/double (дробные числа) — дробные числа (0.5, 0.25, 3.14, -0.14587). Такой тип данных присваивается жирномолочности и содержанию белка в молоке.

Character(строковые) — представляет собой строковые записи. Автоматически определяет как любую запись, которая не является целым либо дробным числом, комплексными, бинарными и логическими данными. Как правило, представляет собой записи о качественных характеристиках: форма рогов, комолость, масть кличка животного. Числовые значения так же могут быть представлены в виде строковых записей. Например, это может быть использовано для обозначения экспериментальных групп. Числа разделённые двумя и более разделителями, не определённые как даты так же автоматически определяются как *characrer.*

В R существует понятие автоматического определения типов данных при вводе. В этом процессе существуют иерархия Рисунок 4. Первоначально система пытается определить данне как *logical*. Если данные не соответствуют *logical*, они определяются как *integer,* после — как *numeric*. Если ни одна из этих итераций не завершилась успехом, данные определяются как character. Такой процесс называют ещё ***естественным приведением типов данных***.

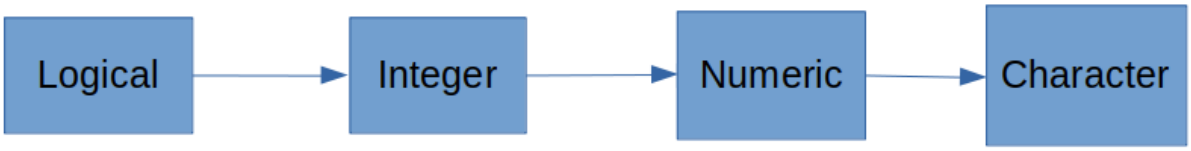


Рисунок 4 Схема иерархии автоматического преобразования типов данных

Обычным пользователям R крайне редко приходится иметь дело ещё с двумя типами данных.

Raw (бинарные) - используется для хранения “сырых” последовательностей байтов. Взаимодействие с этим типом данных как правило происходит вне зависимости от сознания пользователя;

Complex (комплексный) - тип данных, состоящий из комплексных чисел, предназначен для хранения чисел с мнимой составляющей[15]. Комплексные числа имеют общий вид

$$Z = a + ib$$

где a — действительная часть комплексного числа, b — мнимая часть комплексного числа, а i определяется как корень из -1. Используются в алгебраических расчётах, широкое применение получили в электронике и робототехнике [Ibarz и др., 2021].

В R выделяется ещё два типа данных ***Factor*** и ***Date*** с классификацией которых имеются некоторые сложности. Система R определяет оба типа данных, однако их существование в литературе как правило не упоминается.

Factor (фактор) — тип данных, который может быть представлен как числовыми,так и нечисловыми значениями. Обязательным условием для типа данных ***factor*** является наличие двух или более градаций. Использование типа данных ***factor*** имеет смысл при связанных с изучаемым показателем числовых зависимых переменных. Например, в стаде имеются свиноматки чёрной и белой масти, по которым ведётся учёт по живой массе, многоплодию, крупноплодию, массе поросят в 21 день и сохранности. Так вот, масть будет являться фактором, а живая масса, многоплодие и т.д. будут зависимыми переменными. В R имеет смысл переводить данные в тип ***factor*** при осуществлении анализа влияния градаций группирующего признака(фактора) на изменчивость зависимой переменной такими методами как ANOVA, критерий Краскела-Уоллеса,

15 https://www.geeksforgeeks.org/how-are-complex-numbers-used-in-real-life/

линейными смешанными моделями и т. д. [Мастицкий, Шитиков, 2015; Crawley, 2007; Venables, Ripley, 2002; Wickham, Grolemund, 2017].

Date (даты) — тип данных, в виде которого хранятся данные о любых временных величинах: часах, минутах, днях, месяцах и т.д. Сложность работы с датами в R состоит в том, что система по умолчанию воспринимает их как строковые записи. Что бы система начала различать даты от строковых записей следует провести трансформацию типов данных при помощи функции

as. Date(x, format = "%ep%ep%e").

где x — объект для преобразования в формат Date, e — элемент даты (число, месяц, год и т.д.), p — разделитель. Обозначения элементов даты в коде представлено в Таблица 3.

В зоотехническом учёте в формате Date могут быть представлены записи о дате рождения, выбытия, взвешивания и т.п., в биологии — дата мониторинга численности популяции диких животных, в пищевой промышленности — дата проведения дегустации. Например, необходимо создать объект, содержащий записи о дате рождения животных в принятом на территории России формате «*число.месяц.год*».

```
a <- as. Date(c("08.08.2020", "09.09.2020", "21.05.2021"), format = "%d.%m.%e")
str(a)
 a[1:3], format: "2024-08-20" "2024-09-20" "2024-05-20"
```

Таблица 3 Обозначения временных промежутков в коде языка программирования R

Обозначение	Описание	Пример
%a	Аббревиатура дня недели	Sun, Mon
%A	Полное наименование дня недели	Sunday, Monday
%b or %h	Аббревиатура месяца	Sep, Nov
%B	Полное наименование месяца	September, November
%d	Порядковый номер дня в месяце (число) 01-31	10,02
%j	Порядковый номер дня в году 001-366	250,31

%m	Порядковый номер месяца в году 01-12	09,11
%U	Порядковый номер недели в году 01-53 с воскресеньем в качестве первого дня недели	35,45
%w	Будний день 0-6, воскресенье - 0	0,4
%W	Неделя	21,27
%y	Год без столетия (сокращённая нумерация года) 00-99	84,05
%Y	Полная нумерация года	1984,2011
%C	Век, столетие	19,2
%D	Дата в формате %месяц/%день/%год без столетия	09/10/93, 11/20/93
%u	День недели начиная с понедельника	7,4

Кроме формата Date в R существуют дополнительные форматы, позволяющие проводить манипуляции с датами например, *POSIXlt* и *POSIXct.* Работа с указанными форматами доступна после установки пакета *lubridate* [Grolemund, Wickham, 2011]. При помощи соответствующих функций пакета *lubridate* возможно извлечение числа, месяца и других элементов даты при помощи соответствующих функций[16].

Типы данных возможно трансформировать из одного в другой используя функции семейства as*.data_type()*, где под *data_type* подразумевается название типа данных в который необходимо трансформировать объект. Например, имеется вектор a, состоящий из чисел от одного до десяти. По умолчанию, созданный вектор будет иметь тип данных *integer.* Следовательно, для трансформации вектора a в типы данных *character* или *numeric* необходимо использовать функции *as.character* и *as.numeric* соответственно.

16 https://cran.r-project.org/web/packages/lubridate/vignettes/lubridate.html

```
a <- 1:10
str(a)
 int [1:10] 1 2 3 4 5 6 7 8 9 10
a <- as.character(a)
str(a)
 chr [1:10] "1" "2" "3" "4" "5" "6" "7" "8" "9" "10"
a <- as.numeric(a)
str(a)
 num [1:10] 1 2 3 4 5 6 7 8 9 10
```

Основных типов данных в R, с которыми постоянно сталкиваются пользователи, четыре[17,18].

Контрольные вопросы к разделу:

1. К какому типу данных относятся целые числа?
2. Что обозначает этот оператор: `==`?
3. Каким оператором можно присвоить свойства какому-либо объекту?
4. С помощью какого оператора происходит извлечение записей из объектов?
5. Для чего служит оператор `^`?
6. Каким образом в R извлекается корень n-й степени?
7. Как называется тип данных, используемый для хранения “сырых” последовательностей байтов
8. К какому типу данных будет относиться объект:

   ```
   a
   [1] "2020-08-08"
   ```
9. К какому типу данных будет относиться объект:

   ```
   a <- c(1, TRUE, 8.25, "apple")
   ```

17 https://www.programiz.com/r/data-types
18 https://swcarpentry.github.io/r-novice-inflammation/13-supp-data-structures.html

3.Функции

3.1. Классификация функций

Классификация функций в R слабо освещена в научной литературе. Функции в R, как и объекты, по происхождению подразделяются на три группы (стр. 15). Другой системой классификации функций может быть количество аргументов. К первой группе можно отнести большинство функций. Это функции с фиксированным количеством аргументов, или те, где число аргументов определяется разработчиком. Для примера разберём функцию *mean*. На официальном ресурсе *R y*казано три её аргумента: *x, trim, na.rm*[19] и их количество остаётся неизменным вне зависимости от входных данных. Другой тип функций — с неограниченным числом аргументов. Их количество определяется пользователем и варьирует в зависимости от конкретной ситуации. Примером такой функций может быть *data.frame (),* где количество аргументов по сути равно числу создаваемых столбцов.

3.2. Базовые функции

Функции — это основной инструмент работы в языке программирования R. Функция в программировании — это именованный, независимый и многократно используемый блок кода, который предназначен для выполнения конкретной задачи [Martin, 2012]. По другому функция - это набор инструкций, объединённых для выполнения определённой операции. Функция - это объект, поэтому интерпретатор может передавать управление функции вместе с аргументами, которые могут потребоваться функции для выполнения действий. Функция, в свою очередь, выполняет задачу и возвращает управление интерпретатору, а также любые возвращаемые значения, которые могут храниться в других объектах. Функция принимает в себя входные данные (называемые аргументами или параметрами), обрабатывает их и возвращает результат своей работы. В большинстве случаев в качестве первого аргумента функции принима-

19 https://www.rdocumentation.org/packages/base/versions/3.6.2/topics/mean

ют набор исходных данных [Gentleman, Ihaka, 2000; Ihaka, Gentleman, 1996]. Аргумент — это конкретное значение, которое передаётся функции при её вызове, чтобы она использовала его в своих вычислениях. Другими словами, аргументы являются своеобразными настройками функции, так или иначе изменяющими функционирование, а соответственно и результат. Для примера рассмотрим функцию *seq()*, генерирующую последовательность чисел с заданными параметрами. Основные аргументы у функции *seq()* два: *from* — обозначающий число с которого начинается генерируемая последовательность, *to* — конечное значение генерируемой последовательности чисел.

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

Зачастую аргументов функции значительно больше, но большинство из них в коде напрямую не используется, т.е. работает по умолчанию. Например, третий аргумент функции *seq()*, *by*, отвечает за интервал, по умолчанию равен единице. То есть, если пользователя устаревает работа функции по умолчанию, соответствующий аргумент как правило не пишется в коде. Если нужно изменить «настройки» функции, в коде обязательно пишется соответствующий аргумент. Например, для изменения интервала генерируемой последовательности следует изменить значение аргумента *by*.

```
seq(1, 10, 2)
[1] 1 3 5 7 9
```

Обращаем внимание на то, что если написать аргументы функции в строго определённом порядке, то прописывать название самих аргументов необязательно. Например, аргумент *from* функции *seq()* по умолчанию пишется первым, *to* — вторым, *by* третьим. Но что бы не забивать себе голову порядком аргумента функций достаточно запомнить используемые вами аргументы и прописывать их в коде в произвольной последовательности.

```
seq(to = 10, by = 2, from = 1)
```

```
[1] 1 3 5 7 9
```

Обычно в R функции обладают таким свойством, как векторизированность, то есть функция, без дополнительных «настроек» в коде способна работать только с линейным одномерным набором данных. Некоторые функции, такие как *summary()*, способны распространять своё действие на таблицы целиком.

По разным оценкам, среднестатистический пользователь R постоянно и осознанно использует от 20 до 200 функций. Это составляет менее 0.1% от всего доступного арсенала языка, но является ядром его продуктивности. Во многом, это число зависит от квалификации пользователя. Начинающий пользователь вполне может обходиться 50-ю функциями, опытный аналитик — 200, разработчик пакетов — 300 функций [Wickham, Grolemund, 2017]. В таблице 4 представлен ряд функций, который поможет начинающему пользователю начать работать с R.

Рассмотрим некоторые базовые функции R. Например, функция *sum* позволяет получить суммарное значение всех записей, *mean* — получить среднее арифметическое, функции семейства *as* позволяют трансформировать объекты и типы данных. Информацию о структуре объекта можно получить используя функцию *str()*. В качестве примера используем функцию *str()* для получения информации о структуре встроенного набора данных *iris.*

```
str(iris)
'data.frame':    150 obs. of  5 variables:
 $ Sepal.Length: num  5.1 4.9 4.7 4.6 5 5.4 4.6 5 4.4 4.9 ...
 $ Sepal. Width : num  3.5 3 3.2 3.1 3.6 3.9 3.4 3.4 2.9 3.1 ...
 $ Petal. Length: num  1.4 1.4 1.3 1.5 1.4 1.7 1.4 1.5 1.4 1.5 ...
 $ Petal. Width : num  0.2 0.2 0.2 0.2 0.2 0.4 0.3 0.2 0.2 0.1 ...
 $ Species     : Factor w/ 3 levels "setosa","versicolor",..: 1 1 1 1 1 1
1 1 1 1 ...
```

Таблица 4 Некоторые базовые функции R

№пр	Название	Назначение
1	sum()	Выдаёт сумму значений объекта
2	mean()	Выдаёт среднее значение записей
3	str()	Отображает структуру и тип(ы) данных объекта
4	length()	Отображает количество записей вектора либо количество ячеек матрицы
5	library()	Активирует дополнительную библиотеку
6	install.packages ("название")	Скачивает дополнительную библиотеку с официального репозитория
7	table()	Подсчитывает количество повторений каждой записи
8	prop.table(table())	Подсчитывает долю каждой записи
9	grep("pattern", x)	Возвращает адреса ячеек объекта x, содержащие записи "pattern"
10	help()	Возвращает информацию о свойствах встроенного объекта, в частности о принципе использования функций
11	rm()	Удалить объект
12	abs()	Возвращает модуль числа
13	summary()	Отображает информацию о содержимом объекта.
14	sqrt()	Возвращает квадратный корень
15	match(x, y)	возвращает номера записей y, присутствующие в x
16	dim()	Возвращает размерность матрицы или таблицы
17	print()	Выводит содержимое объекта на экран
18	round(x, n)	Округляет число x до n десятичных знаков
19	prod()	Перемножение всех чисел в векторе
20	format(x, scientific=FALSE)	Преобразование формата чисел из научного формата в числовой. Например, 5.445698e-03 преобразуется в 0.005445698413
21	rep("pattern", n)	Производит запись "pattern" n-е количество раз

Результатом действия функции *str()* в данном случае представляет собой информацию о классе объекта, его размерности и типах данных. Показано, что iris является таблицей (data.frame), состоящей из 150 строк и пяти столбцов, четыре из которыхзаполненыы данными типа numeric. Пятый столбец заполнен данными типа factor, который представлен тремя градациями (категориями).

Очень полезной является функция *summary()*, позволяющая получать наиболее общую информацию о содержимом объекта. При использовании функции *summary()* на наборы данных, имеющие тип *integer* или *numeric* в консоле появляется информация о наиболее общих статистических показателях: средняя арифметическая (*Mean*), минимум (*min*), максимум (*max*), первый (*1st Qu*) и третий (*3st Qu*) квантили и медиану (*Median*). на данных типа *character* — информация о долях записей от общего количества.

```
  summary(iris)
    Sepal.Length    Sepal.Width     Petal. Length    Petal. Width          
Species  
  Min.   :4.300   Min.   :2.000   Min.   :1.000   Min.   :0.100   setosa    :50
  1st Qu.:5.100   1st Qu.:2.800   1st Qu.:1.600   1st Qu.:0.300   versicolor:50
  Median :5.800   Median :3.000   Median :4.350   Median :1.300   virginica :50
  Mean   :5.843   Mean   :3.057   Mean   :3.758   Mean   :1.199                 
  3rd Qu.:6.400   3rd Qu.:3.300   3rd Qu.:5.100   3rd Qu.:1.800                 
  Max.   :7.900   Max.   :4.400   Max.   :6.900   Max.   :2.500
```

Один из базовых терминов, используемых при работе с функциями, является рекурия. Рекурсия или рекурсивная процедура (функция) — это процедура (функция), которая вызывает сама себя на-прямую или через другие процедуры и функции. Частым примером использования рекурсии в R является использование одной функции внутри другой. Рассмотрим случай нахождения квадратного корня из суммы значений.

```
a <- 1:10
sqrt(sum(a))
```

3.3. Создание пользовательских функций

Для создания пользовательских функций в R существует функция *function(x){}*. В круглых скобках перечисляют аргументы, в фигурных скобках — тело функции. Например, требуется функция, рассчитывающая сумму каждой записи с единицей, будет выглядеть:

```
my_fun <- function(x){x+1}
 x <- 1:10
my_fun(x)
 [1]  2  3  4  5  6  7  8  9 10 11
```

В животноводстве одним из важных показателей откормочной продуктивности является относительный прирост живой массы, рассчитываемый по формуле (1) [Свечин, 1976]

$$K = \frac{W_t - W_0}{W_0} * 100 \quad (1)$$

где W_0 и W_t – живая масса в начале и в конце изучаемого периода.

Для разбора примера расчёта относительного прироста в R создадим два вектора, содержащие записи о массе животных на начало и конец изучаемого периода.

```
Wt <- c(952, 890, 934, 924, 928, 965)
W0 <- c(400, 450, 399, 454, 398, 455)
```

Функция расчёта относительного прироста в R будет выглядеть:

```
K <- function(x, y){((x-y)/y)*100}
K(Wt, W0)
[1] 138.00000  97.77778 134.08521 103.52423 133.16583 112.08791
```

Пользовательские функции можно сохранять в файлах (см. разд. 7.2). Для их загрузки в систему можно использовать функцию *source()*. В круглых скобках прописывается путь к файлу[20].

```
source("my_functions.R")
```

20Тамбовцева А. Основы программирования в R.Функции // https://rpubs.com/AllaT/rbase-functions. 2020

3.4. Загрузка дополнительных пакетов

Дополнительная библиотека (пакет) представляет собой набор функций, как правило для решения задач в какой-либо определённой области. Например, пакет *nortest* содержит в себе функции расчётов критериев нормальности[21]. Как правило, дополнительные пакеты хранятся на официальном репозитории языка R – CRAN. В подавляющем большинстве случаев загрузка дополнительных библиотек с CRAN осуществляется посредством функции *install.packages ("ggplot2").* В кавычках пишется название необходимого пакета, в данном случае представлен популярная библиотека для расширения функционала работы с графиками, *ggplot2.* Установка пакета выполняется единожды, до выхода более актуальной версии R. Следует помнить, что перед началом использование дополнительной библиотеки её необходимо вручную активировать при помощи команды *library()*. Название необходимой библиотеки можно писать как в кавычках, так и без них. Активация дополнительной библиотеки необходима при каждом запуске R. В некоторых случаях требуется загрузка дополнительного пакета со стороннего ресурса. Для этого в функции *install.packages()* добавляют аргумент *source* с указанием адреса внешнего ресурса.

```
install.packages("/путь к папке/ название пакета.tar.gz", type = "sourse", repos = NULL)
```

Интересным представляется репозиторий *Bioconductor,* содержащий ряд функций и библиотек для решения задач по обработке данных по генеалогии, генетики и т.п. Установка библиотек из репозитория *Bioconductor* имеет свои особенности. Сперва требуется установка специальной библиотеки, для чего нужно выполнить следующий код:

```
if (!require("BiocManager", quietly = TRUE))
  install.packages("BiocManager")
BiocManager::install(version = "3.19")
```

21 https://rdocumentation.org/packages/nortest/versions/1.0-4

Затем для установки необходимого пакета выполняется команда:

```
BiocManager::install("GeneticsPed")
```

В данном случае оператор `::` выполняет функцию разделителя между названием пакета и названием функции. В коде это очень удобно, потому, что в нескольких пакетах могут встречаться функции с идентичным названием. Использование такого оператора (`::`) позволяет уточнять вызов функции.

Контрольные вопросы к разделу:

1. С помощью какой функции можно получить информацию о сумме всех чисел объекта?
2. С помощью какой функции можно узнать информацию о количестве записей в объекте?
3. Какая функция служит для создания пользовательских функций?
4. Какая функция позволяет загружать дополнительные библиотеки?
5. Какой репозиторий является основным хранилищем библиотек и функций языка программирования R?
6. Какой репозиторий является хранилищем библиотек, приспособленных для обработки данных в области биологии?

Практические задания:

1. Напишите функцию расчёта абсолютного прироста
2. Напишите функцию расчёта доли особей в популяции, выраженной в процентах
3. Установите библиотеку *MASS*
4. *Округлите 0.349207 до сотых*

5. Возведите число 3 в 4-ю степень
6. Извлеките квадратный корень из 16 при помощи специальной функции
7. Извлеките корень 3-й степени из 27

4.Вектор

4.1.Создание векторов

Вектор - одномерный объект, содержащий в себе один тип данных. Если вектор содержит в себе всего одну запись, его называют *единичным вектором.* Базовой функцией для создания вектора является *c(),* от англ. *construction*, то есть в дословном переводе «создавать». Но по сути вектор создаётся любой функцией, результатом действия которой является одномерный набор данных, например *runif(), rep(), seq()* и т. д. Однако создание вектора возможно и при помощи оператора последовательности:

```
a <- 1:5
> a
[1] 1 2 3 4 5
```

При попытке создать вектор, содержащий в себе одновременно логические выражения (TRUE, FALSE), цельные числа, дробные числа, строковые записи данные будут преобразованы в один тип данных по принципу:

```
a <- c(1, 1.2, FALSE)
b <- c(1, 1.2, FALSE, "apple")
str(a)
 num [1:3] 1 1.2 0
str(b)
 chr [1:4] "1" "1.2" "FALSE" "apple"
```

Для повторения одной и той же записи n-е количество раз используют функцию *rep(x, n)*, где *x* – запись, *n* – количество повторов. Необходимость использования функции *rep()* может возникнуть при заполнении всего вектора одной и той же записью, например, названием хозяйства.

```
a <- rep("Farm_name", 5)
a
[1] "Farm_name" "Farm_name" "Farm_name" "Farm_name" "Farm_name
```

Функция *rep()* может пригодится при создании вектора, содержащим записи о нескольких категориях. Например, необходимо создать вектор, содержащий по 2 записи о породной принадлежности лошадей.

```
Horse <- c(rep("арабская", 2), rep("андалузская", 2), rep("ЧКВ", 2))
Horse
[1] "арабская" "арабская" "андалузская" "андалузская" "ЧКВ"   "ЧКВ"
```

Для рандомизации записей в векторе подходит функция *sample()*.

```
sample(Horse)
[1] "ЧКВ" "арабская" "арабская" "андалузская" "ЧКВ" "андалузская"
```

При создании возрастающей последовательности чисел с интервалом равным единице достаточно просто обозначить прогрессию, а функции не использовать.

```
a <- 1:10
a
 [1]  1  2  3  4  5  6  7  8  9 10
```

Для создания векторов, содержащих совокупность чисел с фиксированным интервалом, удобна функция *seq()*. Данная функция имеет следующие основные аргументы: *from* — число с которого начинается генерация, to — число на котором генерация должна прекратиться, *by* — интервал, *length.out* — ограничение количества записей.

4.2. Адресация

Адресация в векторе происходит двумя способами. Первый из них, по номеру записи. Например из вектора Horse необходимо извлечь 3-ю запись

```
Horse[3]
[1] "андалузская"
```

При необходимости извлечения нескольких соседних записей адресацию осуществляют используя номер первой и последней записи разделённых `:`.

```
Horse[1:4]
```

```
[1] "арабская"    "арабская"    "андалузская" "андалузская"
```

Для адресации на несколько не соседних записей используют функцию *c()*.

```
 Horse[c(1,2,5,6)]
[1] "арабская" "арабская" "ЧКВ"      "ЧКВ"
```

Второй способ адресации осуществляется по содержимому записей, происходит с использованием логических операторов. Например, из вектора *Horse* необходимо извлечь все записи, относящиеся к арабской породе лошадей.

```
Horse[Horse == "арабская"]
[1] "арабская" "арабская"
```

Над векторами возможно осуществление математических операций. Если с каждой записью вектора необходимо совершить какую либо математическую операцию, код в R будет выглядеть:

```
a <- 1:10
a +1
[1]  2  3  4  5  6  7  8  9 10 11
```

При сложении (вычитании, делении, умножении) 2-х и более векторов математическая операция будет осуществляться между идентичными записями векторов: первой и первой, второй и второй и т.д. Поэтому обязательным условием выполнения математических действий между векторами является их одинаковая длина.

```
a + b
 [1]  2  4  6  8 10 12 14 16 18 20
```

4.3.Объединение векторов.

К уже существующему вектору можно добавить любую запись. Последовательность будет зависеть от написания кода. Для примера к вектору *cattle*, содержащим записи о половозрастных категориях крупного рогатого скота, добавим запись *piglet* (поросёнок, англ.).

```
cattle <- c("Haifer", "Cow", "Bull", "Calve")
cattle1 <- c(cattle, "piglet")
cattle1
[1] "Haifer" "Cow"    "Bull"   "Calve"  "piglet"
cattle2 <- c("piglet", cattle)
cattle2
[1] "piglet" "Haifer" "Cow"    "Bull"   "Calve"
```

По аналогии происходит и объединение двух и более векторов в один.

```
cattle <- c("Haifer", "Cow", "Bull")
pigs <- c("Sow", "Boar", "Piglet")
farm_animal <- c(cattle, pigs)
farm_animal
[1] "Haifer" "Cow"    "Bull"   "Sow"    "Boar"   "Piglet"
```

Следует помнить, что при объединении 2-х векторов с разными типами данных, тип данных будет преобразован согласно принципу естественного приведения типов данных Рисунок 4.

4.4.Генерация псевдослучайных наборов данных

Иногда приходится сталкиваться с задачами, требующими данных с заранее известными параметрами. Конечно, можно попытаться найти их среди ранее изученных наборов данных, но это крайне долго, трудно, а в некоторых случаях практически невозможно. Гораздо проще воспользоваться так называемыми ***синтетическими данными***, то есть искусственно созданными наборами данных с заданными параметрами[Jordon и др., 2022]. Для их создания пользуются семейством функций ***генерации псевдослучайных чисел***. В результате создаётся набор данных, сгенерированный случайным образом, но всё же ограниченный заданными параметрами (минимум, максимум, стандартное отклонение, распределение и т.п.). По этой причине такие наборы данных нельзя считать полностью случайными [Kietzmann, Schmidt, Wählisch, 2021]. В качестве

простейших генераторов псевдослучайных чисел можно использовать функции *runif()* либо *rnorm ()*.

Функцию *runif()* можно использовать в тех случаях, когда необходимо обозначить лимиты и объём выборки.

```
runif(min = 10, max = 20, n = 10)
 [1] 16.41761 13.96487 17.80435 14.19786 11.37036 17.58223 11.27903 18.01207 11.57774 10.35842
```

Функцию *rnorm ()* удобно применять в тех случаях, когда необходимо задать среднее значение (аргумент mean), объём выборки (аргумент n) и величину изменчивости, выраженную через стандартное отклонение (аргумент sd).

```
rnorm(n = 10, mean = 4, sd = 0.25)
[1] 3.930004 4.306284 3.807578 4.046539 4.196039 3.861392 4.071587 3.824430 3.986992 3.539687
```

Контрольные вопросы к разделу:

1. Как называется вектор, состоящий из одной записи
2. В какой тип данных будет преобразован вектор, образованный объединением векторов: a и b

```
a <- 1:5
b <- c("apple", "cherry", "banana")
```

3. Какую функцию следует использовать для повтора записей заданное количество раз?
4. Какую функцию следует использовать для генерации последовательности чисел с заданным интервалом?
5. Каким аргументом следует воспользоваться для ограничения количества записей?
6. Для решения каких задач используются синтетические данные?
7. Какой функцией следует воспользоваться, если нужно сгенерировать данные при известных лимитах и объёме выборки.
8. Какую функцию можно использовать для генерации набора данных с заданными параметрами нормального распределения (n, mean, sd).

9. Вектор a состоит из чисел от 1 до 5. Вектор b создали действием

```
b <- a*2
```

Из каких чисел будет состоять вектор b .

10. Есть два вектора

```
a <- 1:5
b <- 10:6
```

Какие числа получаться при вычитании:

```
b-a
```

Практические задания:

1. Из вектора *month.abb* извлеките 10-ю запись
2. Из вектора *state.name* извлеките 13-20 записи
3. Создайте единичный вектор, содержащий в себе запись типа данных logical
4. Создайте вектор, который начинается на 2, заканчивается на 8 и состоит из 14 записей с равным интервалом.
5. Создайте вектор, из 14 записей начинающийся на 3 с интервалом 1.243
6. Из получившегося вектора извлеките 6-ю, 8-ю и 10-ю записи
7. Создайте вектор, в котором слово «Лошадь» повторяется 10 раз.
8. Создайте вектор, в котором в случайной последовательности 3 раза повторяются записи «Бык», «Тёлка» и «Корова»
9. Из получившегося вектора извлеките записи «Бык».
10. При помощи функции *runif()* создайте вектор, имитирующий набор данных о питатательности пищевых продуктов из свинины (130-250 ккал, n = 100).

5.Матрицы

Матрицы — двумерный объект, состоящий из строк и столбцов содержащий в себе данные одного типа. Создание матриц осуществляется как минимум тремя функциями. В литературе наиболее часто встречается информация, что создание новой матрицы происходит посредством функции *matrix(data, nrow, ncol)*, где *data* — набор данных, *nrow* — количество строк, *ncol* — количество столбцов [Уикем, Гроссер, Буманн, 2024; Fieller, 2016].

```
my_mat <- matrix(data = 1:25, nrow = 5, ncol = 5)
my_mat
     [,1] [,2] [,3] [,4] [,5]
[1,]    1    6   11   16   21
[2,]    2    7   12   17   22
[3,]    3    8   13   18   23
[4,]    4    9   14   19   24
[5,]    5   10   15   20   25
```

Обращаем внимание, что заполнение матрицы по умолчанию происходит по столбцам. Если необходимо заполнение матрицы по строкам, в коде прописывается аргумент *byrow* = TRUE.

```
my_mat <- matrix(data = 1:25, nrow = 5, ncol = 5, byrow = TRUE)
my_mat
     [,1] [,2] [,3] [,4] [,5]
[1,]    1    2    3    4    5
[2,]    6    7    8    9   10
[3,]   11   12   13   14   15
[4,]   16   17   18   19   20
[5,]   21   22   23   24   25
```

Так же матрицы возможно создать из готовой таблицы с одним типом данных, преобразовав соответствующую таблицу с помощь функции *as.matrix()*. Ещё один возможный способ создания матриц, это объединение векторов с помощью функций *rbind()* и *cbind()*. Важно помнить, что результатом объединения двух и более векторов посредством функций *rbind()* и *cbind()*, будет именно матрица, а не таблица

```
a <- 1:5
b <- 6:10
c <- 11:15
d <- 16:20
```

```
rbind(a,b,c,d)
  [,1] [,2] [,3] [,4] [,5]
a    1    2    3    4    5
b    6    7    8    9   10
c   11   12   13   14   15
d   16   17   18   19   20

cbind(a,b,c,d)
     a  b  c  d
[1,] 1  6 11 16
[2,] 2  7 12 17
[3,] 3  8 13 18
[4,] 4  9 14 19
[5,] 5 10 15 20
```

Важно, что «сборки» матрицы из векторов при помощи функций *rbind()* и *cbind()* будет таким же, как прописаны соответствующие аргументы в скобках. С помощью этих же функций можно присоединять вектор к уже существующим матрицам. Функция *rbind()* осуществляет присоединение вектора к матрице с увеличением числа строк, *cbind()* – с увеличением количества столбцов.

```
a <- seq(2.33, by = 0.26, length.out = 5)
rbind(my_mat, a)
  [,1]  [,2]  [,3]  [,4]  [,5]
  1.00  6.00 11.00 16.00 21.00
  2.00  7.00 12.00 17.00 22.00
  3.00  8.00 13.00 18.00 23.00
  4.00  9.00 14.00 19.00 24.00
  5.00 10.00 15.00 20.00 25.00
a 2.33  2.59  2.85  3.11  3.37
```

Адресация на любую ячейку происходит с указанием в квадратных скобках номеров соответствующей строки и столбца. Разделителем нумерации столбцов и строк выступает запятая.

```
 my_mat[2,3]
[1] 12
```

Если нужно адресовать на какой-либо столбец, в квадратных скобках пишется соответствующий номер после запятой, до запятой ничего не пишется

```
my_mat[,3]
[1] 11 12 13 14 15
```

Если нужно адресовать на какую-либо строку, в квадратных скобках пишется соответствующий номер до запятой, после запятой ничего не пишется

```
 my_mat[2,]
[1]  2  7 12 17 22
```

Если необходимо адресовать на группу ячеек, номера соответствующих строк и столбцов группируют при помощи функции *c()* или символа :

```
my_mat[c(2,3), c(2,3)]
     [,1] [,2]
[1,]    7    8
[2,]   12   13
```

```
   [,1] [,2] [,3]
[1,]    7    8    9
[2,]   12   13   14
[3,]   17   18   19
```

Особенностью матрицы является возможность адресации на диагональ при помощи функции *diag()*.

```
     diag(my_mat)
[1]  1  7 13 19 25
```

В матрицах под диагональю понимаются ячейки, идущие из левого верхнего угла в правый нижний, как показано ниже.

```
m
     [,1]   [,2]   [,3]   [,4]   [,5]
[1,] "diag" "6"    "11"   "16"   "21"
[2,] "2"    "diag" "12"   "17"   "22"
[3,] "3"    "8"    "diag" "18"   "23"
[4,] "4"    "9"    "14"   "diag" "24"
[5,] "5"    "10"   "15"   "20"   "diag"
```

Присвоение имён строкам и столбцам матрицы осуществляется функциями *row.names()* и *colnames()*. Эти же функции способны извлекать информацию текущих именах строк или столбцов. Результат зависит от расположения функций в строке кода относительно оператора присвоения. Если функция *row.names()* или *colnames()* расположены слева от оператора присвоения, значит строкам или столбцам таблицы будут присвоены новые имена. Если справа от оператора присвоения или без его использования — будет получена информация о текущих именах строк или столбцов

```
row.names(my_mat) <- c("R1", "R2", "R3", "R4", "R5")
my_mat
   [,1] [,2] [,3] [,4] [,5]
R1    1    2    3    4    5
R2    6    7    8    9   10
R3   11   12   13   14   15
R4   16   17   18   19   20
R5   21   22   23   24   25

colnames(my_mat) <- c("C1", "C2", "C3", "C4", "C5")
> my_mat
   C1 C2 C3 C4 C5
R1  1  2  3  4  5
R2  6  7  8  9 10
R3 11 12 13 14 15
R4 16 17 18 19 20
R5 21 22 23 24 25

a <- row.names(state.x77)
> head(a)
[1]"Alabama" "Alaska" "Arizona" "Arkansas" "California" "Colorado"
```

Строки и столбцы матриц возможно менять местами, что называется транспонированием. В R транспонирование осуществляется функцией *t()*. Транспонирование так же можно сравнить с «переворачиванием на бок»/

```
m <- matrix(data = c("A", "B", "C", "D"), ncol = 2, nrow = 2)
> m
     [,1] [,2]
[1,] "A"  "C"
[2,] "B"  "D"
> t(m)
     [,1] [,2]
[1,] "A"  "B"
[2,] "C"  "D"
```

Важной особенностью матриц является то, что такие функции как *mean*, *sum*, *sd* и т.п. одинаково работают как применимо к строкам, так и к столбцам. Но следует помнить, что подобные функции работают только при адресации на 1 строку или 1 столбец. Для вычисления суммы и средней арифметической по нескольким столбцам применяются функции *colMeans()* и *colSums()*.

Контрольные вопросы к разделу:

1. С помощью какой функции в R создаётся матрица?
2. Какой аргумент определяет порядок заполнения информации в матрице?

3. С помощью какой функции можно преобразовать таблицу в матрицу?
4. Какая функция объединяет две и более матрицы или присоединяет вектор к матрице таким образом, что бы увеличивалось количество строк, а количество столбцов оставалось неизменным?
5. Какая функция объединяет две и более матрицы или присоединяет вектор к матрице таким образом, что бы увеличивалось количество столбцов, а количество строк оставалось неизменным?
6. Какая функция позволяет извлечь записи из диагонали матрицы?
7. Какая функция позволяет транспонировантъ матрицы?

Практические задания:

1. Создайте матрицу M1 5×5, наполнив её числами начиная с 4.548 и интервалом 1.257
2. Извлеките диагональ
3. Извлеките 2-ю и 3-ю строки 1-го и 4-го столбца
4. Создайте матрицу M2 55. Содержимое: 25 чисел, заканчивающихся на 53 с интервалом 0.656. Число 39.88 должно располагаться в 1 строке 5 столбце
5. Найдите сумму диагонали матрицы M2
6. Найдите среднее значение 2-й строки
7. Найдите сумму нечётных столбцов
8. По данным матрицы state.x77 вычислите среднее значение по каждому столбцу
9. Объедините матрицы M1 и M2 таким образом, что бы число строк осталось неизменным
10. Объедините матрицы M1 и M2 таким образом, что бы число столбцов осталось неизменным. Транспонируйте получившуюся матрицу

6.Таблицы или data.frame

Data.frame — класс двуменых объектов хранения данных, организованный как набор столбцов одинаковой длины, где каждый столбец представляет собой отдельный список элементов. Ключевое отличие от матрицы заключается в том, что эти столбцы могут содержать данные различных типов [Maindonald, 2008; Peng, 2022]. Вторым отличием таблиц от матриц является отсутствие понятия диагонали. По общепринятому правилу, в столбах содержится информация о признаках (удой за 305 дней среднесуточный прирост, конверсия корма, группа крови и т.д). В строках располагаются записи о наблюдениях. Для примера продемонстрированы первые шесть строк одной из наиболее популярных таблиц в R — *iris.*

```
  Sepal.Length Sepal.Width Petal. Length Petal. Width Species
1          5.1         3.5          1.4         0.2  setosa
2          4.9         3.0          1.4         0.2  setosa
3          4.7         3.2          1.3         0.2  setosa
4          4.6         3.1          1.5         0.2  setosa
5          5.0         3.6          1.4         0.2  setosa
6          5.4         3.9          1.7         0.4  setosa
```

Результат был получен при использовании функции *head()*. Схожий результат получается при использовании функции *tail()*, только результатом служат последние 6 строк.

```
tail(iris)
    Sepal.Length Sepal.Width Petal. Length Petal. Width   Species
145          6.7         3.3          5.7         2.5 virginica
146          6.7         3.0          5.2         2.3 virginica
147          6.3         2.5          5.0         1.9 virginica
148          6.5         3.0          5.2         2.0 virginica
149          6.2         3.4          5.4         2.3 virginica
150          5.9         3.0          5.1         1.8 virginica
```

Как правило, таблица включает в себя два и более типа данных, хотя встречаются таблицы, состоящие из одного типа данных. Таблицы являются настолько востребованным классом объектов, что в R появились такие

разновидности как разновидности *tibble* [Wickham, Grolemund, 2017] и *data.table*[22] со своими особенностями адресации и управлением данными.

Информацию о размерности таблицы можно узнать при помощи функции *dim ()* получая на выходе число строк и число столбцов, *ncol()* - число столбцов, *nrow()* - число строк.

6.1.Адресация в таблицах

Адресация на сроки и ячейки по номерам не отличается от правил, описанных в разделе «Матрицы». Адресация на столбцы так же возможна по номерам столбцов, но чаще используют названия. Преимущество такого способа заключается в том, что при увеличении количества столбцов либо при перемени их местами производительность кода не меняется. Для этого можно воспользоваться оператором извлечения с написанием названия столбца в кавычках. Что бы не выводить в консоль всё содержимое столбца воспользуемся функцией *head()* для демонстрации первых нескольких строк.

```
head(Bulls["Number"])
      Number
1 3263337318
2 3251555942
3 3264522457
4 3269404461
5   12948857
```

Схожим образом можно сразу извлечь два и более столбцов

```
      head(Bulls[c("Number", "Date")])
      Number       Date
1 3263337318 2023-02-12
2 3251555942 2023-03-24
3 3264522457 2023-02-21
4 3269404461 2023-03-23
5   12948857 2017-10-25
```

Но чаще при адресации на один столбец пользуются следующим кодом:

```
 Bulls$ID
[1] 1 2 3 4 5
```

22 https://rdatatable.gitlab.io/data.table/articles/datatable-intro.html

6.2.Получение итоговых таблиц из первичных данных

Для получения итоговых таблиц из первичных данных создают пользовательский объект при помощи функции *data.frame*, которую наполняют функциями, непосредственно извлекающими информацию. Например, для получения общей информации о средней арифметической в столбцах и строках существуют функции *colMeans()*и *rowMeans()*; для вычисления суммы *colSums()* и *rowSums()*.

```
df <- data.frame(colSums(iris[1:4]))
> df
             colSums.iris.1.4..
Sepal.Length              876.5
Sepal.Width               458.6
Petal. Length             563.7
Petal. Width              179.9
```

Обратите внимание на две интересные особенности:

1. Имена категорий исходных данных становятся именами строк новой таблицы;
2. Название столбца с информацией формируется из записи кода его создавшего, что для итоговой таблицы не есть хорошо. Исправить данный недостаток можно используя функцию *colnames()*.

```
colnames(df) <- "Сумма"
> df
             Сумма
Sepal.Length 876.5
Sepal.Width  458.6
Petal.Length 563.7
Petal.Width  179.9
```

Информацию о встречаемости строковых записей можно извлекать при помощи функций *table()* и *prop.table()*. Функция *table()* в результате даёт информацию об абсолютной встречаемости каждой записи. В качестве примера возьмём данные эксперимента по вскармливанию в приёмных семьях с матерями-крысами и выводками четырёх различных генотипов: A, B, I и J. Крысята были отделены от своих естественных матерей при рождении и переданы приёмным

матерям для выращивания. Целью эксперимента было выяснить, как влияет генотип биологической матери и приёмной матери на рост крысят [Bailey, 1953]. Данные присутствуют в наборе данных *MASS::genotype*.

```
library(MASS)
gen <- MASS::genotype
df <- data.frame(table(gen))
  Var1 Freq
1    A   16
2    B   14
3    I   16
4    J   15
```

Обратите внимание, что в таких случаях результатом будет таблица с постоянными названиями столбцов: *Var1* — имена категорий, *Freq* — число записей. Преобразование в доли происходит при помощи «скармливания» столбца *Freq* функции *prop.table()*. Для примера создадим ещё один столбец:

```
df$Prop <- prop.table(df$Freq)
  df
  Var1 Freq      Prop
1    A   16 0.2622951
2    B   14 0.2295082
3    I   16 0.2622951
4    J   15 0.2459016
```

Точно так же можно собирать информацию по описательной статистике из таблиц с исходными данными в итоговые таблицы. Для начала, создадим функцию, собирающую данные о численности выборки, средней арифметической (mean), ошибке средней арифметической (se), стандартном отклонении (sd), медиане(median), лимитах (min, max) и квантилях. В качестве исходных данных возьмём таблицу *crabs* из пакета *MASS*, содержащую результаты изучения размера панцирей крабов [Campbell, Mahon, 1974]. Для начала, создадим функцию, которая будет собирать необходимые данные.

```
opstat <- function(x){
  df = data.frame(
    n = sum(!is.na(x)),
    Mean = mean(x),
    SE = sd(x)/sqrt(sum(!is.na(x))-1),
    SD = sd(x),
    Med = median(x),
    Min = min(x),
    Max = max(x),
```

```
    Q1 = quantile(x, 0.25),
    Q3 = quantile(x, 0.75)
  )
  df = t(df)
}
```

Используем нашу функцию opstat для создания итоговой таблицы по длине панциря спереди (FL) у крабов с оранжевыми панцирями

```
df1 <- opstat(crabs$FL[crabs$sp == "O"])
df1
```

```
n    100.0000000
Mean  17.1100000
SE     0.3292077
SD     3.2755754
Med   17.5000000
Min    9.1000000
Max   23.1000000
Q1    14.5250000
Q3    19.4750000
```

Переименуем получившийся столбец

```
colnames(df1) <- "Orange"
      Orange
n    100.0000000
Mean  17.1100000
SE     0.3292077
SD     3.2755754
Med   17.5000000
Min    9.1000000
Max   23.1000000
Q1    14.5250000
Q3    19.4750000
```

В результате получилась таблица из одного столбца, где показатели описательной статистики зашифрованы в строках. Повторим действие с крабами, имеющим голубой цвет панциря, а затем объединим получившиеся таблицы. В итоге у нас получилась итоговая таблица, сравнивающая показатели описательной статистики передней длины панциря у крабов оранжевой и голубой окраски.

```
df2 <- opstat(crabs$FL[crabs$sp == "B"])
colnames(df2) <- "Blue"
df <- cbind(df1, df2)
df
```

```
     Orange        Blue
n    100.0000000 100.0000000
Mean  17.1100000  14.0560000
SE     0.3292077   0.3034822
SD     3.2755754   3.0196100
Med   17.5000000  14.4500000
Min    9.1000000   7.2000000
Max   23.1000000  21.3000000
Q1    14.5250000  11.8000000
Q3    19.4750000  16.1250000
```

6.3.Управление таблицами.

Создание таблиц осуществляется через команду *data.frame*. Для примера, создадим таблицу *Bulls*.

```
Bulls <- data.frame(
ID = 1:5,
Number = c(3263337318, 3251555942, 3264522457, 3269404461, 12948857),
FullName = c("PEAK CH ALTAINSPIRE-ET", "PEAK ALTAZEROGRAVITY-ET", "PEAK
ALTAVORTEXVAULT-ET", "PEAK ALTATURBOTIME-ET", "PEAK ALTAKLAEBO-ET"),
Country = c("USA", "USA", "USA", "USA", "CAN"),
Date = as.Date(c("12.02.2023", "24.03.2023", "21.02.2023", "23.03.2023",
"25.10.2017"), format = "%d.%m.%Y")
)
```

Часто приходится сталкиваться с ситуациями, когда необходимо внести в таблицу дополнительные столбцы. Например, в хозяйстве начали учитывать дополнительный признак. Для примера внесём в таблицу *Bulls* данные по TPI.

```
Bulls$TPI <- c(3275, 3270, 3263, 3245, 2940)
Bulls
  ID     Number                 FullName Country       Date  TPI
1  1 3263337318  PEAK CH ALTAINSPIRE-ET     USA 2023-02-12 3275
2  2 3251555942 PEAK ALTAZEROGRAVITY-ET     USA 2023-03-24 3270
3  3 3264522457 PEAK ALTAVORTEXVAULT-ET     USA 2023-02-21 3263
4  4 3269404461   PEAK ALTATURBOTIME-ET     USA 2023-03-23 3245
5  5   12948857      PEAK ALTAKLAEBO-ET     CAN 2017-10-25 2940
```

Объединение таблиц с увеличением числа столбцов и строк производится при помощи функций *rbind()* и *cbind()* соответственно. Потребность в функции *cbind()* может возникнуть при появлении нового показателя в учёте. Напимер, исследователь в течении нескольких лет занимался моделированием среднесуточного прироста подсвинков, имея данные о происхождении и продуктивности родителей. И вот он выиграл грант на генотипирование животных по STR-локу-

сам. У него сохранились ушные выщипы с данными о происхождении и продуктивности и у него появились данные о генотипировании животных. Следовательно, в таблицу с данными нужно добавить столбцы, содержащие данные генотипирования. Подобные задачи в R решаются при помощи *cbind()*. Этой функцией можно объединять как таблицу и вектор, так и две и более таблиц между собой. При объединении вектров функциями *cbind()* и *rbind()* получается матрица. Ограничением использования функции *cbind()* является одинаковое количество строк объединяемых таблиц. При объединении таблицы и вектора нужно, что бы длина вектора соответствовала количеству строк таблицы. С помощью функции *rbind()* рекомендуется объединять таблицы, при условии равенства числа столбцов, идентичности их названия и порядка. Присоединение вектора к таблице в качестве новой строки при помощи функции *rbind()* технически возможно, однако существует риск преобразования типов данных. Решением проблемы может быть трансформация вектора в однострочную таблицу с приведением типов данных. Для удаления ненужных столбцов можно либо адресовать на них через отрицание, либо присвоить NULL.

```
Bulls[-1]
      Number                 FullName Country       Date  TPI
1 3263337318  PEAK CH ALTAINSPIRE-ET     USA 2023-02-12 3275
2 3251555942 PEAK ALTAZEROGRAVITY-ET     USA 2023-03-24 3270
3 3264522457 PEAK ALTAVORTEXVAULT-ET     USA 2023-02-21 3263
4 3269404461   PEAK ALTATURBOTIME-ET     USA 2023-03-23 3245
5   12948857      PEAK ALTAKLAEBO-ET     CAN 2017-10-25 2940
```

```
Bulls$ID <- NULL
Bulls
Number                 FullName Country       Date  TPI
1 3263337318  PEAK CH ALTAINSPIRE-ET     USA 2023-02-12 3275
2 3251555942 PEAK ALTAZEROGRAVITY-ET     USA 2023-03-24 3270
3 3264522457 PEAK ALTAVORTEXVAULT-ET     USA 2023-02-21 3263
4 3269404461   PEAK ALTATURBOTIME-ET     USA 2023-03-23 3245
5   12948857      PEAK ALTAKLAEBO-ET     CAN 2017-10-25 2940
```

При создании таблиц следует однако учитывать, что длины всех столбцов таблицы должны быть одинаковыми. Если перед вами стоит задача создать таблицу с разным количеством записей в столбцах, то недостающее количеством

записей дополняют строками с NA[23]. Количество строк с NA рассчитывается по формуле:

$$x = n_{max} - n_i \quad (2)$$

где x — количество записей, которое необходимо заполнить NA, n_{max} — численность наиболее многочисленной группы, n_i - численность текущей группы. Для примера рассмотрим создание таблицы о содержании килокалорий в продуктах из говядины, свинины и курицы. В качестве исходных данных возьмём набор данных о пищевой ценности продуктов[24] . Для начала, найдём численность каждой группы. Наиболее многочисленную группу передаём без изменений. В оставшиеся выборки добавляем NA согласно формуле (2). Для этих целей логично использовать функцию *rep(x, n)*. Алгоритм создания таблицы будет следующим

```
      #Загружаем исходные данные
      food <-
read.csv("/home/hp1234/Документы/Учебное/Комьпьютеризация/Практика/
food_kaggle.csv",
                       sep = ",", dec = ".", header = TRUE)
      # находим численность каждой группы
      tapply(df1$Protein, df1$Category, mf)
      # создаём таблицу с равной длиной столбцов
      df <- data.frame(
        Beef = food$Kilocalories[food$Category == "BEEF"], ы
        Chicken = c(food$Kilocalories[food$Category == "CHICKEN"], rep(NA,
294)),
        Pork = c(food$Kilocalories[food$Category == "PORK"], rep(NA, 153))
        )
```

6.4. Объединение таблиц по записям ключевого столбца

Ключевым называют столбец в таблице данных, значение которого однозначно идентифицирует каждую строку [Abadi, Boncz, Harizopoulos, 2009]. В животноводстве это столбец, содержащий записи об уникальных номерах животных. Бывают ситуации, когда необходимо объединить две таблицы, где уникальные номера могут совпадать не на 100%. Такая ситуация может возникнуть

23 Подробнее об NA см. разд. 11

24https://www.kaggle.com/datasets/shrutisaxena/food-nutrition-dataset

из-за асинхронного сбора данных. Например, в животноводстве данные по продуктивности возможно получить лишь по достижению животным определённого возраста: завершения лактации у молочных коров, откорм подсвинков и молодняка молочного скота, опороса у свиней и т.д. Данные генотипирования можно получить от молодых животных и не у всех животных, от которых получена продуктивность могли прогенотипировать. В изучении диких животных при повторном изучении могут быть обследованы новые животные, а особи, изученные в прошлый раз могут погибнуть, мигрировать, быть съедены хищниками и тд. Рассмотрим пример из области животноводства. Имеются данные по продуктивности (*Production*) и генотипированию (*Gen*) животных. В таких случаях в R применяется функция *merge(x, y, by.x, by.y)*, где x, y названия объединяемых таблиц, *by.x*, *by.y* названия ключевого столбца в первой и второй таблицах. Если названия ключевого столбца идентичны *by.x*, *by.y* заменяют аргументом *by.*

```
Production
     ID  Milk  Fat Protein
1  1121  9908 3.90    3.57
2  2023 10190 4.53    3.67
3  4120  9093 3.97    3.46
4   918  9587 4.70    3.28
5  5417  8453 4.11    3.64
6  4578  9960 5.31    3.69
7  5647  8470 4.91    3.24
8  8954  7972 5.25    3.66
9  1452  9672 5.03    3.31
10 5874  9776 5.35    3.22
```

```
Gen
     id G1 G2
1  5417  A  A
2  4578  A  a
3  5647  A  a
4  8954  A  A
5  3456  a  A
6  2345  a  A
7  2376  A  A
8  2390  A  A
9  3456  A  a
10 2312  A  a
```

```
merge(Production, Gen, by.x = "ID", by.y = "id")
    ID Milk  Fat Protein G1 G2
1 4578 9960 5.31    3.69  A  a
2 5417 8453 4.11    3.64  A  A
3 5647 8470 4.91    3.24  A  a
4 8954 7972 5.25    3.66  A  A
```

6.5. Фильтрация таблиц

Для извлечения информации из таблицы по данным столбца, содержащего записи об условиях фильтации базовый код языка R будет выглядеть следующим образом.

```
Bulls[Bulls$Country == "USA",]
  ID     Number                FullName Country       Date
1  1 3263337318  PEAK CH ALTAINSPIRE-ET     USA 2023-02-12
2  2 3251555942 PEAK ALTAZEROGRAVITY-ET     USA 2023-03-24
3  3 3264522457 PEAK ALTAVORTEXVAULT-ET     USA 2023-02-21
4  4 3269404461   PEAK ALTATURBOTIME-ET     USA 2023-03-23
```

Извлечение из таблицы по данным столбца, содержащего информацию об условии извлечения, возможно при помощи функции *subset()*. Первым аргументом функции является массив данных, из которого осуществляется извлечение, при помощи аргумента subset задаётся условие извлечения посредством логических операторов Таблица 1. Схема извлечение данных согласно условиям фильтрации сведена в Таблица 5

```
subset(Bulls, subset = Bulls$Country == "USA")
  ID     Number                FullName Country       Date
1  1 3263337318  PEAK CH ALTAINSPIRE-ET     USA 2023-02-12
2  2 3251555942 PEAK ALTAZEROGRAVITY-ET     USA 2023-03-24
3  3 3264522457 PEAK ALTAVORTEXVAULT-ET     USA 2023-02-21
4  4 3269404461   PEAK ALTATURBOTIME-ET     USA 2023-03-23
```

Сортировка таблицы по данным некого отдельного столбца имеет свои особенности. Обычно для сортировки таблицы сначала пишут её название, а внутри оператора извлечения функцию *order()*, внутри которой адресуют на ключевой столбец. Запятая после круглых скобок указывает функции, что ей нужно работать по строкам.

```
Bulls[order(Bulls$TPI),]
  ID     Number                FullName Country       Date  TPI
5  5   12948857      PEAK ALTAKLAEBO-ET     CAN 2017-10-25 2940
4  4 3269404461   PEAK ALTATURBOTIME-ET     USA 2023-03-23 3245
3  3 3264522457 PEAK ALTAVORTEXVAULT-ET     USA 2023-02-21 3263
2  2 3251555942 PEAK ALTAZEROGRAVITY-ET     USA 2023-03-24 3270
1  1 3263337318  PEAK CH ALTAINSPIRE-ET     USA 2023-02-12 3275
```

Таблица 5 Схема извлечение по данным столбца с именами категорий

При помощи оператора извлечения	С использованием функции Subset()	Условия извлечения данных
iris$Sepal.Length [iris$Sepal.Length > 5]	subset(iris$Sepal.Length, subset = iris$Sepal.Length > 5)	Данных о-го столбца по условиям о-го столбца
iris$Sepal.Width [iris$Species != "virginica"]	subset(iris$Sepal.Width, subset = iris$Species != "virginica")	Данных о-го столбца по условиям о-го столбца
iris[iris$Sepal.Length > mean(iris$Sepal.Length),]	subset(iris, subset = iris$Sepal.Length > mean(iris$Sepal.Length))	Данных таблицы по условиям о-го столбца
iris[iris$Sepal.Length <= 3 \| iris$Petal.Length < 1.2,]	subset(iris, subset = iris$Sepal.Length <= 3 \| iris$Petal.Length < 1.2)	Данных таблицы при независимом соблюдении условий фильтрации по двум и более столбцам
iris[iris$Species == "setosa" & iris$Petal.Length < 1.5,]	subset(iris, iris$Species == "setosa" & iris$Petal.Length < 1.5)	Данных таблицы при совместном соблюдении условий фильтрации по двум и более столбцам

6.6. Широкие и длинные таблицы.

Нередко приходится сталкиваться с задачей трансформации таблиц из широких в длинные и наоборот. Широкими таблицами называют те где зависимый признак распределён по разным столбцам в зависимости от градаций категориального фактора. Примером может служить записи молочной продуктивности коров по лактациям в отдельных столбцах. Длинными называют те, где записи зависимого признака сведены в один столбец, а имена категорий образуют отдельный столбец. В качестве примера рассмотрим таблицу, имитирующую записи об удой коров за 305 дней по первым четырём лактациям. Так как для наполнения таблицы был использован генератор псевдослучайных чисел *rnorm()*, её наполнение раз за разом будет иметь определённые различия.

```
wide <- data.frame(
  ID = 1:10,
  `Лактация1` = round(rnorm(n = 10, mean = 8500, sd = 500)),
  `Лактация2` = round(rnorm(n = 10, mean = 9000, sd = 520)),
  `Лактация3` = round(rnorm(n = 10, mean = 10500, sd = 450)),
  `Лактация4` = round(rnorm(n = 10, mean = 10000, sd = 530))
)
```

```
ID Лактация_1 Лактация_2 Лактация_3 Лактация_4
1   1       8551       8435      10189      10521
2   2       7993       9175      10176      10180
3   3       8360       9615      11503      10155
4   4       9277       8454      10460       9541
5   5       8351       9111      10207       9757
6   6       8659       9212       9774      10643
7   7       8107       9440      10058      10217
8   8       8394       7410      10700      10136
9   9       8378       9427      10017       9448
10 10       8488       8316      10708       9193
```

Получившуюся таблицу преобразуем в длинный формат при помощи функции *reshape()*. Аргументы функции описаны ниже как комментарии к примеру.

```
long <- reshape(
  data = wide, # имя исходной широкой таблицы
  sep = "",
  idvar= "ID",#столбец с записями о индивидуальном ключе записей
  varying = 2:5, #столбцы с данными, подлежащие преобразованию
  timevar= "Номер лактации", # название столбца с именами категорий
    new.row.names= 1:(ncol(wide[2:5])*nrow(wide)), #количество строк в длинной таблице. Аргумент нужен для непрерывной нумерации строк
  direction = "long"
)
```

```
   ID Номер лактации Лактация
1   1              1     8551
2   2              1     7993
3   3              1     8360
4   4              1     9277
5   5              1     8351
6   6              1     8659
7   7              1     8107
8   8              1     8394
9   9              1     8378
10 10              1     8488
11  1              2     8435
12  2              2     9175
13  3              2     9615
14  4              2     8454
15  5              2     9111
16  6              2     9212
17  7              2     9440
18  8              2     7410
19  9              2     9427
```

```
20 10              2     8316
21  1              3    10189
22  2              3    10176
23  3              3    11503
24  4              3    10460
25  5              3    10207
26  6              3     9774
27  7              3    10058
28  8              3    10700
29  9              3    10017
30 10              3    10708
31  1              4    10521
32  2              4    10180
33  3              4    10155
34  4              4     9541
35  5              4     9757
36  6              4    10643
37  7              4    10217
38  8              4    10136
39  9              4     9448
40 10              4     9193
```

Функции *reshape()* трансформирует как широкие таблицы в длинные, так и наоборот. Решающим в данном аспекте выступает аргумент *direction* . Если прописать *long* будет трансформация в длинную таблицу, *wide* в широкую. Обращаем внимание на использование аргумента sep. Аргумент используется в зависимости наличия разделителя в именах колонок между буквенной и цифровой составляющими названия. Недостатком функции *reshape()* является наполнение столбца с именами категорий только данными типа *integer()*.

Теперь рассмотрим преобразование длинной таблицы в широкую. В качестве примера создадим таблицу, содержащую записи о количестве людей с разным цветом волос и разным цветом глаз по данным базового объекта R *HairEyeColor* [Snee, 1974].

```
HECM <- data.frame(HairEyeColor)
```

В результате получается длинная таблица, где столбцы содержат информацию о поле, цвете волос и цвете глаз. Последний столбец содержит количество наблюдений.

```
HECM
      Sex HairColor EyeColor Freq
1    Male      Blac    Brown   32
2    Male      Blac     Blue   11
3    Male      Blac    Hazel   10
4    Male      Blac    Green    3
5    Male     Brown    Brown   53
6    Male     Brown     Blue   50
7    Male     Brown    Hazel   25
8    Male     Brown    Green   15
9    Male       Red    Brown   10
10   Male       Red     Blue   10
11   Male       Red    Hazel    7
12   Male       Red    Green    7
13   Male     Blond    Brown    3
14   Male     Blond     Blue   30
15   Male     Blond    Hazel    5
16   Male     Blond    Green    8
17 Female      Blac    Brown   36
18 Female      Blac     Blue    9
19 Female      Blac    Hazel    5
20 Female      Blac    Green    2
21 Female     Brown    Brown   66
22 Female     Brown     Blue   34
23 Female     Brown    Hazel   29
24 Female     Brown    Green   14
25 Female       Red    Brown   16
26 Female       Red     Blue    7
27 Female       Red    Hazel    7
28 Female       Red    Green    7
29 Female     Blond    Brown    4
30 Female     Blond     Blue   64
31 Female     Blond    Hazel    5
32 Female     Blond    Green    8
```

Для трансформации в широкий формат используем функцию *reshape()*.

```
wide <- reshape(data = HECM,
        direction = "wide", #выбирает тип трансформации
      idvar = c("HairColor"), #столбец с индивидуальными идентификаторами
записей
         timevar = "EyeColor", #столбец с именами категорий
         v.names = "Freq" #столбец с записями зависимой переменой
                  )
```

```
 Sex HairColor Freq.Brown Freq.Blue Freq.Hazel Freq.Green
1  Male      Blac         32        11         10          3
5  Male     Brown         53        50         25         15
9  Male       Red         10        10          7          7
13 Male     Blond          3        30          5          8
```

В данном примере в качестве индивидуальных идентификатор записей были приняты названия цвета волос. В результате получилась таблица содержащая записи о людях с разным цветом глаз (столбцы) и разным цветом

волос(строки). Однако, в итоговую таблицу вошли только записи о мужчинах. Что бы в итоговой широкой таблице добавились ещё и записи о женщинах, включим столбец S*ex()* в число индивидуальных идентификаторов.

```
wide <- reshape(data = HECM,
                direction = "wide",
                idvar = c("HairColor", "Sex"),
                timevar = "EyeColor",
                v.names = "Freq"
                  )
wide
      Sex HairColor Freq.Brown Freq.Blue Freq.Hazel Freq.Green
1    Male      Blac         32        11         10          3
5    Male     Brown         53        50         25         15
9    Male       Red         10        10          7          7
13   Male     Blond          3        30          5          8
17 Female      Blac         36         9          5          2
21 Female     Brown         66        34         29         14
25 Female       Red         16         7          7          7
29 Female     Blond          4        64          5          8
```

6.7. Поиск дублированных записей

В животноводстве постоянно требуется уделять вниманию качеству записей зоотехнического учёта [Камалдинов и др., 2022; Камалдинов и др., 2024; ÇeliKyürek и др., 2019]. Имеют место повторение номеров и кличек животных в строках. Для примера возьмём таблицу со случайным набором данных о кличках коров Таблица 6

Таблица 6 Пример дублирования кличек молочных коров с использованием случайных данных

```
df
Number      Name  d  m    Y       Date
1    6296 Радостная 14  1 2006 2006-01-14
2    5979    Качеля 20  4 2021 2021-04-20
3    1398   Бурёнка 10  7 2016 2016-07-10
4    6448 Радостная 23  2 2005 2005-02-23
5    2376     Ягода  4 11 2000 2000-11-04
6    2376     Ягода 18  5 2018 2018-05-18
7    6932   Бурёнка 21  3 2009 2009-03-21
8    7224    Качеля 16  8 2004 2004-08-16
9    7218 Радостная 22  8 2019 2019-08-22
10   7218 Радостная 22  8 2019 2019-08-22
11   4886    Качеля 18 11 2004 2004-11-18
12   4895   Бурёнка 16  3 2007 2007-03-16
```

Существует проблема идентификации дубликатов от записей разных животных. Решением оказалось идентификация животных по ключу, состоящему из номера, клички и даты рождения. В таблице 6 коровы Ягода 2376 и Радостная 7218 встречаются дважды, но благодаря разным датам рождения понятно, что записи о коровах Ягода 2376 относятся к разным животным. Записи о корове Радостная 7218 являются дубликатами. Избавиться от дублированных строк позволяет функция *unique()*.

```
df
   Number       Name  d  m    Y
1    6296 Радостная 14  1 2006
2    5979    Качеля 20  4 2021
3    1398   Бурёнка 10  7 2016
4    6448 Радостная 23  2 2005
5    2376     Ягода  4 11 2000
6    2376     Ягода 18  5 2018
7    6932   Бурёнка 21  3 2009
8    7224    Качеля 16  8 2004
9    7218 Радостная 22  8 2019
10   7218 Радостная 22  8 2019
11   4886    Качеля 18 11 2004
12   4895   Бурёнка 16  3 2007
```

```
unique(df)
   Number       Name  d  m    Y
1    6296 Радостная 14  1 2006
2    5979    Качеля 20  4 2021
3    1398   Бурёнка 10  7 2016
4    6448 Радостная 23  2 2005
5    2376     Ягода  4 11 2000
6    2376     Ягода 18  5 2018
7    6932   Бурёнка 21  3 2009
8    7224    Качеля 16  8 2004
9    7218 Радостная 22  8 2019
11   4886    Качеля 18 11 2004
12   4895   Бурёнка 16  3 2007
```

Контрольные вопросы к разделу:

1. С помощью какой функции в R создаётся таблица?
2. Перечислите основные отличия таблицы от матрицы?
3. Какая информация обычно содержится в строках, а какая — в столбцах таблицы?

4. Что называют ключевым столбцом?
5. Какая функция позволяет извлекать строки по записям столбуа с именами категорий?
6. Какая функция используется для объединения таблиц по записям ключевого столбца?
7. Что подразумевают понятия «широкие» и «длинные» таблицы?
8. Какая функция позволяет преобразовывать широкие таблицы в длинные и наоборот?

Практические задания:

1. Создайте таблицу *Prod*, имитирующую записи о продуктивности 100 свиноматок с нормальным распределением признаков: скороспелость (mean = 160, sd = 11), затраты корма на килограмм прироста (mean = 2.5, sd = 0.15), длина туловища (mean = 125, sd = 3), многоплодие (mean = 15, sd = 2).
2. К таблице добавьте столбец с индивидуальными номерами от 1 до 100
3. Создайте таблицу *Gen*, имитирующую результаты генотипирования. Таблица должна состоять из 2-х столбцов. Первый столбец содержит информацию об индивидуальном номере животного: 20 животных со случайными номерами от 1 до 100 и ещё 20 со случайными номерами от 101 до 999. Второй столбец должен содержать 20 записей AB, 10 AA, 10 BB, распределённых случайным образом.
4. Объедините таблицы *Prod* и *Gen* по записям ключевого столбца в таблицу Data таким образом, что бы все строки таблиц *Prod* и *Gen* сохранились.
5. По данным таблицы *Gen* сформируйте итоговую таблицу, содержащую информацию о доле животных с разными генотипами в процентах.

6. Используя пользовательскую функцию opstat создайте таблицу, сравнивающую показатели описательной статистики длины листа у разных видов ириса.
7. Получившуюся таблицу преобразуйте в длинный формат
8. Из таблицы *MASS::crabs* извлеките записи с оранжевым цветом панциря.
9. Преобразуйте таблицу *InsectSprays* в широкий формат.

7. Импорт и экспорт данных

7.1.Импорт данных

Импорт данных является одним из ключевых деталей работы с языком программирования R, так как представляет собой ввод данных пользователей в среду программирования для их последующей обработки. Как правило, ими могут быть данные зоотехнического или иного учёта, результаты экспериментов и т.п. Если первичные данные изначально хранятся на бумажном носителе или получены впервые в ходе эксперимента (в тех случаях, когда лабораторные исследования не позволяют сохранять данные автоматически), наиболее практичным является их трансформация в электронный формат при помощи табличного процессора с последующим импортом в R. Внесение первичных данных в среду программирования напрямую довольно неудобно, хотя возможно. Так же в среде программирования R пользовательские объекты весьма недолговечны и могут быть в любой момент утрачены в результате случайного сбоя. Поэтому первичные данные рекомендуется хранить в виде файлов csv, excel, либо в базах данных.

Наиболее простым способом является импорт файлов с помощью соответствующего меню интерфейса RStudio, а именно ImportDataset, расположенным в правом верхнем углу. Данный инструмент позволяет загружать данные из файлов формата xlsx. Недостатком способа является то, что при каждой новой

загрузке приходится вручную «прокладывать путь» к нужному файлу, что крайне неудобно при необходимости частых загрузок.

В ситуации когда регулярно приходится обрабатывать данные по одному и тому же алгоритму, например, готовить отчёт о продуктивности животных или о движении сырья в процессе производства продуктов питания удобно один раз прописать путь к нужной папке в коде, меняя лишь название файла. В литературе распространено описание загрузки данных в среду программирования R при помощи функций семейства *read()* [Мастицкий, Шитиков, 2015; Peng, 2022]. Преимущество данного способа в том, что функции этого семейства являются базовыми и не требуют установки дополнительных библиотек. Обычно в рекомендациях по импорту данных рекомендуется использовать функции

На практике наиболее востребованными оказываются функции *read.csv()* и *read.csv2()*. Но их использование у начинающих пользователей сопряжено с рядом «подводных камней».

1) Недостаточная квалификация начинающих пользователей при работе с csv-файлами. Обе функции импортируют данные в среду программирования R из файлов формата *csv*. Открытие файлов такого формата в табличном процессоре MicrosoftExcel не всегда возможно. Трансформация листа MicrosoftExcel в файл формата csv в ряде случаев делает файл нечитаемым в среде программирования R. Поэтому для создания csv-файлов и их последующей загрузке в R лучше пользоваться табличными процессорами OpenOfficeCalc или LibreOfficeCalc, которые безо всяких проблем читают csv-файлы. При создании csv-файла в данных программах в качестве кодировки рекомендуется выбирать UTF-8, а в качестве разделителя столбцов `;`.

2) Достаточно большое количество аргументов, которе приходится учитывать. Первый аргумент функций *read.csv()* и *read.csv2()* называется *file = " "* , который показывает, какой именно файл нужно импортировать. В кавычках пишется путь к файлу, причём это может быть как адрес на локальном жёстком

диске, так и ссылка на интернет ресурс. При необходимости загрузки нескольких файлов из одной папки её можно обозначить в качестве рабочей директории при помощи функции *set.wd()*. Далее следует аргумент *sep = " "*, обозначающий разделитель столбцов, которыми чаще всего бывают `,` или `;`. Следующий аргумент *dec = " "*, или разделитель десятичных знаков, которыми зачастую оказываются `,` или `.` Неправильный выбор разделителя десятичного знака приводит к некорректной загрузке таблицы в среду R. В случае загрузки csv-файла с кодировкой UTF-8 рекомендуется прописывать в коде аргумент *file.encoding = " "*.

3) Искажение типов данных при загрузке. Встречается довольно редко, но данное обстоятельство нужно иметь ввиду. Причиной может быть несоответствие выбранного разделителя десятичных знаков фактическому. Следует запомнить, что сохранённые записи в формате Date в табличном процессоре автоматически преобразуются в тип данных character в среде программирования R.

4) Искажение названий столбцов, содержащих символы, отличные от букв и цифр.

5) Искажение записей на кириллице — может возникнуть, если при создания csv файла выбрать кодировку, не относящуюся к UTF-8.

В качестве примера рассмотрим загрузку каталога быков компании ABS за декабрь 2012, исходный файл получен из открытого источника[25]. В качестве хранилища таблиц для загрузки в среду программирования R использовали общедоступный сервис на яндекс диске.[26] Однако для работы все хранящиеся там таблицы необходимо сохранить на локальный компьютер.

```
ABS <- read.csv("/путь к папке/ABS_202212.csv", sep = ",", dec = ".", 
header = TRUE)
head(ABS[1:5])
  Оцененный.или.геномный Hornless ферма.STJacobs Х..Ч.Б Х..окраса
1                Genomic       No             No  BLACK     50-74
2                Genomic       No             No  BLACK     25-49
3                Genomic       No             No  BLACK     50-74
4                Genomic       No             No  BLACK      0-25
```

25 https://absbullsearch.absglobal.com/BullList?VisibilityCountryCode=RUS&BreedTypeCode=D&BreedCode=HO&ProofCode=USA&comp=&o=NetMerit|true

26 https://disk.yandex.ru/d/kAA883JxY55RTQ

```
5                Genomic       No             No  BLACK    75-100
6                Genomic       No             No  BLACK     25-49
```

Так же для импорта данных используются такие функции, как *read.table()* или *read.delim().* Функция *read.table()* хорошо подходит для импорта исходных данных, хранящихся в txt-файлах. Для примера загрузим в *R* результаты испытания на резвость лошадей, полученных на новосибирском ипподроме[27].

```
df <- read.table("/Путь к папке/Horse.txt", header = FALSE)
> df
  V1  V2   V3  V4     V5   V6       V7     V8           V9  V10      V11 V12
1  1 (1) 13.0 (2)  НОВОГ 1600   САНДРА 2.21,2   к.к.р.2020 (бк)  Асташин   С
2  2 (2) 13.2 (2)  КУЛЕШ 1600 ДОБРОДАР 2.17,8 т.с.ж.о.2021  (1) Проценко   П
3  3 (3) 13.4 (2) ПРИЛКЗ 1600 СЕРПЯНКА 2.19,5   г.к.р.2022  (1) Проценко   П
4  4 (4) 14.0 (2)  ВНИИК 1600     ДАНА 2.12,1   с.к.о.2019  (м)    Егоян   К
5  5 (5) 14.2 (2)   ШИШК 1600  КОСПЛЕЙ 2.25,4   с.ж.о.2022 (бк)    Шубин   Д
```

Обычно, работа с первичными данными осуществляется в табличных процессорах типа MicrosoftExcel и его аналогах, например LibreOfficeCalc. В связи с этим, для многих пользователей актуальным является прямое взаимодействие среды программирования R с файлами форматов xls и xlsx, то есть поддерживаемыми табличными процессорами. Для этого существуют специальные библиотеки, например *openxlsx*[28].

Преимуществами такого подхода являются:

1) Непосредственный импорт/экспорт из табличного процессора.

2) Отсутствие необходимости использования ряда дополнительных аргументов.

Наиболее прогрессивным способом обработки данных в R является непосредственный импорт из баз данных. Такой подход особенно хорош, когда с одним и тем же набором исходных данных на постоянной основе работает несколько человек. Для импорта данных непосредственно из базы существуют специальные библиотеки, например, *RmySQL* [29].

27 https://ippodromnsk.ru/rezultaty-begovogo-dnya-1-2025/
28 https://ycphs.github.io/openxlsx/index.html
29 https://www.projectpro.io/data-science-in-r-programming-tutorial/r-tutorial-importing-data-from-relational-database

При необходимости импорта в среду R фрагмента кода можно воспользоваться функцией *source("file").* В качестве аргумента используется путь к файлу, который принимается как в txt так и в R формате. Таким образом удобно загружать пользовательские функции. Для примера загрузим функцию, которая вычисляет t-критерий Стьюдента по данным средни арифметических и и их ошибок по двум выборкам.

```
> source("/Путь к папке/Student_test_caclulator.R")
> Student(100, 110, 10, 11)
[1] 2.182179
```

7.2.Экспорт данных

В процессе обработки данных в среду программирования R обычно загружают «сырые» записи. На выходе обычно получают обработанные таблицы, в частности показатели описательной статистики. Для примера рассмотрим экспорт данных описательной статистики, полученный при обработке данных комплексного индекса оценки племенной ценности быков TPI [Cole, VanRaden, 2018; Vanraden, 2002], генетического потенциала прибавки по удою (Milk), жиру % (Fat) и белку % (Protein) согласно данным каталога быков ABS за декабрь 2022 года. Анализ будем проводить при помощи нашей функции *opstat*.

```
TPI <- round(opstat(ABS$TPI.))
Milk <- round(opstat(ABS$Milk))
Fat <- round(opstat(ABS$Fat..), 2)
Prot <- round(opstat(ABS$Ptot..), 2)
df <- cbind(TPI, Milk, Fat, Prot)
colnames(df) <- c("TPI", "Milk", "Fat", "Prot")
df
      TPI  Milk    Fat   Prot
n     257   257 257.00 257.00
Mean 2722   871   0.16   0.06
SE     11    38   0.01   0.00
SD    173   609   0.12   0.04
Med  2751   923   0.17   0.06
Min  2116 -1314  -0.22  -0.06
Max  3078  2383   0.44   0.19
Q1   2605   449   0.08   0.03
Q3   2838  1316   0.25   0.09
```

Таблица *df* в данном случае является наглядным примером обработанных таблиц, которые необходимо вставлять в отчёты, статьи и т.д. Для экспорта таких таблиц из среды R существует несколько подходов, которые, как правило основаны на функциях семейства *write*(x, file = " ", row.names =), где x — объект среды R, *file = " "* - имя и путь к файлу, в которые планируется экспортировать объект x . Аргумент *row.names* = настраивает экспорт имён строк. По умолчанию, *row.names = TRUE*, в таком случае имена строк, в том числе их номера, будут перенесены в экспортируемый файл, что может вызвать сдвиг шапки таблицы на 1 столбец влево. Если *row.names = FALSE*, номера строк не экспортируются.

В базовой конфигурации среды R имеется функция *write.csv()*, осуществляющая экспорт таблиц в csv файлы. Не очень удобна для пользователей Windows, так как требует установку офисных приложений OpenOffice или LibreOffice для работы с csv файлами. Для импорта в файлы формата xlsx существует функция *write.xlsx()*, которая встроена в пакеты *openxlsx* и *xlsx*. Пример кода выгрузки таблицы df будет выглядеть следующим образом.

```
library(openxlsx)
write.xlsx(df, file = "/Путь к папке/Opstat_ABS.xlsx")
```

в данном случае, аргумент file означает путь к папке, куда будет выгружен файл, а фрагмент Opstat_ABS.xlsx — имя экспортируемого файла.

Экспорт отклика функции в консоль можно осуществить при помощи функции *sink()*. Рассмотрим на примере отклика функции head() применительно к встроенной таблице *Orange*.

```
sink("/Путь к папке/Orange_head.txt") #Создаём файл для выгрузки
head(Orange)# выполняем действия для получения необходимого для выгрузки 
результата
sink()# закрывем файл для выгрузки
```

Практические задания:

1. Скачайте таблицу alta_202408.xlsx[30] из папки на Яндекс диске (https://disk.yandex.ru/d/kAA883JxY55RTQ), преобразуйте в файл *csv* при помощи табличного процессора LibreOfficeCalc, импортируйте в среду программирования R при помощи функции *read.csv()*.
2. Сформировать текстовый документ в формате txt согласно Рисунок 5. Импортируйте в R при помощи функции *read.table()*. Присвойте имена столбцам: «Место на финише», «Происхождение», «Владелец», «Жокей»
3. Из папки на Яндекс диске (https://disk.yandex.ru/d/kAA883JxY55RTQ) импортируйте в R файл *semex_202212*[31] используя функцию *read.xlsx()*.
4. Из папки на Яндекс диске (https://disk.yandex.ru/d/kAA883JxY55RTQ) импортируйте в R файл *cat_traits*[32]. По данным столбца *Age* составьте таблицу, отражающую долю животных разного возраста (%). Выгрузите итоговую таблицу в формате xlsx.
5. Используя *source()* импортируйте файл с пользовательской функцией *opstat*.
6. Выгрузите результат функции *source()* из предыдущего задания в txt-файл используя функцию *sink()*
7. Загрузите таблицу food_kaggle[33], оставьте только строки (Category), соответствующие продуктам из свинины (PORK), говядины (BEEF) и курицы (CHICKEN)

30 Каталог быков-производителей компании AltaGenetics, скачан из открытого источника https://bullsearch.altagenetics.com/us/BS/List

31 Каталог быков компании semex за декабрь 2022 года. Скачан из данных в свободном доступе https://www.semex.com/us/i?lang=en&page=home

32 Набор данных, полученный Эмирханом Булутом. Предоставляет собой таблицу с данными о различных характеристиках кошек в разных регионах. Скачан из открытого источника: https://www.kaggle.com/datasets/emirhanai/global-street-cats-statistics-for-ai-research?resource=download

33 Набор данных о питательности пищевых продуктов, скачан из открытого доступа https://www.kaggle.com/datasets/shrutisaxena/food-nutrition-dataset

1	СЕРПЯНКА	2.19,5	36,5	г.к.р.2022	Роквелл – Специя	Бабичев П.А.	(1) Проценко П
2	ЛОРД КРАИН	2.20,5	36,7	г.ж.а.2022	Энс Кавиар'с Сан — Либерти	Стремоухов В.В.	(3) Стремоухов В
3	КАРМЕН	2.21,4	36,9	г.к.ф.2022	Репаратор — Классика	Одеянко В.Б.	(бк) Илюхин С
4	ЗОЛОТОЙ	2.23,7	37,2	т.г.ж.р.2022	Ладди — Зеста	Курицын Б.М.	(бк) Курицын Б
5	ЛАФИНИЯ	2.48,3	41,3	г.к.р.2022	Фаберже АШ — Лапландия	Перелыгин И.Е.	(бк) Перелыгин И
	БЕБИ ЛАЙТ	снят		г.к.а.2022	Ларссон Ти Джей — Басма	ЗАО ПЗ «Медведский»	(м) Поляков А

Рисунок 5 Результаты третьего заезда бегового дня № 1 (год 2025) на Новосибирском ипподроме[34]

8. Списки

Списки в языке программирования R — это комплексные структурированные объекты, состоящие из упорядоченной коллекции элементов, представляющие собой одномерные гетерогенные структуры данных. Если говорить проще, список — это универсальный "контейнер", который может хранить внутри себя любые другие объекты, даже другие списки. Список может состоять из векторов, матриц, символов, функций и даже других списков. Основное преимущество работы со списками состоит в гибкости и способности объединять объекты любого типа и размера. Список в R создаётся с помощью функции *list()*. Объединение списков так же происходит посредством функции *list()*.

```
my_list <- list(
  Producrion = Production,#1
  my_mat = matrix(data = LETTERS, 4,4),#2
 v1 = 1,#3
  v2 ="GREEN",#4
 v3 = TRUE,#5
 v4 = c(1:15),#6
  list(frut = c("apple", "orange", "banana", "cherry")),
       Bulls = Bulls))#7)
```

34 https://ippodromnsk.ru/ispitaniya/

Доступ к компонентам списка R двумя способами. Первый из них, посредством адресации на номер соответствующего компонента при помощи оператора извлечения []. Другой способ похож на адресацию столбца таблицы с использованием $. Получить доступ к элементам подсписка возможно используя код [[N]][n].

```
my_list[1]
$Producrion
     ID  Milk  Fat Protein
1  1121  9668 4.11    3.22
2  2023  9982 4.73    3.45
3  4120  9628 5.40    3.35
4   918  9118 4.36    3.70
5  5417  9664 4.65    3.55
6  4578 10212 4.87    3.40
7  5647  8195 4.32    3.29
8  8954  8680 4.11    3.46
9  1452  7944 4.99    3.60
10 5874  8274 4.11    3.28
```

```
my_list$Producrion
     ID  Milk  Fat Protein
1  1121  9668 4.11    3.22
2  2023  9982 4.73    3.45
3  4120  9628 5.40    3.35
4   918  9118 4.36    3.70
5  5417  9664 4.65    3.55
6  4578 10212 4.87    3.40
7  5647  8195 4.32    3.29
8  8954  8680 4.11    3.46
9  1452  7944 4.99    3.60
10 5874  8274 4.11    3.28
```

```
my_list[[7]][2]
$Bulls
      Number                 FullName Country       Date
1 3263337318  PEAK CH ALTAINSPIRE-ET     USA 2023-02-12
2 3251555942 PEAK ALTAZEROGRAVITY-ET     USA 2023-03-24
3 3264522457 PEAK ALTAVORTEXVAULT-ET     USA 2023-02-21
4 3269404461   PEAK ALTATURBOTIME-ET     USA 2023-03-23
5   12948857      PEAK ALTAKLAEBO-ET     CAN 2017-10-25
```

Ещё один пример адресации

```
#Создадим лист с баллами оценок студентов
> Student <- list(
+   student1 = list(
+     name = "Екатерина Вилкова",
+     grades = c(88, 92, 85),
+     subjects = c("Math", "Science", "English")
+   ),
+   student2 = list(
+     name = "Иван Медведев",
+     grades = c(95, 89, 91),
+     subjects = c("Math", "Science", "English")
+   )
+ )
> # Выводим информацию
> ivan_grades <- students$student2$grades
> ivan_grades
[1] 88 92 85
```

Практические задания:

1. Загрузите файл Project_list1.txt[35]. Извлеките элемент с датой.
2. Создайте список со следующими элементами:

   weight — вектор, имитирующий массу 100 поросят-отъёмышей от 4 до 8 кг;

   father - вектор, имитирующий клички хряков: "Дубок", "Восточный", " Сибиряк"

   mother - вектор, имитирующий индивидуальные номера матрей от 1 до 8.
3. Загрузите файл Project_list1.txt. Извлеките второй столбец матрицы
4. Создайте матрицу размером 5x5 со случайными числами от 1 до 100: matrix_data <- matrix(sample(1:100, 25), nrow = 5). Напишите код, который: Найдет сумму элементов в каждом столбце. Найдет сумму элементов в каждой строке. Найдите максимальный и минимальный элементы в матрице. Преобразует матрицу в список, где каждый элемент списка — это строка матрицы.

9.Управляющие конструкции языка R. Условия и циклы

Для решения тех или иных задач пользователю нередко приходится прибегать к повторению действий n-е количество раз. Например в теории вероятности существует классическая задача по вытаскиванию разноцветных шариков из коробки. В языке R для решения подобной задачи применяются так называемые циклами. Цикл — разновидность управляющей конструкции в высокоуровневых языках программирования, предназначенная для организации многократного исполнения набора инструкций. Циклы не являются функциями! К классическим или явным циклам относятся *for, while*, и *repeate*, являющихся частью ядра языка программирования R [Bloomfield, 2014]. Хороший обзор

35 https://disk.yandex.ru/d/N9xkmdsklgCbWw

использования циклов представлен в обучающих материалах Алексея Селезнёва[36]. Цикл *while* выполняется до тех пор, пока условие цикла выполнимо. Как только условие цикла перестаёт выполнятся, он прекращает свою работу. Это может быть, например, обработка данных до тех пор, пока не будет достигнут некоторый критерий сходимости в итеративном процессе или обработка потока данных до тех пор, пока не закончится поток. Рассмотрим пример использования цикла *while* в R.

```
counter <- 0
target <- 10
while (counter < target) {
  counter <- counter + 1
  print(paste("Текущее значение счетчика:", counter))
}
```

Цикл *repeat* — это бесконечный цикл, выход из которого осуществляется только с помощью оператора *break.* Этот цикл используется реже всего из всех трёх. Он обычно применяется в специфических ситуациях, когда требуется непрерывная итерация с условием выхода внутри цикла, часто в сочетании с *if u break.* Большинство задач, которые можно решить с помощью *repeat, м*ожно переписать с использованием *while.* Циклы *while* и *repeat* используются значительно реже, чем *for,* и их применение обычно ограничивается ситуациями, где векторизация невозможна или неэффективна. В подавляющем большинстве случаев, разработчики R предпочитают использовать векторизованные операции или функции из семейства apply, чтобы повысить производительность кода. В связи с этим из всех явных циклов языка R подробно описан будет лишь цикл *for.*

9.1. Цикл for

Цикл ***for*** является наиболее распространённым из них. Цикл ***for*** в языке программирования R полезен для выполнения итераций по элементам списка, фрейма данных, вектора, матрицы или любого другого объекта. Это означает,

36https://bookdown.org/selesnow/iterations_in_r_course/. 2022.

что цикл *for* может использоваться для многократного выполнения действия или группы действий в зависимости от количества элементов в объекте. в этом цикле сначала проверяется тестовое условие, затем выполняется тело цикла [Braun, 2021]. Тело цикла не будет выполнено, если тестовое условие неверно Рисунок 5. Цикл *for* автоматически прекращает работу после завершения обработки всех данных в объекте. Каждое действие по обработке отдельно взятой записи называется итерацией, а сам набор исходных данных итерируемым объектом.

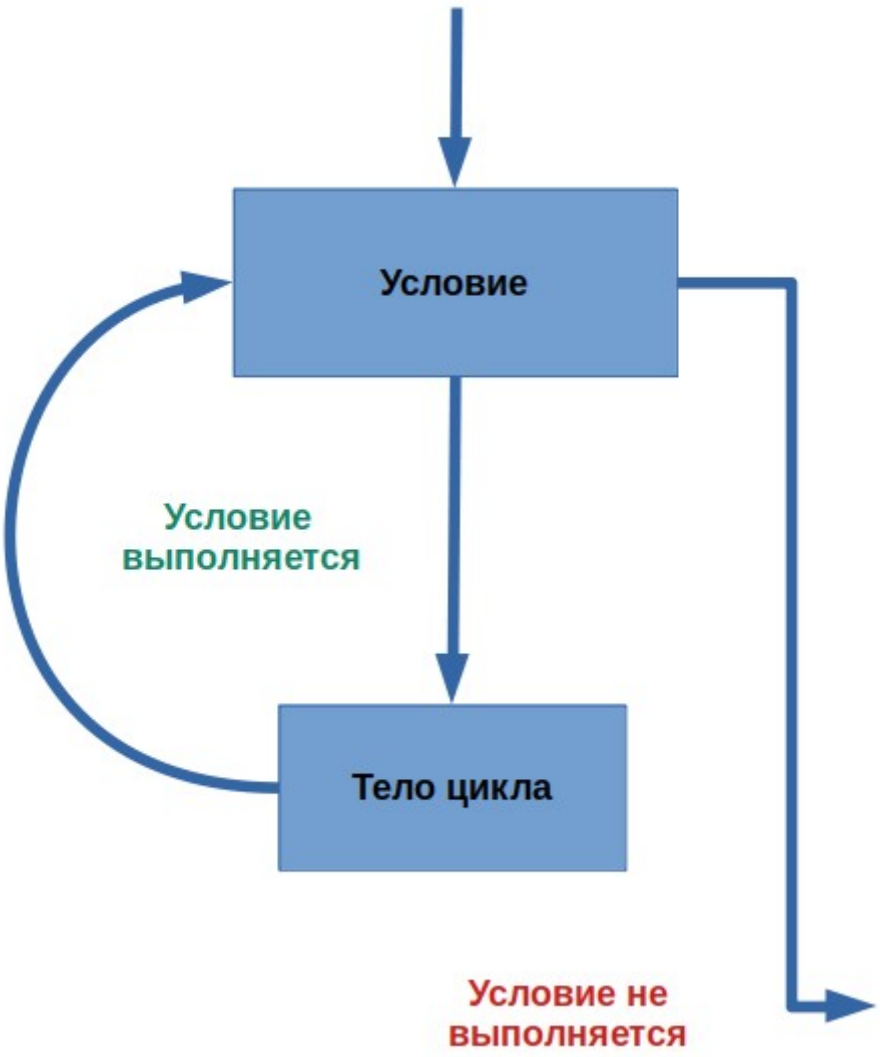


Рисунок 5 Схема функционирования цикла *for*

Для наглядности разберём простейший пример использования цикла *for.* Например, в ходе калибровки приборов было обнаружено, что анализатор молока выдавал систематическую погрешность на 0.01. Цикл *for* поможет её исправить.

```
fat <- c(4.00, 4.12, 3.92, 3.96, 4.05)
for(i in fat){
  print(i + 0.01)
}
[1] 4.01
[1] 4.13
[1] 3.93
[1] 3.97
[1] 4.06
```

Запись в круглых скобках `i in fat` называется условием цикла. В данном случае, под i понимается итератор, то есть составляющая часть объекта, с которой происходит итерация. Обычно, число итераций равно количеству итераторов в итерируемом объекте. Таким образом итерируемый объект, в данном случае вектор *fat*, содержит записи с исходными данными. Запись внутри фигурных скобках называют телом цикла.

При работе с циклами нужно учитывать некоторые особенности. Если вам требуется, что бы результат работы цикла отображался в консоле, необходимо использовать функцию *print()* в теле цикла. Для наглядности, уберём функцию print из предыдущего примера.

```
for(i in fat){
+ i + 0.01
+ }
```

Отчёта об ошибки кода нет, но и результат в консоли не отобразился. Для записей результатов действия цикла для начала необходимо создать пустой объект, который затем помещают в тело цикла с присвоением результата функции. Такой подход получил название «метод пустого контейнера».

```
# 1. Создаём пустой вектор ТОЧНОГО размера, который нам понадобится
results <- vector("numeric", length = length(fat))
```

```
# 2. Запускаем цикл и ЗАПОЛНЯЕМ ячейки, а не "растим" вектор
for (i in seq_along(fat)) {
  results[i] <- fat[i] + 0.01
}

# Печатаем результат
# 1. Создаём пустой вектор ТОЧНОГО размера, который нам понадобится
results <- vector("numeric", length = length(fat))

# 2. Запускаем цикл и ЗАПОЛНЯЕМ ячейки, а не "растим" вектор
for (i in seq_along(fat)) {
  results[i] <- fat[i] + 0.01
}

# Печатаем результат
print(results)
```

Язык программирования R позволяет использовать один цикл внутри другого цикла. В качестве примера рассмотрим случай, описанный в открытом источнике [37]

```
z <- NULL
for(i in 1:5) {
  for(j in 2:4)  {
    z = c(z, i^i)
  }
}
z
```

Поскольку мы используем два цикла *for*, нам нужны две “фиктивные” переменные, например i и j (будьте осторожны: у каждого цикла *for* должна быть своя отличительная фиктивная переменная). В первом цикле for указано, что для i от 1 до 5 мы хотим выполнить определённую операцию. Второй, вложенный цикл *for*, говорит о том, что мы хотим выполнить определённую операцию для j от 2 до 4 (как вы можете видеть, цикл for не обязательно должен начинаться с цифры 1). С помощью z = c(z, i^ j) мы указываем, что на конкретной итерации цикла мы хотим добавить новое число к переменной z, а именно j-ю степень числа i. Итак, во-первых, i присваивается значение 1, а число 1 используется во вложенном цикле *for*, где берутся его вторая, третья и четвёртая степе-

37Fiřtová L. For cycles in R language.n.d.; URL https://exceltown.com/en/tutorials/r-script/for-cycles-in-r-language/

ни и добавляются к переменной z. Затем i присваивается значение 2, и снова число 2 используется во вложенном цикле for, где его вторая, третья и четвёртая степени берутся и добавляются к z. Это повторяется для i, равного 3, 4 и 5.

9.2.Использование цикла for при работе с таблицами

Итерация по столбцам: При работе с циклами *for* и объектами класса *data.frame* в R важно понимать, как происходит обращение к столбцам. Вот ключевые особенности. По умолчанию, при передаче data.frame в цикл *for*, итерации выполняются по столбцам фрейма данных.

```
df <- data.frame(a = 1:3, b = letters[1:3])
for (col in df) {
  print(col)
}
# Выведет:
# [1] 1 2 3
# [1] "a" "b" "c"
```

При использовании цикл for возможно использовать адресацию через имя столбца (если оно известно). Обратите внимание на использование двойных квадратных скобок [[ ]] для извлечения столбца как вектора, а не подмножества data.frame. В таком случае, чикл будет работать с каждым подмножеством столбца.

```
for (i in seq_along(df)) {
  print(df[[i]])
 }
[1] 1 2 3
[1] "a" "b" "c"
```

Аналогичным будет результат при использовании имени столбца:

```
for (col_name in names(df)) {
  print(df[[col_name]])
}
[1] 1 2 3
[1] "a" "b" "c"
```

Для доступа к столбцам внутри цикла предпочитайте df[[col_name]] вместо df$col_name

1. Используйте seq_along(df) или names(df) для безопасной итерации

2. Помните, что for работает со столбцами, а не со строками (по умолчанию)

В качестве набора исходных данных для ряда функций, например функций вычисления показателей простейшей описательной статистики (mean, median, sum и тд), критериев нормальности и функци семейства *as* могут быть использованы только одномерные объекты (векторы, столбцы). Цикл *for* можно использовать для масштабирования функций типа *mean* на несколько столбцов таблицы iris.

```
for(col in iris[1:4]){
  print(mean(col))
}
[1] 5.843333
[1] 3.057333
[1] 3.758
[1] 1.199333
```

Однако для сохранения результатов масштабирования функции вектор применяется следующий подход. Создадим таблицу df

```
df <- data.frame(
  A = c(1, 2, 3, 4),
  B = c(5, 6, 7, 8),
  C = c(9, 10, 11, 12)
)
```

А теперь вычислим средние значения каждого столбца в виде вектора.

#Инициализируем вектор для хранения средних значений
mean_values <- numeric(ncol(df))

```
# Цикл for для расчёта средних значений
for (i in 1:ncol(df)) {
  mean_values[i] <- mean(df[[i]])
}
print(mean_values)
[1]  2.5  6.5 10.5
```

Итерация по строкам:

В этом случае цикл перебирает каждую строку data.frame как отдельный объект. Обычно это делается с помощью функции *nrow()* для определения количества строк и индексации:

```
my_dataframe <- data.frame(a = 1:5, b = letters[1:5])
for (i in 1:nrow(my_dataframe)) {
  current_row <- my_dataframe[i, ]
```

```
  # Действия с текущей строкой, например:
  print(paste("Строка", i, ":", current_row$a, current_row$b))
}
```

Итерация по элементам:

Для перебора каждого элемента data.frame можно использовать вложенные циклы:

```
for (i in 1:nrow(my_dataframe)) {
  for (j in 1:ncol(my_dataframe)) {
    current_element <- my_dataframe[i, j]
    # Действия с текущим элементом
    print(paste("Элемент в строке", i, "и столбце", j, ":", current_element))
  }
}
```

Важные замечания

- Циклы for в R могут быть медленными для больших data.frame
- Векторизованные операции обычно более эффективны
- Для сложных операций над строками лучше использовать пакеты dplyr или data.table

9.3.Бутстрап как пример использования цикла for

Одной из проблем в научных исследованиях является малочисленность выборок фактических данных. Одним из решений проблемы является бутстрап-расширение выборки [Rousselet, Pernet, Wilcox, 2023]. Его принцип основан на многократном рандомном извлечении фактических данных из реальной выборки с указанным количеством повторов [Hesterberg, 2011]. Это как если бы из коробки с разноцветными шариками 1000 раз нужно было бы вытащить случайный шарик. При этом на основании результатов случайных вытаскиваний шарика каждый раз создаётся запись, какого цвета шарик был извлечён. Таким образом создаётся бутстрап выборка. Одним из решений по использованию бутстрап-расширения выборки является использование цикла *for* [Мастицкий, Шитиков, 2015]. Преимуществом такого подхода к бутстрап-расширению является независимость пользователя от сторонних библиотек. Для примера рассмотрим случай рандомного извлечения чисел от 1 до 10, которое повторяли 1000 раз. Кратность (1000 в данном случае) называется

количеством бутстрап-повторов. Пример, если нужно 1000 средних значений из 1000 выборок:

```
a <- 1:10
# Создаём контейнер для хранения средних
boot_means <- vector("numeric", length = 1000)
# Цикл для 1000 итераций
for(i in 1:1000){
  # 1. Создаём одну бутстрап-выборку
  current_sample <- sample(a, length(a), replace = TRUE)
  # 2. Считаем её среднее и сохраняем в i-ю ячейку
  boot_means[i] <- mean(current_sample)
}
# Теперь у нас есть 1000 средних значений
head(boot_means)
```

В данном случае необходимо для начала создать два вектора. Один из них пустой для записи конечного результата. Содержание другого не имеет значение, важно, что бы его длина была равной количеству итераций, или бутстрап-повторов. Именно этот вектор используется в качестве итерируемого объекта. Важно, что при использовании функции *sample ()* в теле цикла вторым аргументом так же указывается количество итераций. Третьим аргументом функции *sample ()* пишется replace = TRUE. Это означает, что мы допускаем количество бутстрап-повторов, превышающее объём исходной реальной выборки.

9.4.Управляющая конструкция *if() {}else{}* внутри цикла *for*

В ряде случаев необходимо что бы выполнение действия менялось в зависимости от условий. То есть, в случае не выполнения условия цикла его работа не прекращалась бы, а выполнялось альтернативное действие. Для таких случаев существует конструкция

```
if(){}else{}
```

которая пишется в теле цикла Рисунок 6. В круглых скобках пишется условие выполнения действия, в фигурных скобках до оператора else пишется само действие, после – альтернативное действие. В качестве примера рассмотрим присвоение категории «чётное» или «нечётное» ряду чисел.

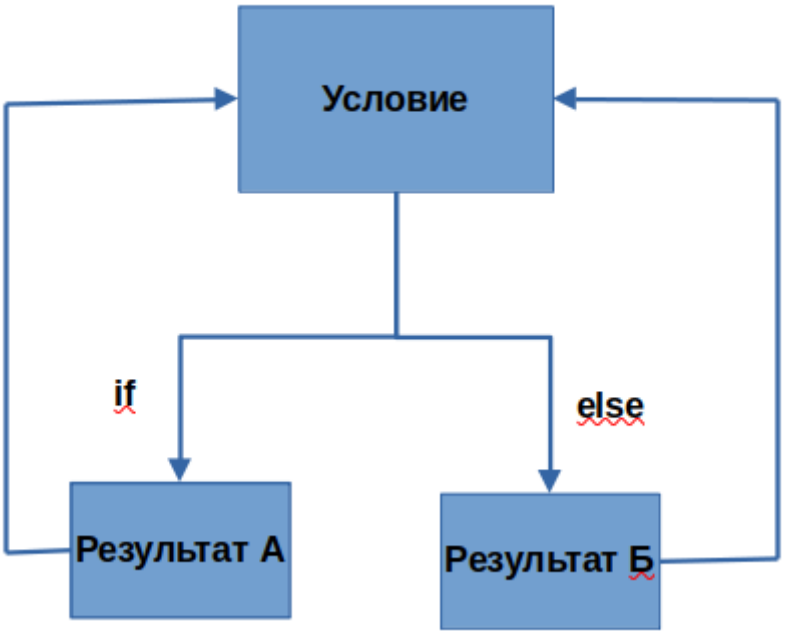


Рисунок 6 Принцип работы управляющей конструкции *if() {}else{}*

```
z <- 1:20
> z
 [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20
> if(z %% 2 == 0){print("ЧЁТ")}else{print("НЕЧЁТ")}
Ошибка в if (z%%2 == 0) { : условие длиной > 1
> for(i in z){
+   if(i %% 2 == 0){print("ЧЁТ")}else{print("НЕЧЁТ")}
+ }
[1] "НЕЧЁТ"
[1] "ЧЁТ"
[1] "НЕЧЁТ"
[1] "ЧЁТ"
```

В примере показано, что конструкция *if() {}else{}* работает только применимо к единичным векторам. Одним из способов векторизации такой управляющей конструкции является встраивание её в тело цикла *for.* В круглых скобках пишется условие выполнения действия, в фигурных скобках до оператора *else* пишется само действие, после – альтернативное действие. В животноводстве примером использование может быть присвоение назначения использования животного в зависимости от его продуктивности. Для этого создадим таблицу animals_production, содержащую данные об удое за 305 дней.

```
animals_production <- data.frame(
  ID = runif(min = 100, max =  9999, n = 20),
  Milk = runif(min = 5000, max = 12000, n = 20)
)
animals_production <- round(animals_production)
```

Присвоим животному категорию основываясь на данных таблицы

```
for(i in animals_production$Milk){
  if(i < 9000){
    print("Брак")
  }else{
    print("Племенное_использование")
  }
}
[1] "Брак"
[1] "Племенное_использование"
[1] "Брак"
[1] "Брак"
[1] "Племенное_использование"
[1] "Брак"
```

Обращаем внимание, что конструкция *if(){}else{}* будет работоспособной только, если операторы *if* и *else* находятся на одной и той же строке, что и сами операторы *{}*. При необходимости введения дополнительных условий в конструкцию *if(){}else{}* встраивают оператор *else if*, обозначающий условие выполнения альтернативного действия. Оператор *else{}* в таком случае содержит информацию о действии, совершаемом при невыполнении ни одного из условий. Для примера рассмотрим случай, когда коровам с продуктивностью от 7000 до 9999 кг за 305 дней лактации присваивают категорию «Товарное использование использование», 10 000 и более – «Племядро», 5999 и менее – «Брак».

```
for(i in animals_production$Milk){
  if(i > 9999){
    print("Племядро")
  }else if(i <= 9999 & i >= 7000){
    print("Товарное_использование")
  }else{
    print("Брак")
  }
}
```

В таком случае существует проблема сохранения результата в вектор, либо столбец таблицы. Решение довольно непростое, но оно есть. Заключается оно запись вывода в консоль в текстовый файл. Далее в первой строке текстового файла удаляем "[1]" тестовом фале (txt). Далее в остальных строках

текстового файла меняем "[1]" на ",". Затем в текстовом файле пишем код присвоения таким образом, что бы значения из консоли попали внутрь скобок который затем преобразуется в вектор.

```
sink("/home/oem/Документы/Учебные пособия/timeput.txt")
for(i in animals_production$Milk){
  if(i < 9000){
    print("Брак")
  }else{
    print("Племенное_использование")
  }
}
sink()
source("/home/oem/Документы/Учебные пособия/timeput.txt")
```

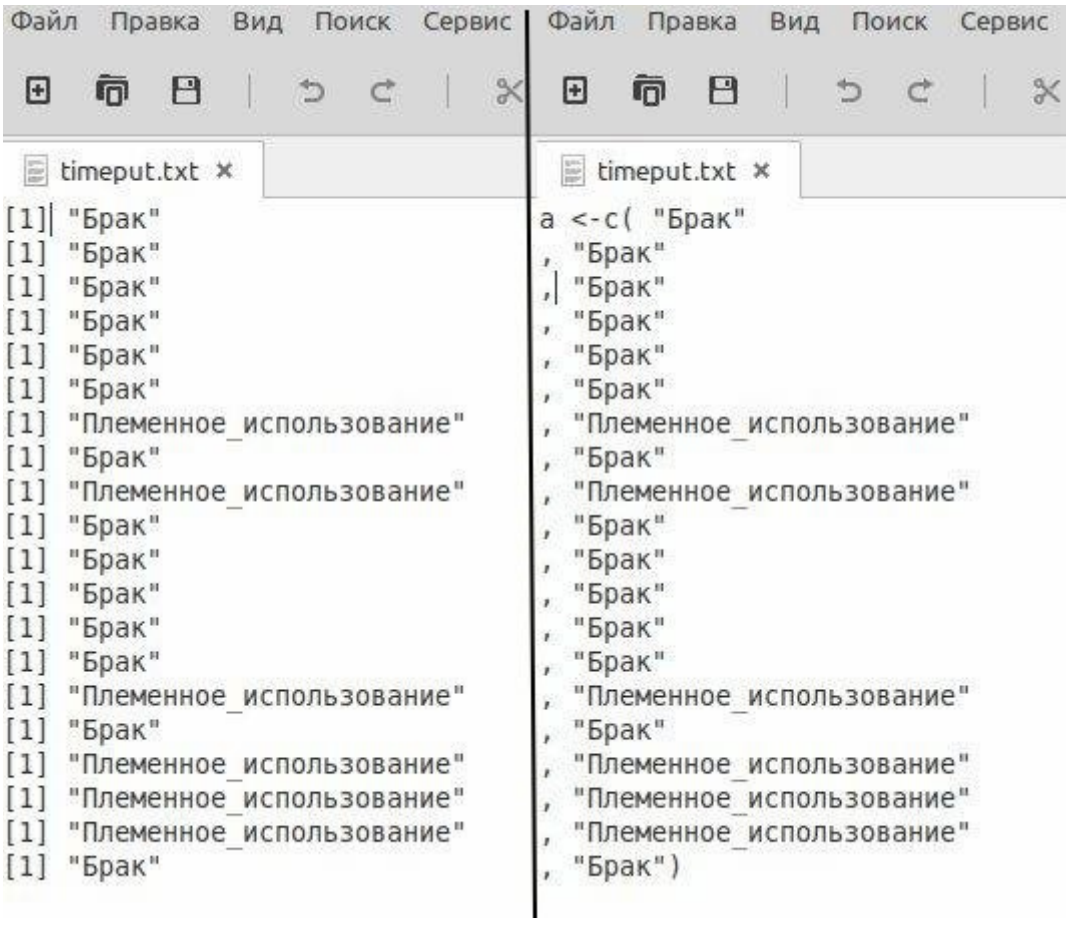


Рисунок 7 Схема трансформации текстового файла, содержащего записи выгрузки из цикла *for*. Слева — выгрузка из цикла, справа — конечный вид файла.

9.5. Операторы *next* и *break*

Оператор **break** полностью останавливает выполнение цикла и передаёт управление коду, следующему за циклом. Например, нам нужно, что бы цикл прекратил работу, если встретит в итерируемом объекте число, кратное 4.

```
for (i in 1:10) {
+   print(i)
+   if (i %% 4 == 0) {break}
+ }
[1] 1
[1] 2
[1] 3
[1] 4
```

В животноводстве такой код можно использовать при вычислении средней арифметической, если мы предполагаем наличие не валидных данных. Например, было условленно, что если жирность молока меньше 3.5% или больше 5% данные следует считать невалидными. В таком случае код прекратит работу и мы не получим никакого результата. Однако в таком случае лучше будет воспользоваться оператором *next*.

Оператор 'next'в R используется для пропуска оставшихся операторов в цикле и продолжения выполнения программы. Другими словами, это оператор, который пропускает текущую итерацию без завершения цикла.

'**next**' — это оператор управления циклом,ы который работает противоположно оператору *break*, вместо того, чтобы завершать цикл, он заставляет выполнить следующую итерацию цикла. В R оператор *next* — это оператор управления, который позволяет перейти к следующей итерации цикла без выполнения каких-либо операторов, которые все ещё находятся в цикле для текущей итерации. Чтобы обработать или пропустить определённые случаи или обстоятельства внутри цикла, эта фраза часто используется в сочетании с условными операторами (например, операторами *if*). Например, что бы из вектора извлечь только чётные числа можно воспользоваться циклом *for* с операторм next. Обратите внимание, что если входные данные не соответствуют условию, на выходе вме-

сто них цикл выдаёт NA. Такой приём можно использовать для фильтрациии исходных данных в зоотехнии, биологии и пищевой промыщленности.

```
> a <- 1:20
> b <- c()
> for(i in a){
+    b[i] = c(if(i %% 2 == 0){print(i)}else{next})
+ }
[1] 2
[1] 4
[1] 6
[1] 8
[1] 10
[1] 12
[1] 14
[1] 16
[1] 18
[1] 20
> b
 [1] NA  2 NA  4 NA  6 NA  8 NA 10 NA 12 NA 14 NA 16 NA 18 NA 20
```

Контрольные вопросы к разделу

1. Какой общий термин носят циклы *for, while* и *repeat*?
2. Что представляют из себя циклы *for, while* и *repeat*. Являются ли они функциями?
3. Какой оператор используется для прекращения цикла?
4. Какие управляющие конструкции используются для постановки условия выполнения цикла?
5. Какой цикл прерывается после условия прекращения?
6. Какой оператор используется для игнорирования операции с итератором?

Практические задания:

1. Создайте вектор от 1 до 15. Напишите цикл *while* таким образом, что бы он использовал созданный вектор в качестве итерируемого объекта. Условие прерывания цикла число 11.
2. Напишите цикл *for,* предназначенный для расчёта медианы с 1-го по 4-й столбца таблицы iris
3. Напишите цикл *for,* предназначенный для бутстрап расширения cars$speed с количеством повторов = 2000
4. Напишите цикл *for* для присвоения категории быков в зависимости от TPI быка по данным alta_202408. Категории: min-Q1 = K1, Q1-mean = K2, mean-Q3 = K3, Q3-max = K4.
5. Создайте вектор от 1 до 100. Напишите цикл *for,* который будет прибавлять 2 к каждому итератору. Цикл должен игнорировать каждый 10-й итератор.
6. Напишите цикл *for,* который будет переписывать данные о содержании
7. Создайте 2 вектора:

```
animals1 <- c("Птица", "Рыба", "Млекопитающее", "Рептилия")#класс
animals2 <- c("Орёл", "Щука", "Шимпанзе", "Черепаха")#животное
```

Напишите цикл *for* таким образом, что бы на выходе печаталось соответствие класс — животное

```
[1] "Птица - Орёл"
[1] "Рыба - Щука"
[1] "Млекопитающее - Шимпанзе"
[1] "Рептилия – Черепаха"
```

10.Функции семейства apply

Функции семейства *apply* или неявные циклы — конструкции языка программирования R, позволяющие многократно повторять одно и то же действие в зависимости от свойств объекта. Предоставляют мощный и эффективный способ обработки данных в различных структурах, таких как массивы, матрицы, списки и data.frame. Эти функции позволяют применять заданную функцию к элементам данных, избегая необходимости писать циклы, что значительно упрощает код и повышает его производительность. Вместо явного перебора элементов, *apply* обрабатывает данные по заданной размерности или структуре [Wickham, 2011].

10.1. apply

Неявный цикл, распространяющий действие некой функции на несколько строк/столбцов. Применяется в случаях, когда некое действие нужно применить не ко всей таблице, а только к ограниченному числу признаков. В общем виде функция apply(x, MARGIN, function)

```
apply(x, MARGIN, function)
```

где x- совокупность столбцов/строк являющихся объектом действия функции function, MARGIN — логический аргумент, принимающий значения 1 (функция работает по строкам) или 2 (функция работает по столбцам). При необходимости, дополнительные аргументы, а частности, na.rm = TRUE, записываются после функции. Данная особенность распространяется и на другие функции семейства *applly()*.

```
apply(x, MARGIN, function, na.rm = TRUE)
```

Очень частой задачей является вычисление средней арифметической для какого-либо продуктивного показателя. Для примера создадим объединённую таблицу (Bulls_db[38]) из каталогов поставщиков быков-производителей: AltaGenetics(n=314), ABS(n=257), semex(n=239), ST Genetics(n=319), WWS(n=861). Общее количество записей n= 1990. Содержание столбцов описано в Таблица 7

Таблица 7 Содержание и типы данных таблицы Bulls_db

Название	Тип данных	Описание
country	chr	Страна происхождения быка
farm	chr	Поставщик семени
number	chr	Международный номер быка
name	chr	Короткая кличка быка
TPI	num	Индекс TPI
milk	num	Прибавка к удою (в фунтах) относительно средней продуктивности коров в США
prot	num	Прибавка к содержанию белка в молоке (%) относительно средней продуктивности коров в США
fat	num	Прибавка жирномолочности (%) относительно средней продуктивности коров в США
k-casein	chr	Генотип по гену к-казеина
haplotype	chr	Генотип по моногенным заболеваниям
Date	POSIXct	Дата рождения быка
Y	chr	Год рождения быка

38 https://disk.yandex.ru/d/NLODMNsJnZGYLAD0%B5

По данным таблицы Bulls_db вычислим средние значения столбцов TPI, milk, prot, fat, используя функцию *apply()*. Функция *round()* была применена для устранения лишних десятичных знаков.

```
round
(apply(Bulls_db[c("TPI", "milk", "prot", "fat")], 2, mean), 2)
    TPI    milk    prot     fat 
2740.57 1105.07    0.04    0.12 
```

Результат функции, масштабированной при помощи функций семейства *apply*, можно сохранить в виде итоговой таблицы. Рассмотрим масштабирование функции opstat на примере таблицы iris.

```
out <- data.frame(apply(iris[1:4], 2, opstat))
> out
  Sepal.Length  Sepal.Width Petal.Length  Petal.Width
1 150.00000000 150.00000000  150.0000000 150.00000000
2   5.84333333   3.05733333    3.7580000   1.19933333
3   0.06783782   0.03570756    0.1446189   0.06244494
4   0.82806613   0.43586628    1.7652982   0.76223767
5   5.80000000   3.00000000    4.3500000   1.30000000
6   4.30000000   2.00000000    1.0000000   0.10000000
7   7.90000000   4.40000000    6.9000000   2.50000000
8   5.10000000   2.80000000    1.6000000   0.30000000
9   6.40000000   3.30000000    5.1000000   1.80000000
```

Получилась уже практически готовая к выгрузке таблица. Не хватает только подписать показатели описательной статистики. Для этих целей можно написать пользовательскую функцию, в тело которой входит неявный цикл apply. В результате, получается итоговая таблица для выгрузки.

```
opstat_df <- function(x){
  source("/home/oem/projectR/breeding/functions/opstat.R")
  out = data.frame(apply(x, 2, opstat))
   stat = matrix(data = c("n", "Mean", "SE", "SD", "Med", "Min", "Max", "Q1",
"Q3"), nrow = 9, ncol = 1)
  out = cbind(stat, out)
}
out <- opstat_df(iris[1:4])
out
```

10.2.tapply

Неявный цикл, распространяющий действие некой функции на анализируемые данные в зависимости от группирующего фактора. Имеет общий вид

```
tapply(x, INDEX, function)
```

где x — набор анализируемых данных, INDEX — группирующий фактор, function — функция. Для примера вычислим средний индекс TPI в зависимости от года рождения быка производителя по данным таблицы Bulls_db. При использовании неявных циклов удобно «упаковывать» результат вычисления в виде сводной таблицы.

```
df <- data.frame(tapply(Bulls_db$TPI, Bulls_db$Y, mean))
df$Year <- row.names(df)
colnames(df) <- c("TPI", "Year")
          TPI Year
2008 2088.000 2008
2009 1821.000 2009
2010 2048.333 2010
2011 2142.364 2011
2012 2240.357 2012
2013 2243.966 2013
2014 2415.759 2014
2015 2477.930 2015
2016 2524.539 2016
2017 2615.893 2017
2018 2715.939 2018
2019 2798.006 2019
2020 2893.071 2020
2021 2962.287 2021
2022 3059.000 2022
```

Функции *tapply()* можно передать максимум два аргумента в качестве группирующего фактора. Для этого аргумент INDEX заключают внутрь функции *list()*. Результат отображается в консоли в виде матрицы. При попытки использовать три и более группирующих фактора, функция *tapply()* отработает, но результат будет рассчитан только для первых двух факторов.

```
tapply(Bulls_db$TPI,  list(Bulls_db$Y, Bulls_db$farm), mean)
          ABS     Alta    Semex       ST
2008       NA       NA 2088.000       NA
2009       NA       NA 1821.000       NA
2010       NA       NA 2048.333       NA
2011       NA       NA 2142.364       NA
2012       NA       NA 2206.769 2677.000
2013 2449.000 2529.000 2188.048 2240.333
2014 2494.857 2260.500 2403.600       NA
2015 2500.286 2508.000 2458.032 2498.500
2016 2488.136 2628.250 2514.915 2804.000
2017 2623.689 2649.069 2603.110 2595.538
2018 2704.034 2681.514 2724.444 2783.000
2019 2781.971 2772.952       NA 2840.647
2020 2872.271 2855.753       NA 2926.781
2021 2949.105 2938.415       NA 2992.707
2022       NA 3059.000       NA       NA
```

10.3. aggregate

Недостатком функции *tapply()* является возможность использования только одного столбца с анализируемыми даными и максимум двух группирующих факторов. При необходимости использования какой-либо функции на одновременно на несколько наборов данных в зависимости от градаций группирующего фактора существует неявный цикл *aggregate(x, list(by), function)*. Особенностью неявного цикла *aggregate()* является передача группирующего фактора внутри функции *list()*. Для примера вычислим среднюю арифметическую по индексу TPI, а так же прибавке по удою, жиру и белку по данным таблицы Bulls_db В качестве группирующего фактора используем год рождения животного. Результат функции *aggregate()* удобно трансформировать в сводную таблицу для выгрузки и последующей вставки в отчёт, статью и т/д.

```
df   <-   data.frame(aggregate(Bulls_db[c("TPI",   "milk",   "prot",   "fat")],
list(Bulls_db$Y), mean))
df[c("TPI", "milk")] <- round(df[c("TPI", "milk")])
df[c("prot", "fat")] <- round(df[c("prot", "fat")], 2)
df
 Group.1  TPI milk  prot   fat
1     2008 2088  321 -0.04 -0.04
2     2009 1821 -246  0.03  0.02
3     2010 2048  553 -0.03 -0.03
4     2011 2142  819 -0.01 -0.06
5     2012 2240  523  0.00  0.02
6     2013 2244  498  0.01  0.03
7     2014 2416  575  0.03  0.06
8     2015 2478  889  0.01  0.03
9     2016 2525  919  0.03  0.04
10    2017 2616  992  0.03  0.08
11    2018 2716 1081  0.04  0.10
12    2019 2798 1085  0.06  0.15
13    2020 2893 1286  0.05  0.16
14    2021 2962 1351  0.06  0.18
15    2022 3059 1342  0.11  0.2
```

10.4. sapply и lapply.

Оба неявных цикла предназначены для применения функции к каждому элементу какого либо объекта. Как правило используются совместно с пользовательскими функциями, включающими в себя конструкцию *if(){}else{}*. Отличие между функциями заключается в том, что отклик *lapply()* всегда

представлен в виде списка. *Sapply()* имеет дополнительный аргумент *simplify* позволяющий выбрать класс объекта для вывода данных. Для примера создадим числовой вектор от 1 до 10 и функцию, прибавляющую к каждому значению вектора 1. Функцию применим через неявный цикл *sapply()* таким образом, что бы результат можно было преобразовать в таблицу. Для этого укажем соответствующий класс объекта при помощи аргумента *simplify.*

```
x <- 1:10
my_fun <- function(x){x+1}
df <- data.frame(
  x = x,
  y = sapply(x, my_fun, simplify = "data.frame")
)
df
    x  y
1   1  2
2   2  3
3   3  4
4   4  5
5   5  6
6   6  7
7   7  8
8   8  9
9   9 10
10 10 11
```

Функцию *sapply()* можно использовать для извлечения из таблиц столбцов с конкретным типом данных. Например, из таблицы Bulls нужно извлечь только столбцы с типом данных character.

```
Bulls[names(Bulls)[sapply(Bulls, is.character)]]
                  FullName      Country
1  PEAK CH ALTAINSPIRE-ET     USA
2 PEAK ALTAZEROGRAVITY-ET     USA
3 PEAK ALTAVORTEXVAULT-ET     USA
```

С помощью *sapply* можно использовать управляющую конструкцию *if(){} else{}* . Для этого создаётся пользовательская функция. Сама конструкция пишется в теле функции. В круглых скобках пишется условие выполнения действия, в фигурных скобках до оператора *else* пишется само действие, после – альтернативное действие. Например, необходимо присвоить племенную категорию коров в зависимости от удоя. Создадим таблицу animals_production, содержащую данные об удое за 305 дней.

```
animals_production <- data.frame(
```

```
  ID = runif(min = 100, max =  9999, n = 20),
  Milk = runif(min = 5000, max = 12000, n = 20)
)


animals_production <- round(animals_production)
head(animals_production)
     ID  Milk
1  3762 11869
2  1398 10609
3   230  7790
4  3743  9002
5  5730  8789
6  5963  7904
```

Условно, особи с удоем 9 000 и более считаются племенными, ниже — определяются в брак. Для присвоения категории необходимо написать пользовательскую функцию с использованием конструкции *if(){}else{}*

```
my_fun <- function(x){
  if(x < 9000){
    print("Брак")
  }else{
  print("Племенное_использование")
    }
}
```

В качестве примера создадим столбец Cat, содержащий 2 градации «Племенное_использование» и «Брак».

```
animals_production$Cat  <-  sapply(animals_production$Milk,  my_fun,  simplify  =
data.frame)
animals_production
     ID  Milk                      Cat
1  3762 11869 Племенное_использование
2  1398 10609 Племенное_использование
3   230  7790                   Брак
4  3743  9002 Племенное_использование
5  5730  8789                   Брак
6  5963  7904                   Брак
7  4956  8580                   Брак
8  8105 10019 Племенное_использование
9   235  7386                   Брак
10 7961 10183 Племенное_использование
11 2495 10168 Племенное_использование
12 7294  7803                   Брак
13 4901  6927                   Брак
14 5002  8165                   Брак
15  412  9169 Племенное_использование
16  481  6955                   Брак
17 1490  9652 Племенное_использование
18 6445  5708                   Брак
19 9527  6026                   Брак
20 6401  7823                   Брак
```

Преимуществом такого подхода является то, что в отличии от цикла *for* созданный столбец с категориями будет той же размерности, что и столбец с исходными данными.

10.5. mapply

Функция *mapply* в R является многомерным аналогом *sapply* и применяет указанную функцию к элементам нескольких векторов/списков параллельно. В общем виде функция записывается как:

```
mapply(FUN, x, y)
```

где :

1.FUN — применяемая функция.

2. x, y — векторы/списки, передаваемые как аргументы в FUN (должны быть одинаковой длины или подчиняться правилу переписывания).

3.MoreArgs — список дополнительных аргументов для FUN, не изменяемых в итерациях.

4.SIMPLIFY — упрощать результат до вектора/матрицы (TRUE по умолчанию). При SIMPLIFY = FALSE результат выдаётся в виде списка.

5.USE.NAMES — сохранять имена первого аргумента (TRUE по умолчанию).

То есть mapply обеспечивает взаимодействие элементов используемых наборов данных по принципу первый с первым, второй со вторым и так далее. Рассмотрим на конкретном примере

```
> a <- c(1, 2, 3, 1, 2, 3)
> b <- c(1, 3, 3, 2, 2, 1)
> mapply(sum, a, b)
[1] 2 5 6 3 4 4
```

На первый взгляд *mapply* может показаться не очень полезной так как в большинстве случаев код вполне может обойтись без него. Однако эта функция позволяет более гибко использовать код с использованием управляющих

конструкций. Например, требуется получить сумму элементов двух векторов, при условии, что они равны.

```
result <- mapply(function(x, y){
if(x == y){print(x+y)}else{print("-")}}, a, b)
result
[1] "2" "-" "6" "-" "4" "-"
```

В животноводстве функция *mapply* может быть использована для применения разных весовых коэффициентов удоя коров при составлении общего селекционного индекса для разных пород. Например, у голштинской породы весовой коэффициент удоя = 0.4, айширской — 0.5, симментальской — 0.6. Рассмотрим на конкретном примере.

```
milk <- c(9800, 5300, 8000, 9543, 10342)
breed <- c("HO", "SE", "AI", "HO", "HO")
mapply(function(x, y){
  if(x == "HO"){
    print(y*0.4)
  }else if(x == "SE"){
      print(y*0.6)
    }else{print(y*0.6)}
    }, breed, milk)
```

Важные нюансы:

- Если длины векторов различаются, используется правило переписывания (recycling), но выводится предупреждение.
- Если первый аргумент имеет имена, они сохраняются в результате (если USE.NAMES = TRUE).
- По умолчанию mapply пытается преобразовать вывод в вектор/матрицу (как *sapply*). Для сохранения списка используйте SIMPLIFY = FALSE.

Практические задания:

1) По данным таблицы food_kaggle рассчитайте среднее содержание протеина (Protein) используя лишь строки, где содержаться данные о продуктах из говядины, свинины и курицы (Category == "BEEF", Category == "CHICKEN", Category == "PORK").

2) По данным каталога быков AltaGenetics (alta_202408.xlsx) рассчитайте средние значения прибавки к содержанию протеина в молоке дочерей (Prot%) в зависимости от генотипа по гену каппа-казеина (Kappa-Casein).
3) По данным каталога быков голштинской породы от компании ST Genenic за декабрь 2022 года (ST.scv[39]) вычислите средние значения прибавки к продуктивности дочере по удою (PTA Milk), жирномолочности (% Fat), содержанию белка в молоке (% Pro), а так же комплексному индексу племенной ценности TPI при помощи функции *apply()*.
4) В таблице cat_traits создайте столбец, содержащий записи о том, является ли живая масса животного больше, либо меньше среднего значения
5) По данным каталога быков AltaGenetics (alta_202408.xlsx) создайте таблицу. Добавьте столбец, содержащий записи TRUE/FALSE. Запись должна быть присвоена в том случае, если генотип животного по гену бета-казеина (Beta-Casein) — A2A2, каппа-казеина (Kappa-Casein) — AA.

11. Работа с пропущенными значениями

В реальных данных очень часто встречаются строки с пропущенными значениями и тому есть самые разные причины. Наиболее простая из них – измерения не были выполнены по тем или иным причинам [Salgado и др., 2016]. Например, из-за беспокойного поведения некоторых животных не имелось возможности объективно осуществить их оценку экстерьера или же произвести отбор проб биоматериала для последующей генетической паспортизации. Или во время оценки нужные животные были переведены в другое отделение или в летние лагеря. Другая причина появления пропущенных записей – получение сомнительных результатов в ходе лабораторных исследований. Принято считать, что жирномолочность коров ниже 3.5 и выше 6.0 % является следствием не физиологических особенностей животного, а нарушением

39 https://disk.yandex.ru/client/disk/Data, исходный файл скачан в свободном доступе с сайта компании https://www.stgen.com/sire-directory/dairy-proof.aspx?proof=usa&code=holstein&language=english&title=dairy-cattle

технологии отбора проб или проведения лабораторных исследований. Доказано, что сомнительные результаты экспериментов либо наблюдений могут повлиять на точность статистической обработки данных, что может являться причиной неправильно принятых гипотез [Wiernik, Dahlke, 2020]. Поэтому при обработки данных сомнительные результаты принято исключать из анализа [Neves, Amaral, 2020]. Одним из способов исключения является замена заведомо сомнительных результатов записями NA (от англ. Not avaliable). В качестве примера рассмотрим синтетическую таблицу *df*.

```
    Milk  Fat
1   7496 4.67
2   9510 5.03
3   8415 4.40
4   7753 6.70
5   7582 5.76
6  10177 6.96
7  11960 6.51
8   7389 5.48
9  10955 3.17
10 11038 3.00
11 10497 4.01
12 11766 4.62
13 10630 6.95
14  8702 5.82
15  7781 4.77
```

В таблице df пробы № 9 и 10 имеют жирномолочность ниже предельно допустимого минимума (lower bound), содержание молочного жира в пробах № 4, 6, 7 и 13 превышают предельно допустимый максимум (upper bound). Заменим эти значения на NA.

```
df$Fat[c(4, 6, 7, 9, 10, 13)] <- NA
df
    Milk  Fat
1   7496 4.67
2   9510 5.03
3   8415 4.40
4   7753   NA
5   7582 5.76
6  10177   NA
7  11960   NA
8   7389 5.48
9  10955   NA
10 11038   NA
11 10497 4.01
12 11766 4.62
13 10630   NA
14  8702 5.82
15  7781 4.77
```

Одной из особенностей работы с пропущенными данными является то, что подавляющее большинство статистических функций R по умолчанию выдают ошибку при наличии хотя бы одного NA в массиве данных. Во избежании этого в ряде функций вычисляющих какие-либо статистические показатели предусмотрен аргумент na.rm = TRUE.

```
mean(df$Fat)
[1] NA
mean(df$Fat, na.rm = TRUE)
[1] 4.951111
```

В контексте использования функций семейства *apply* аргумент *na.rm* пишется после использования основной функции. Для примера рассчитаем стандартное отклонение роста супергероев у мужчин и женщин по данным датасета в открытом доступе heroes information[40], предварительно удалив записи о росте = -99 см.

```
tapply(df$Height, df$Gender, sd, na.rm = TRUE)
       -   Female     Male 
23.83715 23.71440 68.83127
```

Другим способом «избежания» пропущенных записей является удаление всех строк, содержащих NA. Для этой задачи в R предусмотрена функция *na.omit()*. Для примера рассмотрим импровизированную таблицу *df*, имитирующую данные о живой массе (LW) лабораторных мини-свиней разной масти (color). За основу брались реальные данные о массе свиноматок мини-свиней ИЦиГ СО РАН [Никитин и др., 2024; Nikitin и др., 2018].

```
    LW   Color
1   NA   Black
2   55 Spotted
3   65   White
4   59    <NA>
5   70   White
6   58   White
7   63    <NA>
8   64   Black
9   69 Spotted
10  61   Black
11  54    <NA>
12  NA Spotted
```

40 https://github.com/pbiecek/TechnikiWizualizacjiDanych2018/blob/master/Projekt1/WachulecPysiakMakowski/dataset/heroes_information.csv

Очистим таблицу от строк с пропущенными записями. На выходе, таблица df должна лишиться строк № 1, 4, 7, 11 и 12.

```
> na.omit(df)
   LW   Color
2  55 Spotted
3  65   White
5  70   White
6  58   White
8  64   Black
9  69 Spotted
10 61   Black
```

Если нужно заменить любое значение на NA или NA на любое значение можно воспользоваться базовым циклом *for.* Например, все отрицательные числа нам нужно заменить на NA

```
a <- c(1, -5, 2, 5, 0, -3, 1, -4, 0)
for(i in a){
  if(i < 0){print(NA)}else{print(i)}
}
```

В базовом функционале R имеется функция *is.na()*, результатом выполнения которой является TRUE (ячейка содержит NA) или FALSE (заполненная ячейка). Использование результата можно использовать для подсчёта количества пропущенных/заполненных ячеек Таблица 8, так и для создания ключевого столбца.

Для подсчёта количества пропущенных/ заполненных записей в R существуют как базовые элементы кода, так и специальная библиотека *naniar.* Данная библиотека содержит в себе разнообразные функции для операций с пропущенными записями [Tierney, 2024]. Примеры приведены в Таблица 8

Интересной функцией пакета *naniar* является *mcar_test()*, выполняющая тест Литтла. Его суть заключается в оценке случайности наличия пропущенных значений в таблице. Нулевая гипотеза заключается в том, что наличие пропущенных значений полностью случайно, а сам критерий представляет собой значение хи-квадрат [Little, 1988].

Таблица 8 Инструменты подсчёта пропущенных/заполненных записей

Базовые функции	Функции библиотеки naniar	Описание
sum(is.na(x))	n_miss()	Подсчёт количества пропущенных значений
sum(!is.na(x))	n_complete()	Подсчёт количества заполненных значений
sum(is.na(df$Fat))/ length(df$Fat)*100	pct_miss()	Подсчёт доли пропущенных значений (%)
sum(is.na(df$Fat))/ length(df$Fat)	prop_miss()	Подсчёт доли пропущенных значений
sum(!is.na(df$Fat))/ length(df$Fat)	prop_complete()	Подсчёт доли заполненных значений
sum(!is.na(df$Fat))/ length(df$Fat)*100	pct_complete()	Подсчёт доли заполненных значений (%)
	add_prop_miss(x)	Создаёт дополнительный столбец в data.frame, содержащий долю NA в каждой строке

Контрольные вопросы к разделу

1. Какие причины могут привести к пропущенным строкам таблицы с исходными данными?
2. Что следует делать с сомнительными результатами наблюдений или экспериментов?
3. Какая специальная библиотека существует для обработки пропущенных данных?
4. Что может повлечь за собой включение сомнительных данных в обработку данных?

Практические задания:

1. По данным ВНИИПлем, находящимся в открытом доступе[41], была сформирована таблица ВНИИ_ПЛЕМ_Дюрок_хряки. Загрузите эту таблицу в среду R.

41 База данных национального генофонда свиней за 2022 год. Министерство сельского хозяйства Российской Федерации. Департамент животноводства и племенного дела. ФГБНУ Всероссийский научно-исследовательский институт племенного дела. [Электронный ресурс]. URL:https://vniiplem.ru/l/gisc/bd_ng/db-ng-svin/

2. По данным этой таблицы оцените количество строк с пропущенными данными о толщине шпика используя базовые функции R.

3. По данным ВНИИ_ПЛЕМ_Дюрок_хряки оцените долю строк с пропущенными кличками хряков используя базовые функции R.

4. Оцените количество заполненных ячеек в столбце *category* таблицы *storms* из дополнительной библиотеки *dplyr* с использованием возможностей библиотеки *nanyar*

5. По данным таблицы cat_traits[42] *оцените долю пропущенных записей в столбце FoodBrand* с использованием возможностей библиотеки *nanyar*

6. В таблице *heroes_information.xlsx*[43] замените все значения weight = -99 на NA. Рассчитайте средний вес супергероев.

8. В таблице «Поросята задание»[44] уберите все строки с пропущенными записями

12. Работа со строковыми значениями

В животноводстве огромную роль играет ведение зоотехнического учёта. Практика показывает, что кличка одного и того же животного может быть записана в нескольких вариантах. Одной из распространённых форм ошибок записей племенного учёта является написание клички одного и того же животного строчными и заглавными буквами. Эти и подобные задачи в языке R решаются при помощи операций со строковыми значениями. Работа со строками в R происходит с использованием языка регулярных выражений *regex* интегрированного в среду R. Подробному описанию работы с регулярными выражениями посвящена самостоятельная монография [Филд, 2008]. В работоспособном коде R язык регулярных выражений нельзя вставлять в

42 Emİrhan Bulut, Global Street Cats Statistics for AI Research, 2024. URL: https://www.kaggle.com/datasets/emirhanai/global-street-cats-statistics-for-ai-research?resource=download

43 https://github.com/pbiecek/TechnikiWizualizacjiDanych2018/blob/master/Projekt1/WachulecPysiakMakowski/dataset/heroes_information.csv

44 Данные получены в ходе зоотехнического учёта https://disk.yandex.ru/d/kAA883JxY55RTQ

произвольном порядке. Для интеграцию регулярных выражений в код используются специальные функции. В базовой конфигурации R примерами функций для работы с регулярными выражениями служат *grep("pattern", x)* и *grepl("pattern", x)*. В обеих функциях аргумент *"pattern"* передаёт функции фрагмент текста, который необходимо отыскать, x — исходный массив данных. Результат функций несколько различается: *grep()* выдаёт номера записей вектора или столбца data.frame, где содержится искомый фрагмент. Откликом *grepl()* являются логические выражения, соответствующие каждой записи: TRUE — искомый фрагмент обнаружен, FALSE — не обнаружен. В данном примере функции *grep()* и *grepl()* продемонстрировали порядковые номера записей о свиноматках, чьи клички содержат фрагмент «Гир». Такой подход можно использовать для поиска родственных семейств, имеющих схожие клички.

```
Sow_names <- c("Гирлянда_2166", "Гирлянда_4218", "ЕЖЕВИКА_5260",
"Ежевика_1854", "Гиря_2020", "Лапочка_1056", "Лапуля", "290", "5212")
> grep("Гир", Sow_names)
[1] 1 2 5
grepl("Гир", Sow_names)
[1]  TRUE  TRUE FALSE FALSE  TRUE FALSE FALSE FALSE FALSE
```

По умолчанию, отклик функций *grep()* и *grepl()* содержит адреса всех записей, где встречается искомое выражение. Например, если в качестве аргумента *"pattern"* подставить «21», откликом функции будут номера всех кличек, содержащих в себе искомый фрагмент.

```
grep("21", Sow_names)
[1] 1 2 9
```

Для поиска записей с нужным ключом, находящемся в строго определённом месте следует пользоваться приёмами языка регулярных выражений. Наиболее часто используемы сгруппированы в Таблица 9.

Таблица 9 Примеры использования регулярных выражений

Регулярное выражение	Описание	Пример кода в R
^i	поиск по выражению, начинающегося с i-го символа	`grep("^52", Sow_names)` `[1] 9`
i$	поиск по выражению, заканчивающегося на о-й символ	`grep("6$", Sow_names)` `[1] 1 6`
[i]	Поиск записей с любым из символов указанного фрагмента	`grep("[Гир]", Sow_names)` `[1] 1 2 3 4 5`
^[^ i]	Поиск записей, которые начинаются с любого другого фрагмента	`grep("^[^Гир]", Sow_names)`
i{j}	Поиск i-го фрагмента, повторяющегося j-е количество раз	`grep("6{2}", Sow_names)` `[1] 1`
[A-Z] [А-Я]	Поиск записей, содержащих любую заглавную букву.	`grep("[А-Я]", Sow_names)` `[1] 1 2 3 4 5 6 7`
[a-z] [а-я]	Поиск записей, содержащих любую строчную букву.	`rep("[а-я]", Sow_names)` `[1] 1 2 4 5 6 7`
[0-9]	Поиск записей, содержащих любую цифру.	`grep("[0-9]", Sow_names)` `[1] 1 2 3 4 5 6 8 9`
[A-Z0-9]	Поиск записей, содержащих любую заглавную букву и(или) любую цифру.	`grep("[А-Я0-9]", Sow_names)` `[1] 1 2 3 4 5 6 7 8 9`

При поиске кличек следует учитывать регистр. Для игнорирования регистра функции *grep()* и *grepl()* снабжены аргументом ignore.case = TRUE.

```
grep("ЕЖЕВИКА", Sow_names)
grep("ЕЖЕВИКА", Sow_names, ignore.case = TRUE)
```

Для получения в результате самих записей, а не их индексов, при использовании функции g*rep()* задействуют аргумент value = TRUE.

```
grep("Ежевика", Sow_names, value = TRUE)
[1] "Ежевика_5260" "Ежевика_1854"
```

Важно помнить, что при взаимодействии *grep()* с массивом данных, содержащих как заглавные так и строчные буквы в пределах искомого фрагмента лучше писать первую букву заглавной, а остальные -строчной, иначе ожидаемого отклика не получится.

Очень часто приходится сталкиваться с ситуациями по замене одной записи на другую. Для решения подобных задач базовый функционал языка про-

граммирования R укомплектован функциями *sub("x", "y", data)* и *gsub("x", "y", data)*, где *"x"* -исходная запись (что меняем), *"y"* - конечная запись(чем заменяем), *data* – массив данных. Их отличие в том, что *sub()* заменяет только первое вхождение шаблона, *gsub()* — все вхождения. Например установлено, что свиноматки с кличками «Ежевика» на самом деле относятся к семейству Красавицы. Замена кличек с «Ежевика» на «Красавица» будет выглядеть следующим образом:

```
gsub("ЕЖЕВИКА", "Красавица", Sow_names, ignore.case = TRUE)
[1]  "Гирлянда_2166"     "Гирлянда_4218"     "Красавица_5260"   "Красавица_1854"
"Гиря_2020" "Лапочка_1056" "Лапуля" "290"          "5212
```

Иногда в электронных записях зоотехнического учёта появляются лишние символы, не относящиеся к буквенно-циферному ряду. Для очистки записей от нежелательных спецсимволов

```
Sow_names1    <-    c("Гирлянда_2166",    "Гир///лянда_4218",    "ЕЖ#ЕВИКА_5260",
"Еже&вика_1854", "Гир.я_2020", "Лапочка_1056",  "Лапуля", "290", "5212")
gsub("[^А-Яа-я0-9]", "", Sow_names1)
[1] "Гирлянда_2166" "Гирлянда_4218" "ЕЖЕВИКА_5260"  "Ежевика_1854"  "Гиря_2020"
"Лапочка_1056"  "Лапуля"        "290"           "5212"
```

Для разделения строки на две и более существует функция *strsplit(x," ")*. В кавычках указывается разделитель.

```
x <- "apple;orange;banana"
> strsplit(x, ";")
[[1]]
[1] "apple"  "orange" "banana"
```

Аналогично, объединение строк осуществляется функцией *paste(x, y, sep = " ")*, где *x* и *y* — объединяемые строки. Аргумент *sep = " "* используется для обозначения разделителя.

```
a <- "Буян"
> b <- "1964"
> paste(a, b, sep = ".")
[1] "Буян.1964"
```

Для расширенного функционала работы со строковыми выражениями в R используется специальная библиотека — *stringr* [Wickham, Grolemund, 2017].

Разберём работу основных функций библиотеки *stringr* на примере вектора x, содержащего записи о номере и кличке быков-производителей. Данную библиотеку удобно использовать при работе с записями о кличках, гаплотипах, моногенных заболеваниях и микросателлитном профиле. Описание основных функций представлено в Таблица 9

Таблица 9 Основные функции библиотеки *stringr*

Название	Описание	Пример
str_length(x)	Выводит информацию о количестве символов в каждой записи	`str_length(x)` `[1] 29 29 25 27 32 28 27`
str_c(x, collapse = ", ")	Объединяет все записи в одну с использованием указанного разделителя	`str_c(x, collapse = ", ")` `[1] "SWE000000000045182_VH HADDOCK, …… USA000003215425396_Fugleman"`
str_sub(x, i, j)	Ограничивает записи с i-го по j-й символ	`str_sub(x, 1, 4)` `[1] "SWE0" "DNK0" "CAN0" "DEU0" "USA0" "HOUS" "USA0"`
str_subset(x, "pattern")	Извлекает записи содержащие фрагмент "pattern" из массива данных x	`str_subset(x, "USA")` `[1] "USA000003215425521_ALTAALANZO-ET" "HOUSAM003010354356_AltaDISCO" "USA000003215425396_Fugleman"`
str_count(x, "pattern")	Выводит информацию о количестве повторений фрагмента "pattern" в каждой записи набора данных x	`str_count(x, "0")` `[1] 10 9 8 6 5 4 5`
str_detect(x, "pattern")	Сообщает о наличии/отсутствии фрагмента "pattern" в записях x	`str_detect(x, "USA")` `[1] FALSE FALSE FALSE FALSE TRUE TRUE TRUE`
str_extract(x, "pattern")	Извлекает фрагмент "pattern" из записей объекта x	`str_extract(x, "USA")` `[1] NA NA NA NA "USA" "USA" "USA"`
str_replace(x, "pattern", "replacement")	заменяет совпадения "pattern" новым текстом "replacement":	`str_replace(x, "USA", "840")` `"HO840M003010354356_AltaDISCO" "840000003215425396_Fugleman"`
str_split(x, "sep")	разбивает строку на несколько частей	`str_split(x, "_")` `[1] "SWE000000000045182" "VH HADDOCK"`

Контрольные вопросы

1. Какой функцией в R проверить наличие шаблона в строке и получить логический вектор?
2. Чем отличаются *sub()* и *gsub()*?
3. Какой шаблон соответствует строкам, начинающимся с гласной буквы?
4. Что вернёт *grep("a.b", c("acb", "axb", "a b"))*?
5. Как заменить все последовательности цифр в строке на "[число]"
6. Как разбить строку "apple;orange;banana" используя ";" в качестве разделителя

Практические задания:

1. Выберите все слова, начинающиеся с заглавной буквы

```
c("Собака", "осёл", "КОТ", "пЕТУХ")
```

2. Замените "кг" на "килограмм" только когда это отдельное слово: `c("5 кг яблок", "перекг 2024", "вес 0.5кг")`
3. Удалите лишние символы в строке: `"Дан@ные с п^ро;;;бел!ами"`
4. Разделите строку "яблоки, груши; апельсины/бананы" на отдельные слова по любым не-буквенным символам
5. По данным каталога быков компании AltaGenetics за август 2024 года (alta_202408.xlsx) сформируйте столбец, содержащий информацию о стране происхождения быка. В качестве исходных данных используйте столбец InterRegNumber. Информация о стране рождения быка зашифрована в 3-5 символах.
6. По данным таблицы food_kaggle найдите все категории продуктов, в нанзвании которых присутствует слово "MILK".
7. Объедините имена и фамилии сотрудников используя пробел в качестве разделителя

```
c("Иван", "Ольга", "Михаил")
c("Максимов", "Кузнецова", "Потапов")
```

13. Визуализация данных при помощи базовых возможностей языка R

Выбор метода статистического анализа очень важен для получения объективного результата. Для выбора метода необходимо иметь представление об объекте исследования. К наиболее общим выборочным характеристикам можно отнести такие показатели как как средняя арифметическая, показатели изменчивости (стандартное отклонение, коэффициент вариации, дисперсия) лимиты, характер распределения. Наиболее быстрым и простым способом получения информации о наборе данных является её графическое представление. Джон Тьюки считал [Тьюки, 1981], что хороший график подчас позволяет лучше понять закономерность чем сложные статистические вычисления. Работе с графиками полностью или частично посвящен ряд книг [Брюс П., Брюс Э., 2018; Самсонов, 2017; Chang, 2013]. Язык программирования R представляет широкий выбор базовых инструментов визуализации данных. В этом разделе будут рассмотрены общие принципы построения диаграмм при помощи R.

Основными функциями визуализации данных являются: *hist()* — строит гистограммы,

barplot() – строит столбчатые диаграммы,

plot() – строит диаграммы рассеяния,

boxplot() – «ящик с усами»,

qqnorm() – график «квантиль-квантиль»

pie() - круговая диаграмма

lines() наносит линии дополнительно к основной диаграмме. Соединяет точки на диаграмме рассеяния, преобразуя её в график. Добавляет линию распределения на гистограмму.

У базовых графических функций имеется ряд общих аргументов –

col – позволяет выбрать цвет графика. В кавычках указывается название цвета по английски или его код в классификаторе[45].

45 http://www.sthda.com/english/wiki/colors-in-r

xlim – позволяет ограничить набор данных по оси абсцисс

ylim – позволяет ограничить набор данных по оси ординат

xlab – позволяет переименовать ось абсцисс

ylab – позволяет переименовать ось ординат

main – название графика

legend.text – адресует на набор данных значения которых преобразуются в текст легенды, по умолчанию legend.text = NULL.

cex.axis – регулирует размер шрифта разметки осей

lwd – регулирует толщину линии

lty – регулирует тип линии

cex.main – регулирует размер шрифта подписи рисунка

cex.lab – регулирует размер шрифта подписи осей

Если вам необходимо полностью избавиться от подписей оси или всего графика, то в качестве аргумента *xlab, ylab* и *main* передаём *NULL*

```
hist(ChickWeight$weight, main = NULL)
```

Для экспорта изображений используются функции *bmp(), jpeg(), png()* и им подобные. В качестве основного аргумента путь к папке импорта с указанием имени файла. Содержимое рисунка создаётся при помощи функций *barplot(), plot(), hist()* и т.д. Важными аргументами перечисленных функций являются *width* (ширина изображения) и *height* (высота изображения), выражаемые в количестве пикселей. После завершения создания рисунка в коде прописывается функция *dev.off()*, внутри скобок не пишем ничего. Только тогда среда R завершает экспорт рисунка. Для примера создадим диаграмму рассеяния, показывающую зависимость длины листа от ширины листа у ирисов вида *setosa*.

```
 jpeg("Путь к папке/Fig2.jpeg", width = 750, height = 750)
plot(setosa[,1], setosa[,2], xlab = "Длина листа", ylab = "Ширина листа")
dev.off()
```

13.1. Столбчатые диаграммы

Столбчатая диаграмма является одним из самых простых способов визуализаци средних и частотных величин. В базовом функционале R создание столбчатой диаграммы возможно при помощи функции *barplot()*. Особенность использования данной функции состоит в том, что она практически не используется на массиве сырых данных. Суть функции *barplot()* состоит в наглядном сравнении двух или более средних величин. Поэтому перед непосредственным использованием *barplot()* рекомендуется создать сводную таблицу, содержащую средние величины. Для примера загрузим каталог быков AltaGenetics. Далее создадим столбец (Country) с аббревиатурой страны происхождения быка. Используя столбец Country в качестве группирующего фактора, создадим таблицу с округлёнными средними значениями TPI для каждой страны. Визуализируем значения TPI на столбчатой диаграмме. Для увеличения свободного места нам графике, воспользуемся аргументом ylim. Для окрашивания столбцов в отдельный цвет используем аргумент col. Получившийся график Рисунок 8 позволяет наглядно сравнить средние значения TPI быков из разных стран. Значения средних величин можно передать через легенду, которая задаётся при помощи аргумента *legend.text*.

```
alta <- readxl::read_excel("alta_2022408.xlsx")
alta$country <- str_extract(alta$InterRegNumber, "^[A-Z0-9]{3,5}")
df <- data.frame(tapply(alta$TPI, alta$country, mean))
colnames(df) <- "TPI"
df$Country <- row.names(df)
df$TPI <- round(df$TPI)
barplot(df$TPI, legend.text = df$TPI, names.arg = df$Country, col =
c("red", "green", "blue"), ylim = c(0, 4000))
```

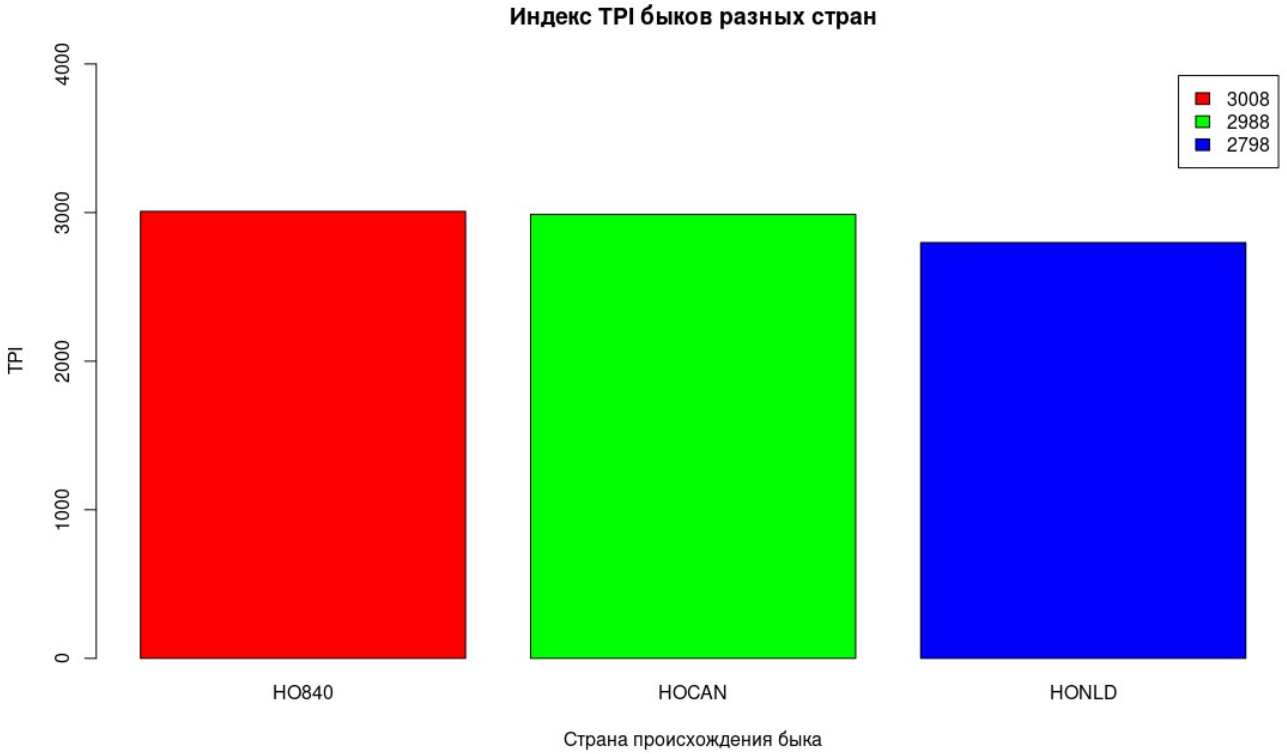


Рисунок 8 Индекс TPI быков разных стран из каталога AltaGenetics за август 2024, визуализированный на столбчатой диаграмме при помощи базовых графических возможностей.

Для визуализации зависимости переменной от двух факторов в качестве аргумента *x* присваивают матрицу, где в качестве строк используется 1-й фактор, в качестве столбцов второго. В качестве примера разберём визуализацию динамики индекса племенной ценности быков TPI по данным таблицы Bulls_db. Для начала соберём дополнительную таблицу, содержащую средние значения TPI в зависимости от года рождения быка и поставщика семени. Удаляем ненужные столбцы, преобразуем таблицу в матрицу. Оставляем один столбец пустым, что бы на рисунке осталось место для легенды.

```
df  <-  data.frame(tapply(Bulls_db$TPI,  list(Bulls_db$farm,  Bulls_db$Y),
mean, na.rm = TRUE))
colnames(df) <- 2008:2022
df[c("2008", "2009", "2010", "2011", "2012", "2019", "2020", "2021")] <-
NULL
df$`2022` <- 0
df <- as.matrix(df)
```

Затем из дополнительной таблицы строим столбчатую диаграмму, названия поставщиков семени выносим в легенду Рисунок 9. Важным является аргумент beside, который по умолчанию равен FALSE. При beside = FALSE происходит «наслоение» категорий друг на друга.

```
barplot(df, legend.text = row.names(df), beside = TRUE,
        col = c("red", "green", "blue", "darkorange3", "aquamarine"),ylim =
c(0, 3000))
```

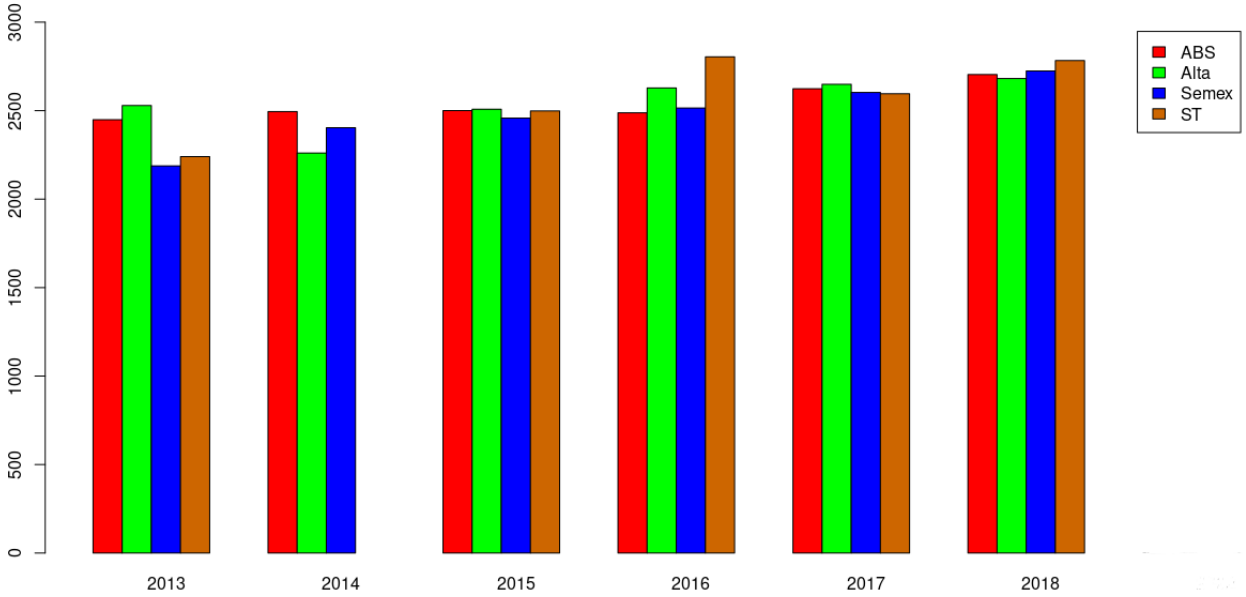


Рисунок 9 Динамика TPI от разных поставщиков семени

13.2. Круговые диаграммы

Признаки с альтернативной изменчивостью принято иллюстрировать с помощью круговых диаграмм. К таким признакам относятся частоты аллей и генотипов, встречаемость фенотипов и т.д. Подобные признаки обычно выражаются в долях от 0 до 1 или в процентах [Лакин, 1990; Меркурьева, 1970]. В R круговые диаграммы строятся при помощи функции *pie()*. В качестве примера возьмём частоты аллелей групп крови системы *e* у мини-свиней ИЦиГ СО РАН [Шатохин и др., 2014]. При помощи аргумента *radius* регулируется размер круга, *labels* контролирует подписи.

```
pie(x    =    E_sistem$Freq,    col    =    c("deeppink4","darkslategray1",
"darkorange3", "cyan4"),
   radius = 1.0,
   labels = E_sistem$Name)
```

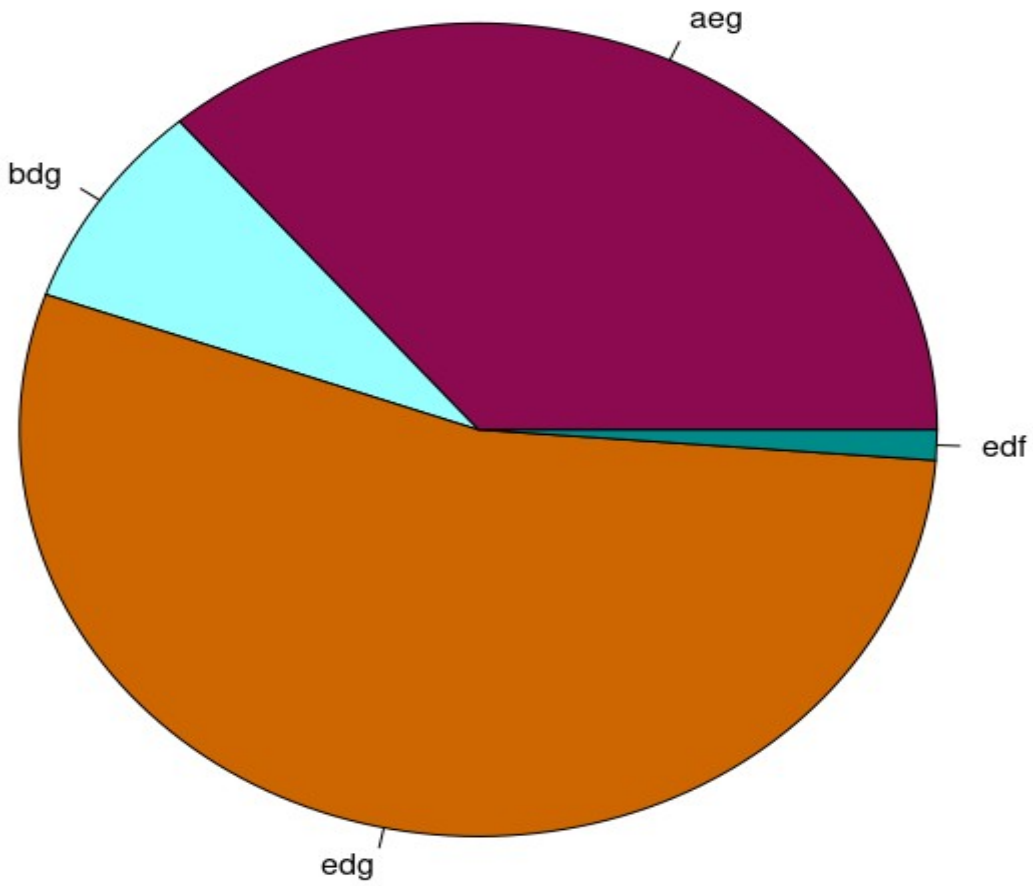


Рисунок 10 Частоты аллей групп крови системы *e* у мини-свиней ИЦиГ СО РАН.

13.3. Гистограммы

В виде гистограмм принято изображать особенность распределения количественных признаков, таких как удой, среднесуточный прирост, яйценоскость и т.п. Простейшим инструментом построения гистограммы в R является функция *hist()*, которой в качестве аргумента передаётся набор исходных данных. Для примера создадим гистограмму, показывающую распределение живой массы цыплят по данным таблицы ChickWeight Рисунок 11.

```
hist(ChickWeight$weight)
```

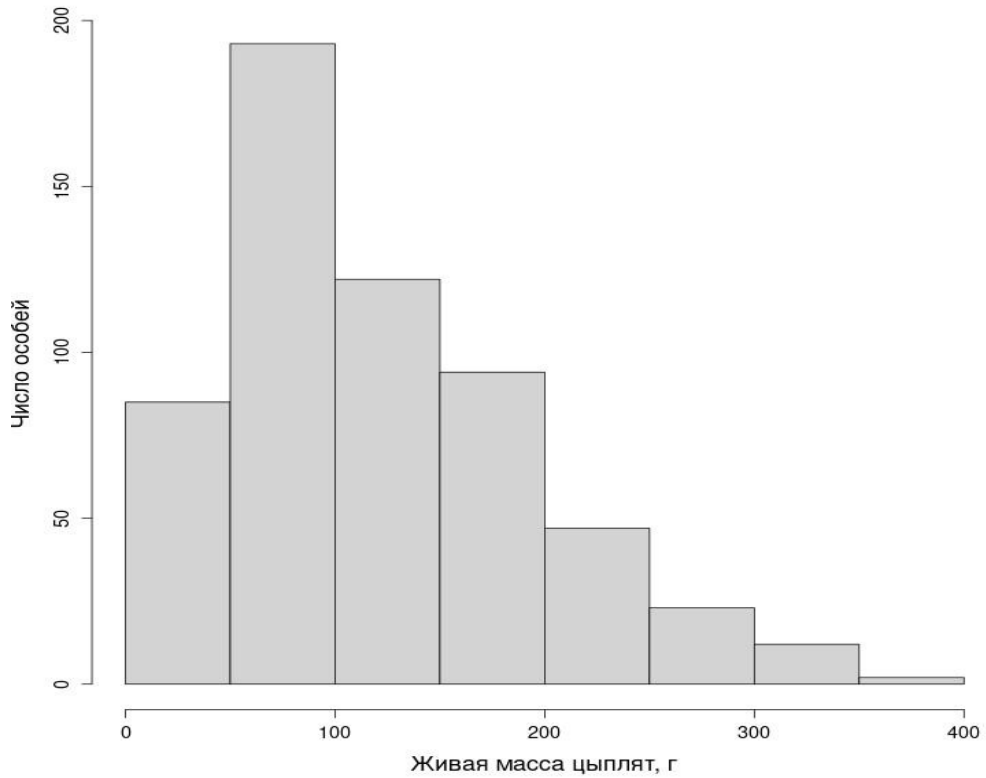


Рисунок 11 Распределение численности цыплят в зависимости о живой массы.

Количество столбцов на гистограмме определяется автоматически, но этот параметр можно контролировать аргументом *nclass*. Но как пишут сами разработчики, эффект от использования аргумента *nclass* нестабилен[46]. Важным аргументом функции *hist()* является *freq*. Это логический аргумент, по умолчанию равный TRUE. В таком случае на гистограмме представлена абсолютная встречаемость, выраженная в количестве наблюдений. Если *freq* = FALSE, тогда гистограмма показывает частоты, выраженные в долях единицы[18]. Гистограммы удобно применять для визуального сравнения фактического наблюдения с нормальным [Мастицкий С.Э., Шитиков В.К., 2023; Leboucher, Lowitz, 1978; Scott, 2010]. Для наглядности, обычно на рисунке рядом с гистограммой строят линию, показывающую нормальное распределение. Рассмотрим это на примере распределения веса цыплят из таблицы ChickWeight. Для нача-

46 https://www.rdocumentation.org/packages/graphics/versions/3.6.2/topics/hist

ла, установим параметры набора данных о живой массе цыплят, для чего удобно будет воспользоваться функцией *describe()* из пакета *psych* [Revelle, 2025].

```
library(psych)
> describe(ChickWeight$weight)
   vars   n   mean    sd median trimmed   mad min max range skew kurtosis
se
X1    1 578 121.82 71.07    103  113.18 69.68  35 373   338 0.96     0.34
2.96
```

Из результата функции мы узнали необходимые параметры: среднюю арифметическую (mean), стандартное отклонение (sd) и объём выборки (n). Используя полученные данные строим вектор с параметрами нормального распределения, который в дальнейшем используем для построения линии сравнения. Дополнительно настраиваем цвет и толщину линии, подписи к осят координат и подписи к ним. Убираем подпись к рисунку используя main = NULL Рисунок 12.

```
a <- rnorm(mean = 121.82, n = 578, sd = 71.07)
jpeg("Путь к папке/Fig4.jpeg", width = 780, height = 750)
hist(ChickWeight$weight, freq = FALSE, xlab = "Живая масса цыплят, г",
cex.lab = 2.1, cex.axis = 2, ylab = "Встречаемость", main = NULL)
lines(density(a), col = "red", lwd = 2.5)
dev.off()
```

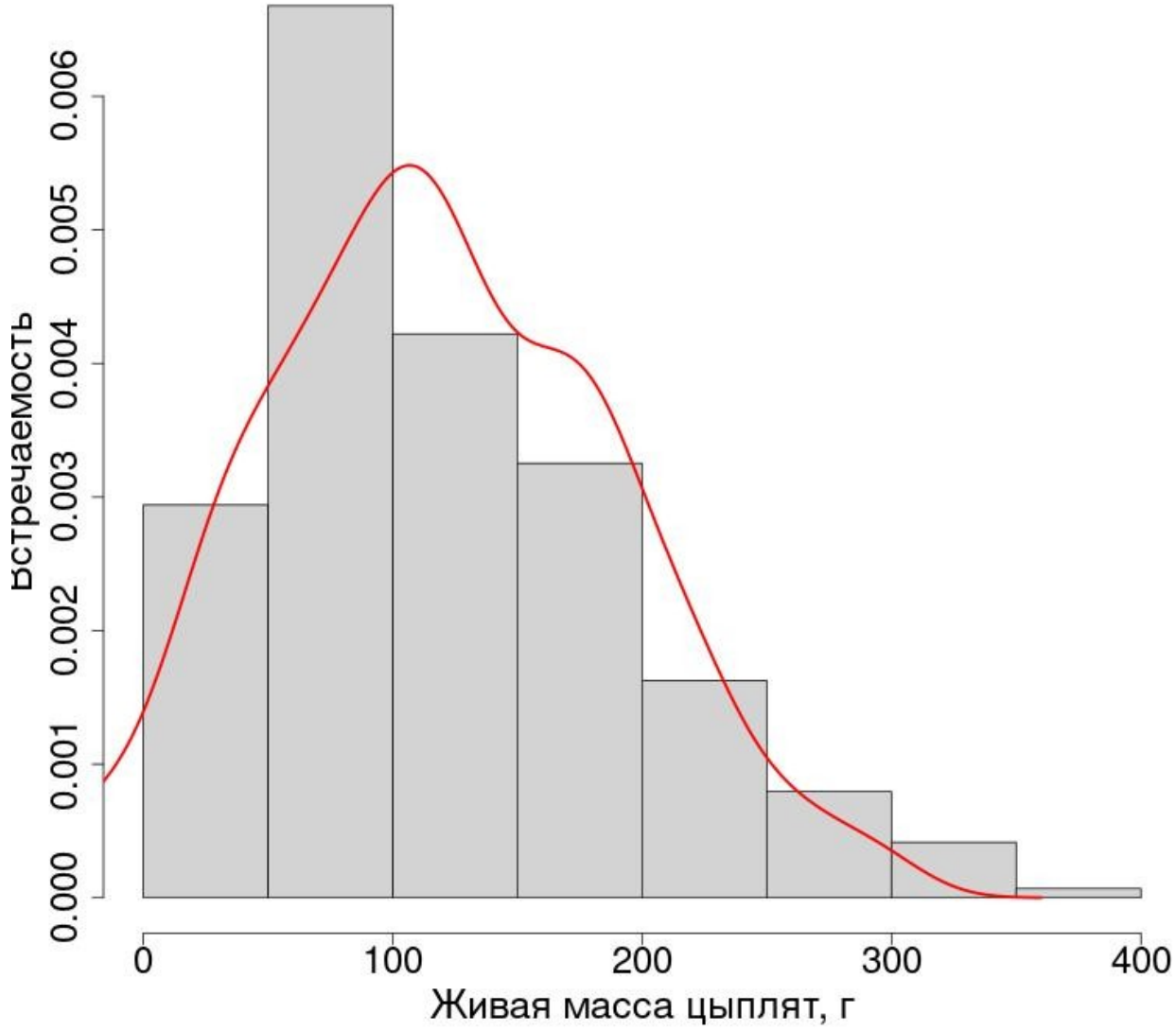


Рисунок 12 Визуальное сравнение распределения живой массы цыплят из таблицы ChickWeight с нормальным

13.4. Диаграммы рассеяния

С помощью диаграмм рассеяния принято показывать взаимосвязь между признаками или зависимость переменной от предиктора [Айвазян и др., 1989]. Другими словами, диаграммы рассеяния представляют собой инструменты корреляционного и регрессионного анализов. В базовой части R построения диаграммы рассеяния осуществляется при помощи функции *plot(x, y)*, где *x* и *y* — наборы исходных данных, признаки, между которыми проводится анализ на зависимость. Важный элемент управления диаграммой рассеяния является выбор маркёра. В функции *plot()* выбор маркёра производится присвоением аргументу *pch* цифр от 1 до 26 Рисунок 13. По умолчанию, в качестве аргумента pch присваивается тип маркёра, соответствующий номеру 1 Рисунок 13. Значимым аргументом функции *plot()* является *type*, определяющий тип соединения между маркёрами Рисунок 14. Для диаграммы рассеяния лучше всего подходит *type* = "p", используемый по умолчанию.

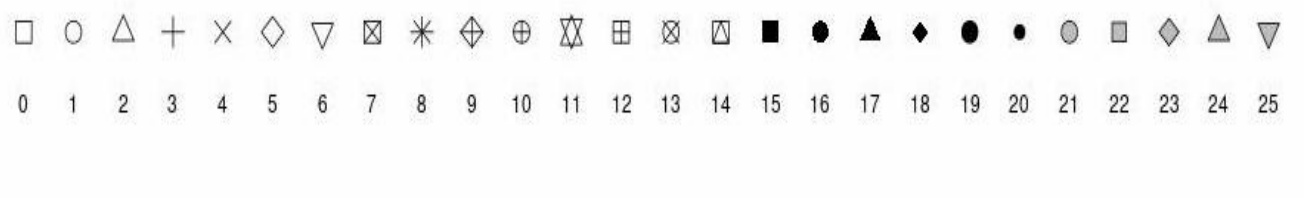


Рисунок 13 Типы маркёров функции *plot()*

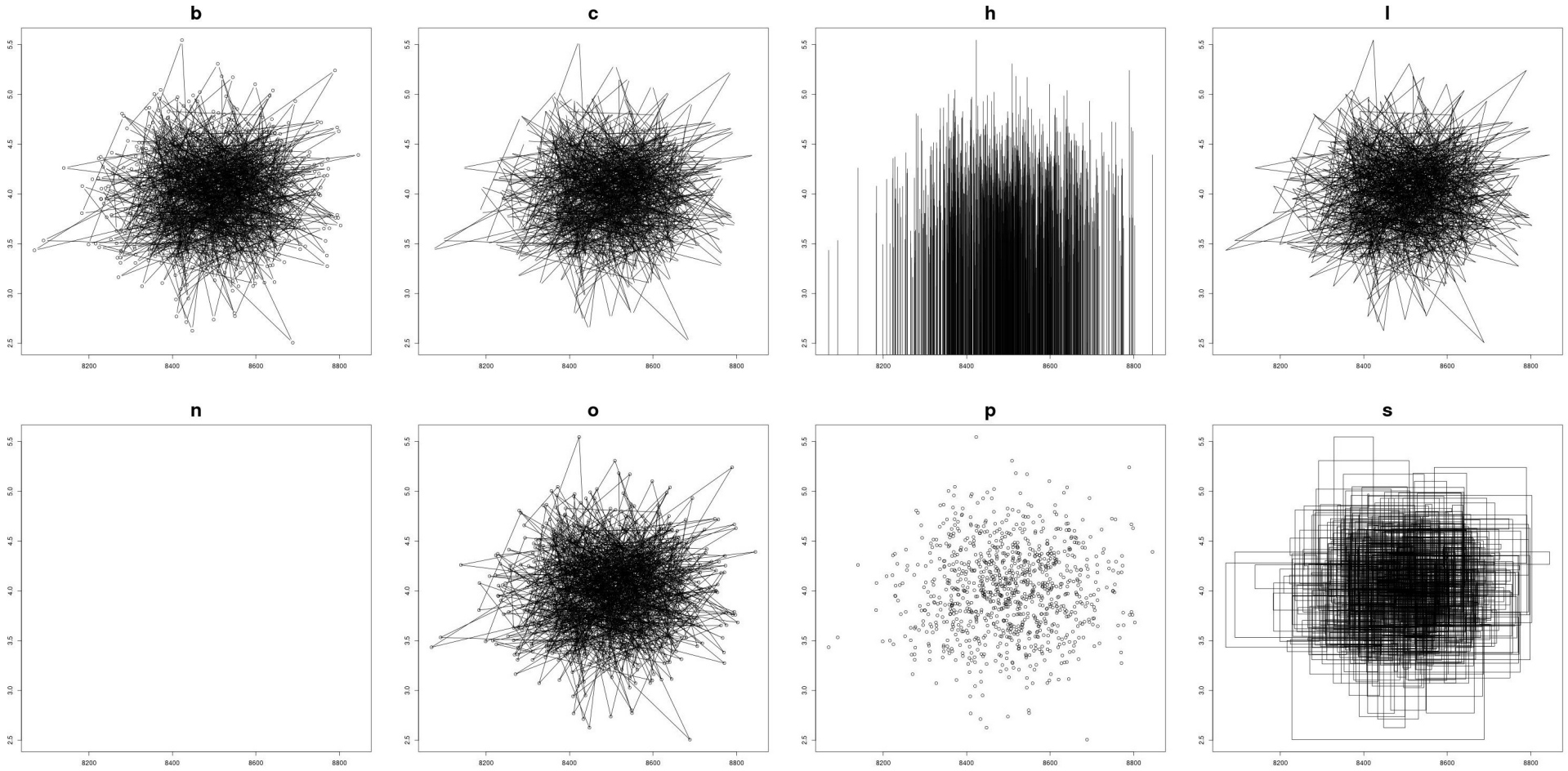


Рисунок 14 Типы соединения маркеров в зависимости от выбранного аргумента *type* функции *plot()*

Для визуализации корреляции вполне достаточно использовать простейший код *plot(x, y)*. Однако при визуализации регрессии нужно помнить некоторые нюансы. Первое, предиктору всегда соответствует ось *x*, а зависимой переменной — *y*. Второе, для визуализации регрессии важно оценить угол наклона и плотность точек вдоль линии тренда. Простейшим способом показать тренд является функция *abline()*, которой в качестве аргумента задаётся заранее построенная модель. Рассмотрим визуализацию регрессии на примере зависимости канадского индекса племенной ценности быка LPI от года рождения животного [Alcantara и др., 2022]. Данные были взяты из базы открытого доступа Canadian Dairy Network за август 2024 года[47]. Получившаяся диаграмма рассеяния изображена на Рисунок 15. Код приведён ниже.

```
library(data.table)
library(stringr)
cdn <- fread("/allobgepa2408_ho.csv")
cdn$Y <- str_sub(cdn$`BIRTH DATE`, 1, 4)
cdn$Y <- as.integer(cdn$Y)
cdn <- cdn[cdn$Y >= 2000,]
mod <- lm(cdn$LPI~cdn$Y)
plot(cdn$Y, cdn$LPI, xlab = "Год рождения быка", ylab = "LPI", cex.lab = 2.1,cex.axis = 2)
abline(mod, col = "red", lwd = 3.5)
```

47 https://lactanet.ca/genetique/evaluations-genetiques/

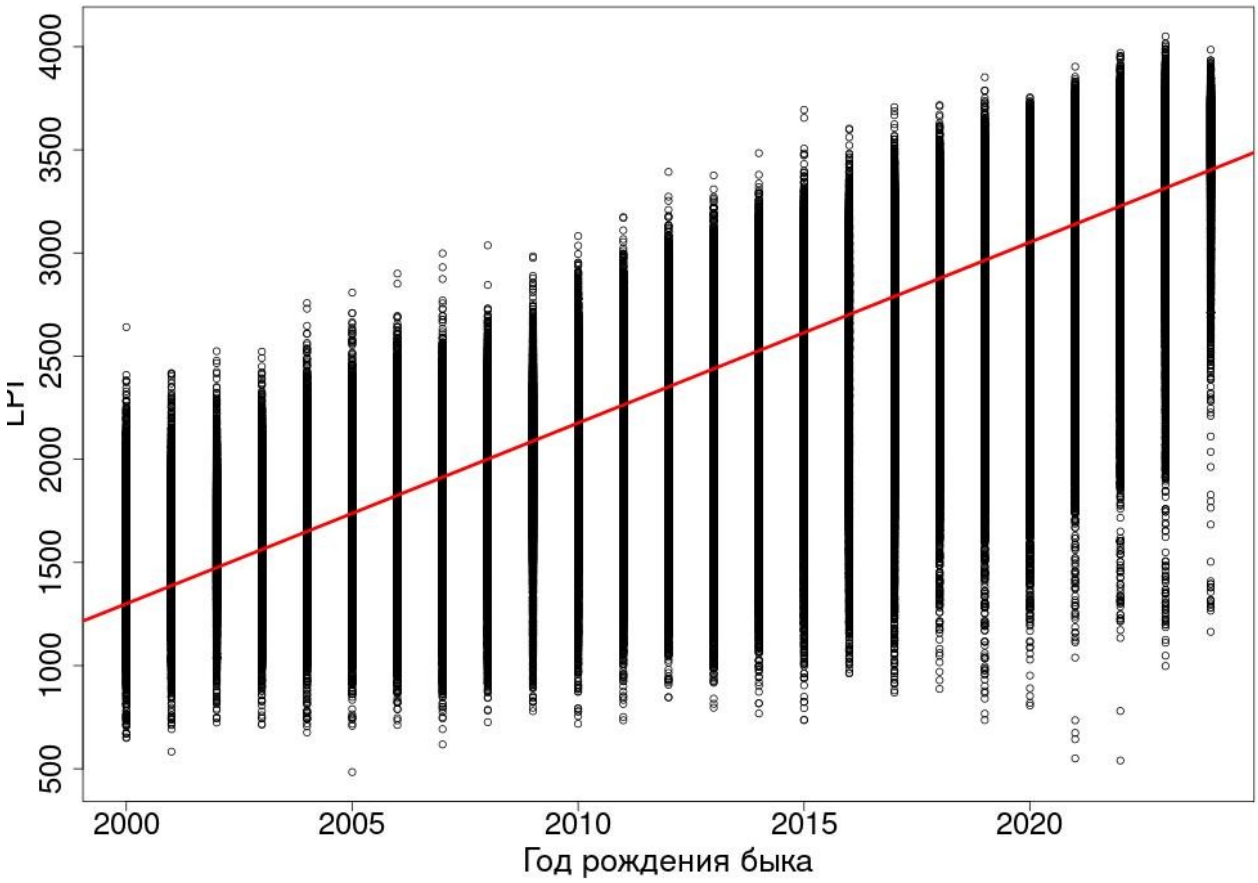


Рисунок 15 Визуализация зависимости LPI от года рождения быка

Что бы разместить несколько групп на одном рисунке необходимо проделать предварительную работу. Допустим, по данным таблицы *iris* нам нужно изобразить зависимость ширины листа от длины листа, при этом обозначив на графике вид ириса. Для этого наборы с исходными данными, передаваемые через аргументы *x* и *y*, нужно представить в виде матриц.

```
x_data <- data.frame(
           setosa = iris$Sepal.Length[iris$Species == "setosa"],
           vigrinica = iris$Sepal.Length[iris$Species == "virginica"],
           versicolor = iris$Sepal.Length[iris$Species == "versicolor"]
                        )
y_data <- data.frame(
              setosa = iris$Petal.Length[iris$Species == "setosa"],
              vigrinica = iris$Petal.Length[iris$Species == "virginica"],
              versicolor = iris$Petal.Length[iris$Species == "versicolor"]
                          )
x_data <- as.matrix(x_data)
y_data <- as.matrix(y_data)
```

Полученные матрицы используем в качестве аргументов с исходными данными. Для большей наглядности изобразим виды ирисов разноцветными значками разной формы Рисунок 16. Добавим легенду с использованием функции *legend()*. В качестве первого аргумента *x* передаётся позиция легенды, как показано на Рисунок 17.

```
plot(x_data, y_data,
     xlab = "Длина листа, см",
     ylab = "Ширина листа, см",
     col = c("red", "blue", "green"),
     pch = c(1, 2, 3))

legend(x = "bottomright",
  legend = c("setosa", "versicolor", "virginica"),
       col = c("red", "blue", "green"),
       pch = c(1, 2, 3)
       )
```

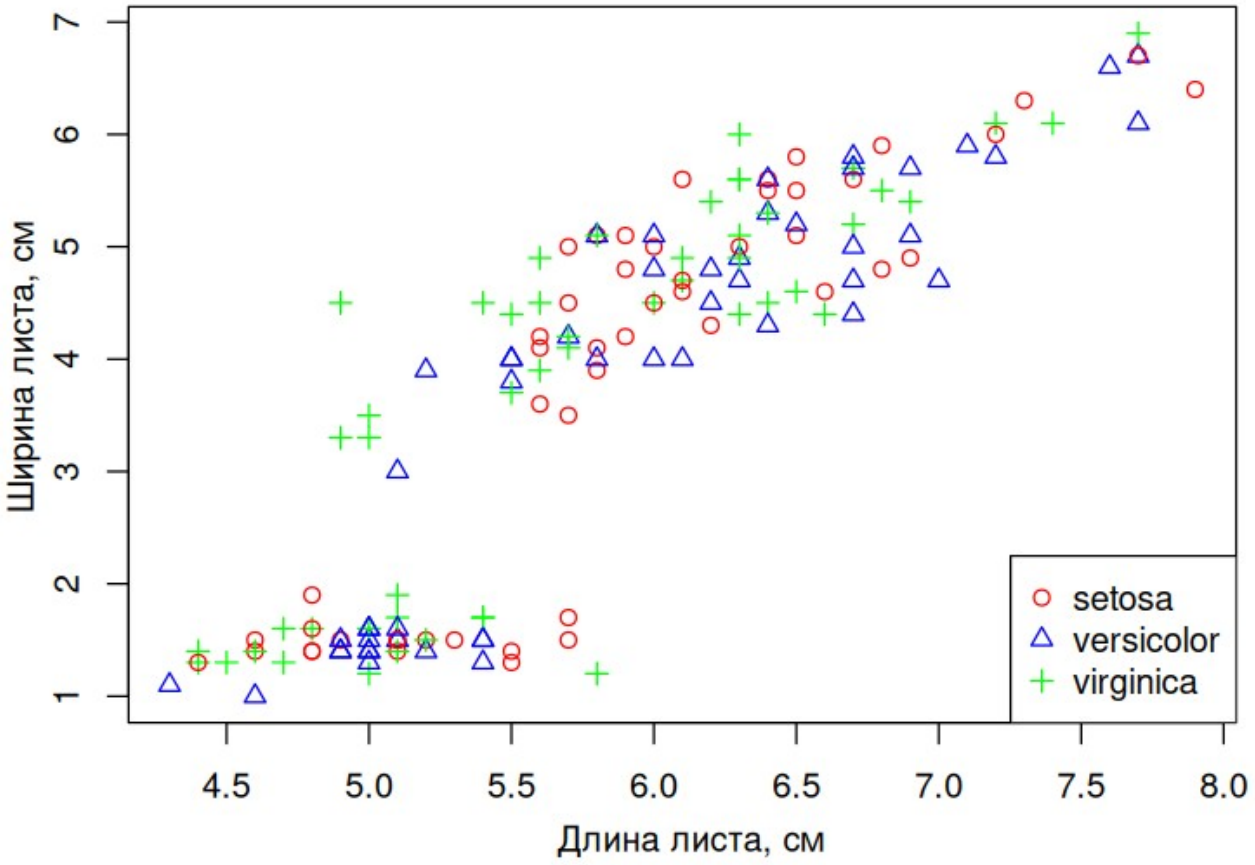


Рисунок 16 Соотношение длины и ширины листа у разных видов ирисов

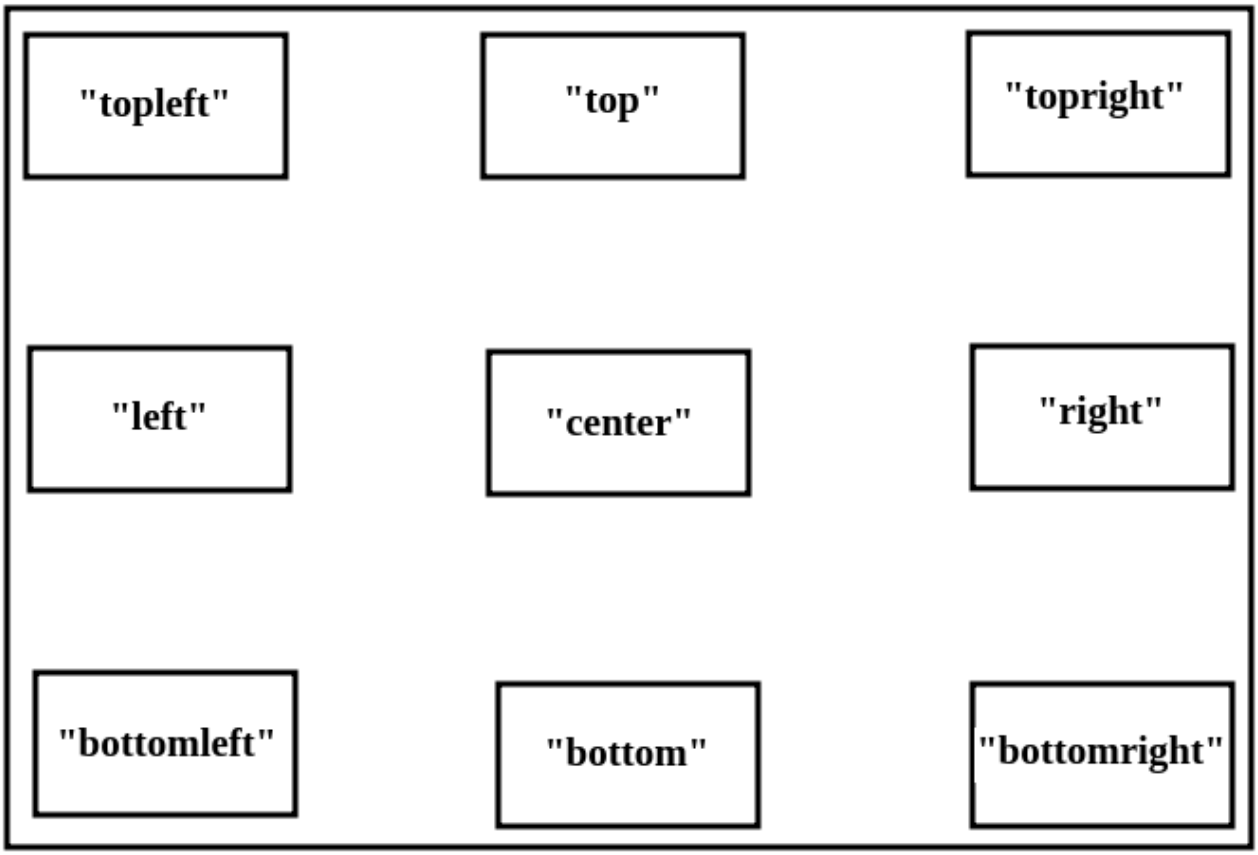


Рисунок 17. Схема расположения легенды согласно аргументу *legend*, функции *legend()*

13.5. Графики

На графиках принято изображать динамику какого-либо показателя [Лакин, 1990]. Наиболее простым способом построения графиков в R является функция *plot()* с аргументом *type = "b"* или *type = "c"*. Важно, что бы данные для оси x были предварительно отсортированы. Рассмотрим построение графика на примере динамики многоплодия в стаде мини-свиней ИЦиГ СО РАН с 2013 по 2018 годы Результаты исследований были опубликованы ранее [Nikitin и др., 2020].

```
NS <- c(6.89, 7.52, 7.41, 6.77, 6.31, 6.63)
Y <- c(2013, 2014, 2015, 2016, 2017, 2018)
mod <- lm(NS~Y)
plot(Y, NS, type = "b", lwd =3.5, xlab = "Год", ylab = "Многоплодие,
гол.", cex.lab = 1.6,cex.axis = 1.5)
abline(mod, col = "red", lwd = 3.0)
dev.off()
```

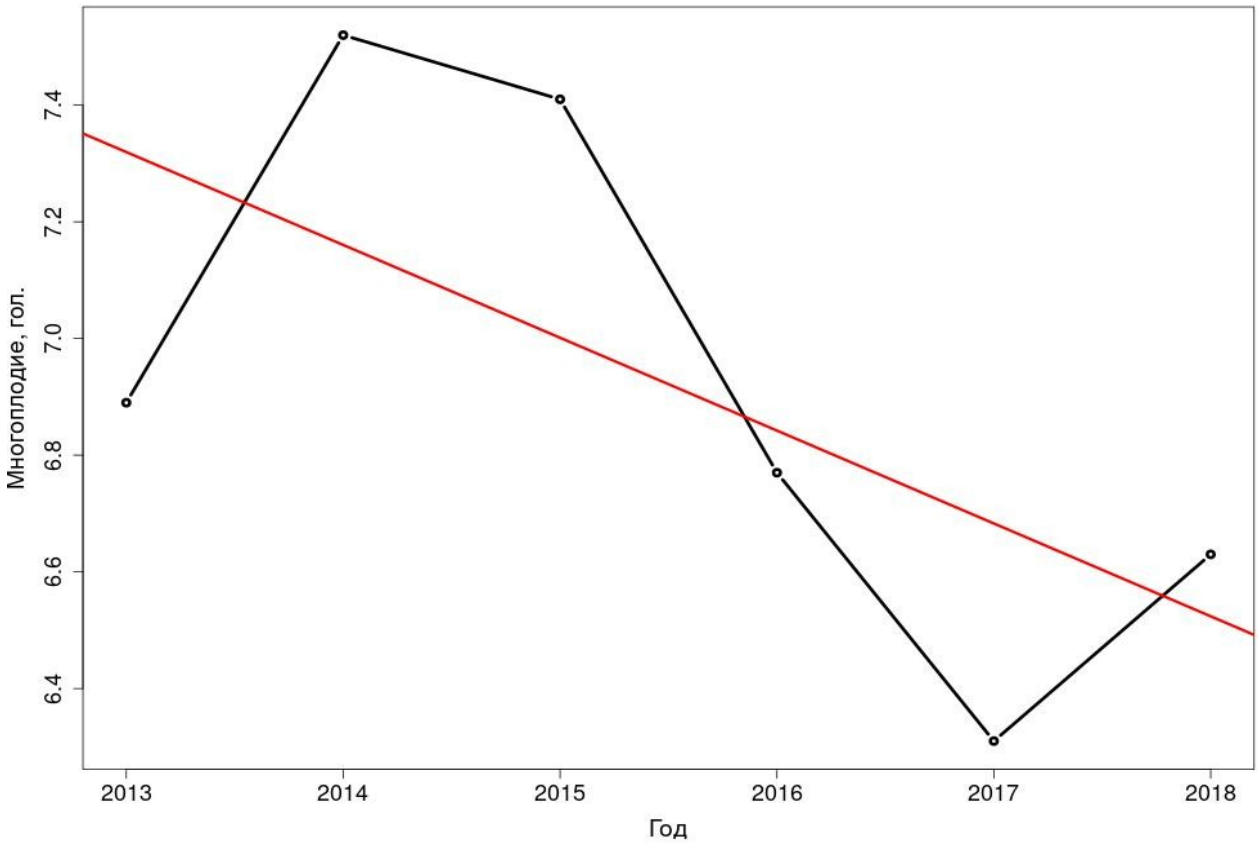


Рисунок 18 Динамика многоплодия в стаде мини-свиней ИЦиГ СО РАН с 2013 по 2018 годы. Красной линией обозначен линейный тренд.

В ряде случаев на одном рисунке необходимо на одном рисунке показать два и более графиков. Для этого среди базовых функций R лучше всего использовать комбинацию функций *plot()* и *lines()*. В качестве исходных данных *lines()* как и *plot()* принимает значения осей x и y соответственно. Следует помнить, что *lines()* не работает «сама по себе». С её помощью можно лишь нанести дополнительную график к уже имеющемуся. Используя функцию *lines()* на одном и том же рисунке можно показать неограниченное количество графиков. Первый строиться функцией *plot()*, второй и последующие — *lines()*. Важно, что по умолчанию функции *plot()* и *lines()* используют разные типы маркёров. Функция *plot()* по умолчанию использует маркёр №1 Рисунок 13, в то время как *lines()* по умолчанию не использует маркёры вообще. Если необходимо, что бы тип маркёров был одинаковым, достаточно просто прописать это в коде при по-

мощи аргумента *type* = . Разберём на конкретном примере. Для этого воспользуемся данными о численности поросят мини-свиней ИЦиГ СО РАН разной масти, рождённых в период с 2013 по 2020 год Таблица 10. Данные были частично представлены в одной из предыдущих работ [Shatokhin и др., 2025].

Таблица 10 Данные о численности поросят мини-свиней ИЦиГ СО РАН разной масти

Год	Масть			
	Агути	Чёрные	Чёрно-пёстрые	Белые
2013	46	151	96	195
2014	46	61	79	127
2015	55	79	56	151
2016	141	93	80	102
2017	64	41	89	72
2018	48	30	66	58
2019	124	71	49	66
2020	138	46	55	42

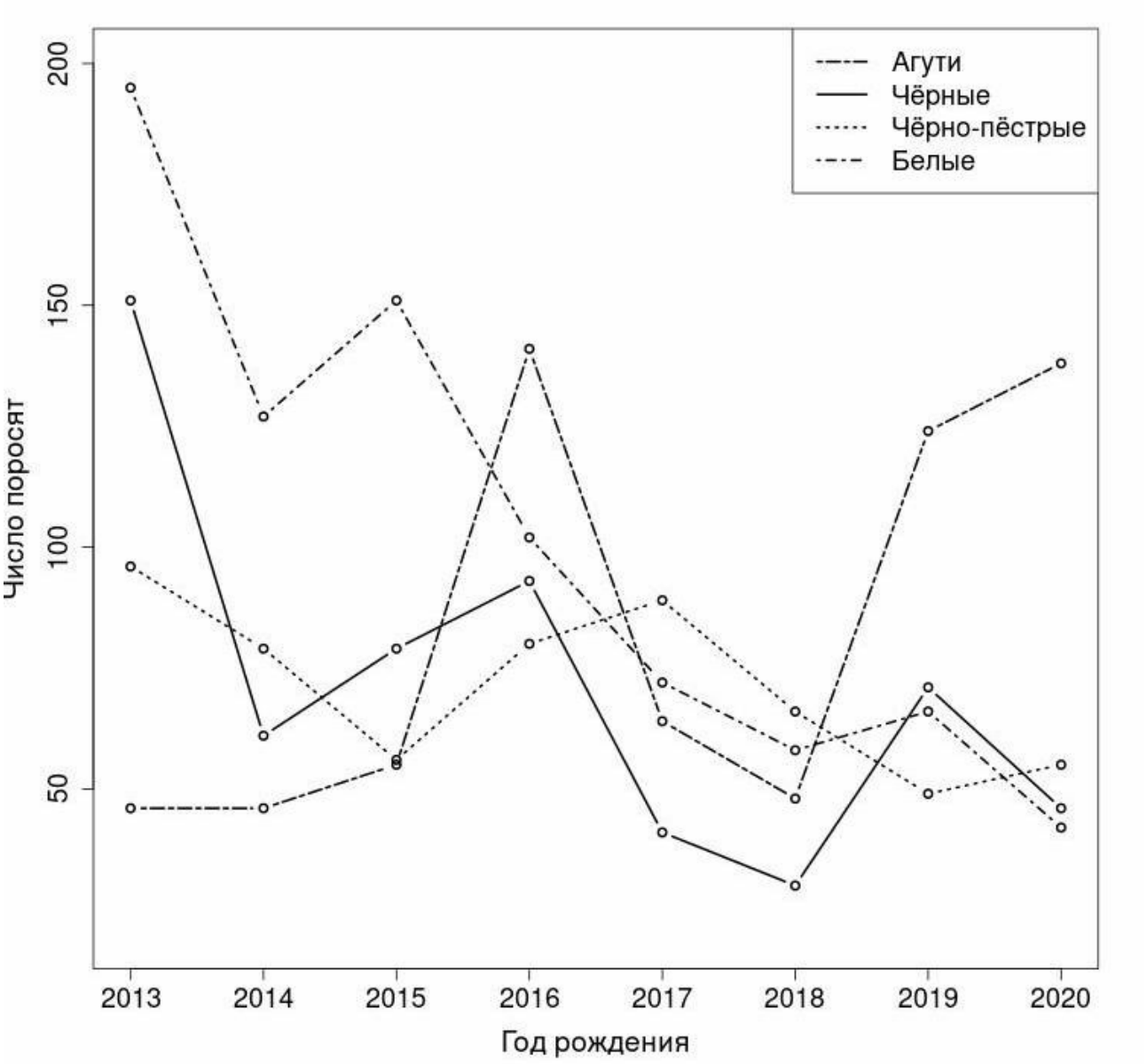
Для начала построим рисунок, на который нанесём график, показывающий численность поросят масти агути. Сразу же обозначим тип графика. Так как численность поросят существенно отличается в зависимости от типа масти, мы сразу же установим границы по оси y. Отрегулируем тип линии при помощи аргумента lty. И наконец, добавим легенду.

```
plot(df$Y, df$agouti, type = "b", ylim = c(20, 200), lty = 6, lwd = 2)
lines(df$Y, df$black, type = "b", lty = 1, lwd = 2)
legend("topright", c("Агути", "Чёрные"), lwd = 2, lty = c(6, 1))
```

Добавим графики динамики рождаемости поросят других окрасок, подкорректируем легенду и финальный рисунок готов.

```
plot(df$Y, df$agouti, type = "b", ylim = c(20, 200),
      lty = 6, lwd = 2, ylab = "Число поросят", xlab = "Год рождения",
      cex.lab = 1.6, cex.axis = 1.5)
lines(df$Y, df$black, type = "b", lty = 1, lwd = 2)
lines(df$Y, df$spotted, type = "b", lty = 3, lwd = 2)
```

```
lines(df$Y, df$white, type = "b", lty = 4, lwd = 2)
legend("topright", c("Агути", "Чёрные", "Чёрно-пёстрые", "Белые"),
       lwd = 2, lty = c(6, 1, 3, 4), cex = 1.5)
```



13.6. Коробчатые диаграммы

Коробчатая диаграмма, она же ящик с усами или boxplot в англоязычной литературе является достаточно информативным инструментом визуализации изменчивости количественного признака Рисунок 19. Позволяет оценить медиану, межквантильный размах и допустимые границы признака. Показывает выбросы при их наличии. Идентификация выбросов и предельно допустимых границ изменчивости признака (верхняя и нижняя) при построении коробчатой

диаграммы с использованием базовой функцией *boxplot()* оценивается при помощи фильтра Тьюки, который определяется как `+/-1.58 IQR/sqrt(n)` [Chambers и др., 1983; McGill, Tukey, Larsen, 1978]. Следует добавить, что верхняя и нижняя границы не являются фактическими лимитами признака.

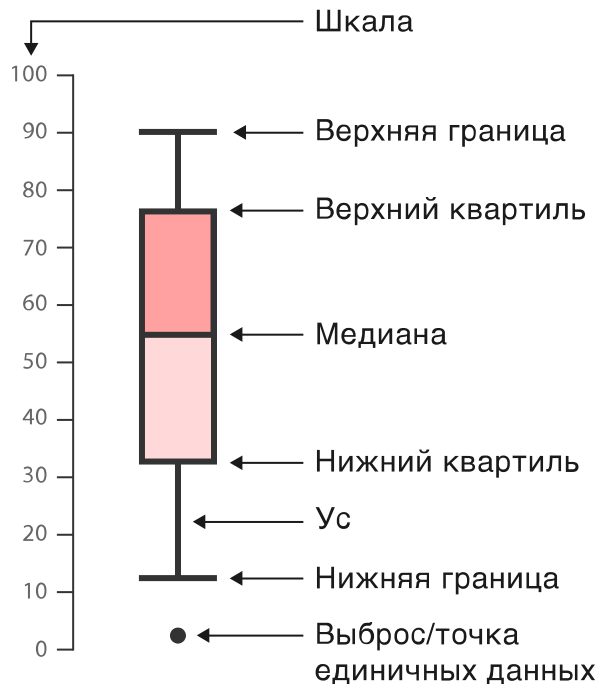


Рисунок 19 Структура коробчатой диаграммы[48]

Технически, коробчатая диаграмма в R при помощи базовой функции *boxplot()* строится при помощи передачи функции набора исходных данных. Получившееся изображение демонстрирует изменчивость только какого-либо одного признака 127. Для примера, рассмотрим изменчивость данных температуры тела бобра из набора данных *beaver1* [Reynolds, 1994]. Код выглядит следующим образом:

```
boxplot(beaver1$temp, cex.axis = 1.5)
```

48 https://datavizcatalogue.com/RU/metody/diagramma_razmaha.html

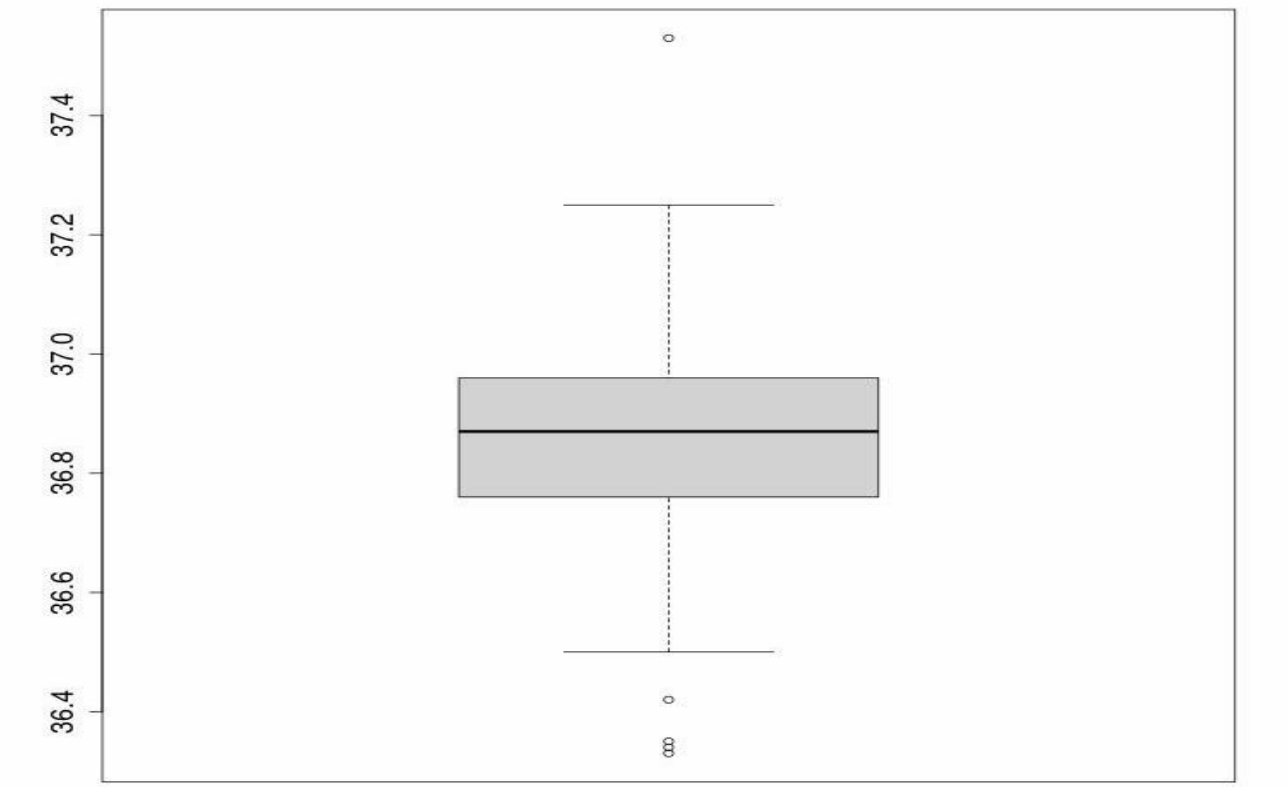


Рисунок 20 Коробчатая диаграмма, показывающая изменчивость температуры тела бобра из набора данных *beaver1*

Более информативным является сравнение изменчивости признака, изученного в двух и более выборках. В животноводстве это может быть удой, среднесуточный прирост, яйценоскость и любой другой продуктивный показатель, наблюдаемый в разных группах. Таковыми группами могут быть: разные хозяйства, год наблюдения, масть и т.д. В качестве примера рассмотрим визуализацию изменчивости длины листа ирисов разных видов. Исходные данные хранятся во встроенной таблице iris. Важно, что при построении коробчатой диаграммы для 2-х и более групп необходимо представление исходных данных в виде широкой таблицы, где каждый столбец соответствует отдельной группе. Так у нас получился чёрно-белый вариант коробчатой диаграммы Рисунок 21 (слева)

```
df <- data.frame(rep(1:50, 3), iris[,1], iris[,5])
colnames(df) <- c("ID", colnames(iris[c(1,5)]))
wide <- reshape(data = df,
```

```
      direction = "wide", #выбирает тип трансформации
    idvar = "ID", #столбец с индивидуальными идентификаторами записей
      timevar = "Species", #столбец с именами категорий
      v.names = "Sepal.Length" #столбец с записями зависимой переменой
                )
colnames(wide) <- c("ID", "Setosa", "Vercicolor", "Virginica")
wide$ID <- NULL
boxplot(wide)
```

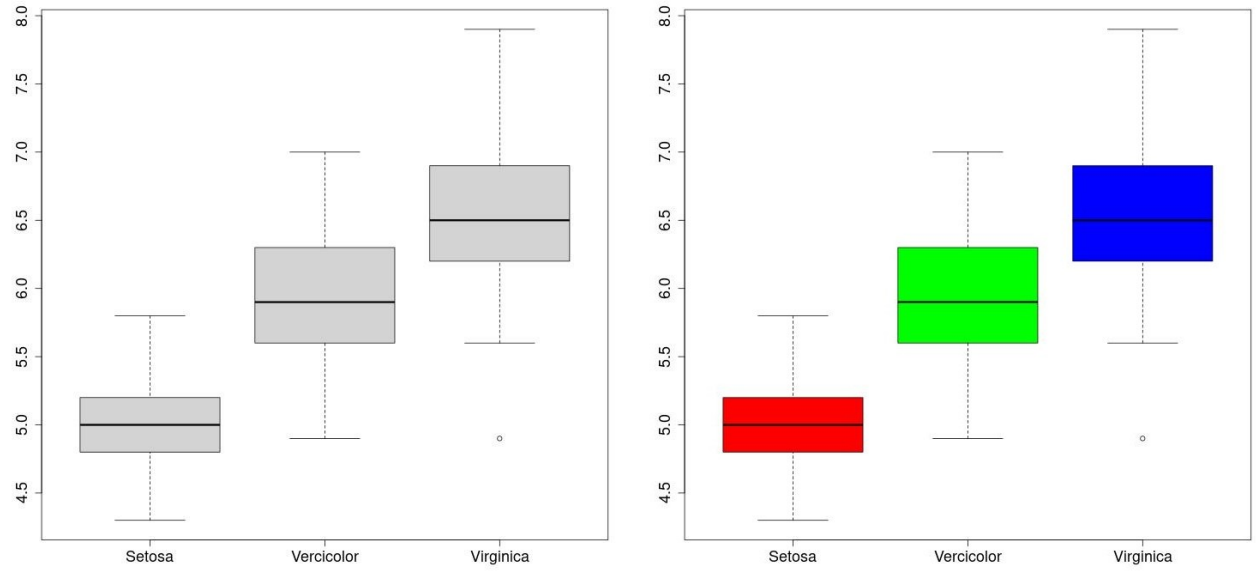


Рисунок 21 Визуализация изменчивости длины листа разных видов ирисов

Добавим цвета Рисунок 21 (справа)

```
boxplot(wide, col = c("red", "green", "blue"), cex.lab = 1.6, cex.axis = 1.5)
```

13.7. Диаграммы квантиль-квантиль

Диаграммы квантиль-квантиль или qqplot появились как инструмент анализа остатков моделей регрессии [Дрейпер, Смит, 1986; Abdul-Aziz, Luguterah, Saeed, 2020; Salinas Ruíz и др., 2023]. Впоследствии их стали использовать в качестве метода оценки нормальности распределения признака [Pleil, 2016]. В животноводстве их можно использовать для обнаружения не типичных значений признака [Камалдинов и др., 2022; Камалдинов и др., 2024]. Интерпретация графиков квантиль-квантиль интуитивно понятна. Чем ближе точки располагаются к линии аппроксимации, тем лучше выборочное распределение похоже на теоретическое. Точки данных не обязательно должны располагаться точно на пря-

мой. Вместо этого они должны лишь в целом следовать прямой, располагаясь выше и ниже неё в соответствии со случайной изменчивостью. Среди базовых функций R построение диаграмм квантиль-квантиль рекомендуется осуществлять при помощи функции *qqnorm()*. Линию проводят при помощи функции *abline()*. Рассмотрим создание диаграммы квантиль-квантиль на примере модели динамики LPI по данным CDN Рисунок 22.

```
library(data.table)
library(stringr)
cdn <- fread("/allobgepa2408_ho.csv")
cdn$Y <- str_sub(cdn$`BIRTH DATE`, 1, 4)
cdn$Y <- as.integer(cdn$Y)
cdn <- cdn[cdn$Y >= 2000,]
mod <- lm(cdn$LPI~cdn$Y)
qqnorm(residuals(mod),   xlab   =   "Теоретические   квантили",   ylab   =
"Фактические квантили", cex.lab = 1.6, cex.axis = 1.5, main = " ")
qqline(residuals(mod), lwd = 3, col = "red")
```

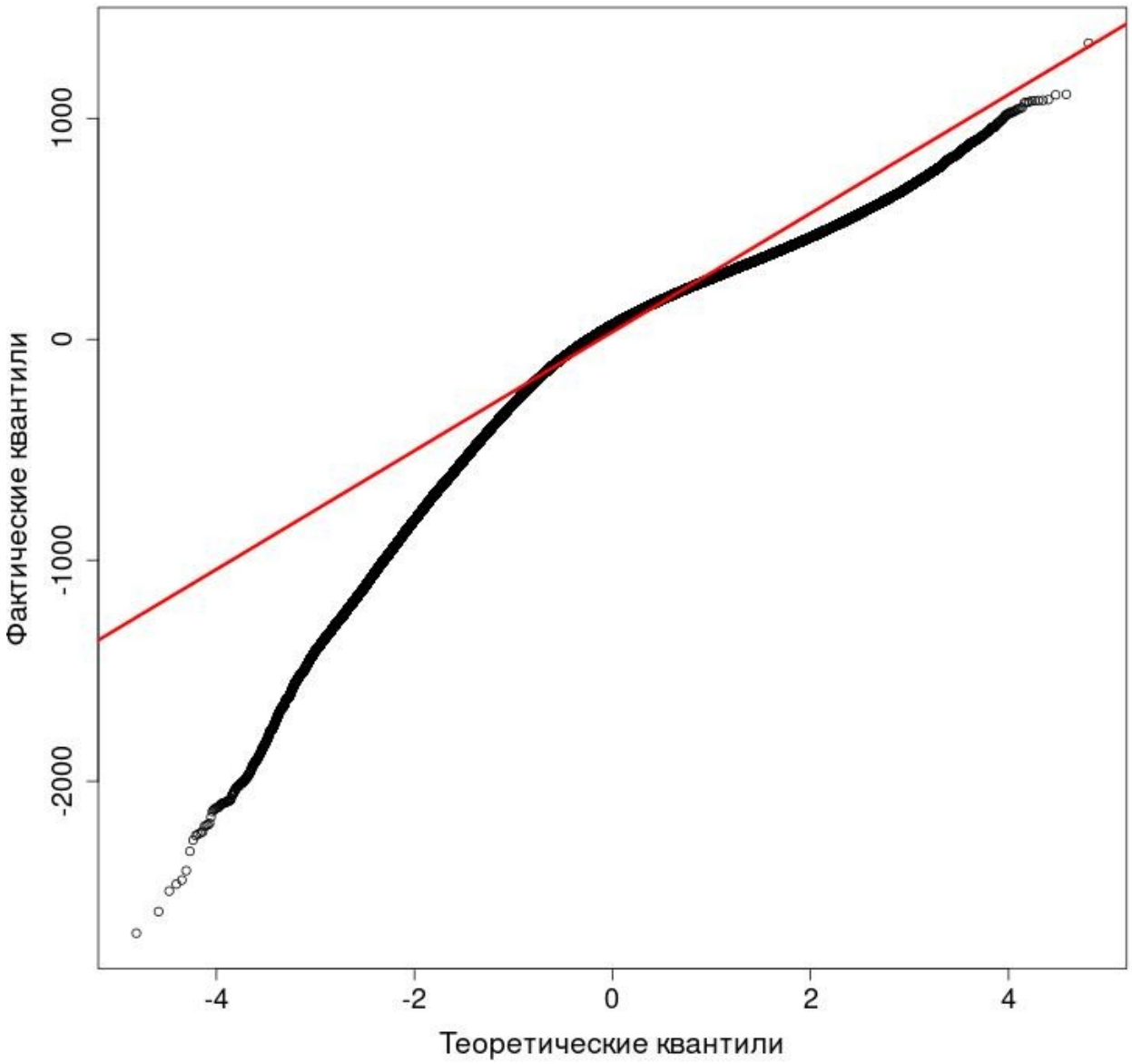


Рисунок 22 Диаграмма квантиль-квантиль на примере распределения остатков модели динамики индекса LPI

Получившийся рисунок Рисунок 22 демонстрирует отклонение фактического распределения остатков от нормального. Ниже приведён пример графика с распределением, не отличимым от нормальногоРисунок 23.

```
data <- rnorm(100)
qqnorm(data)
qqline(data, col = "blue")
```

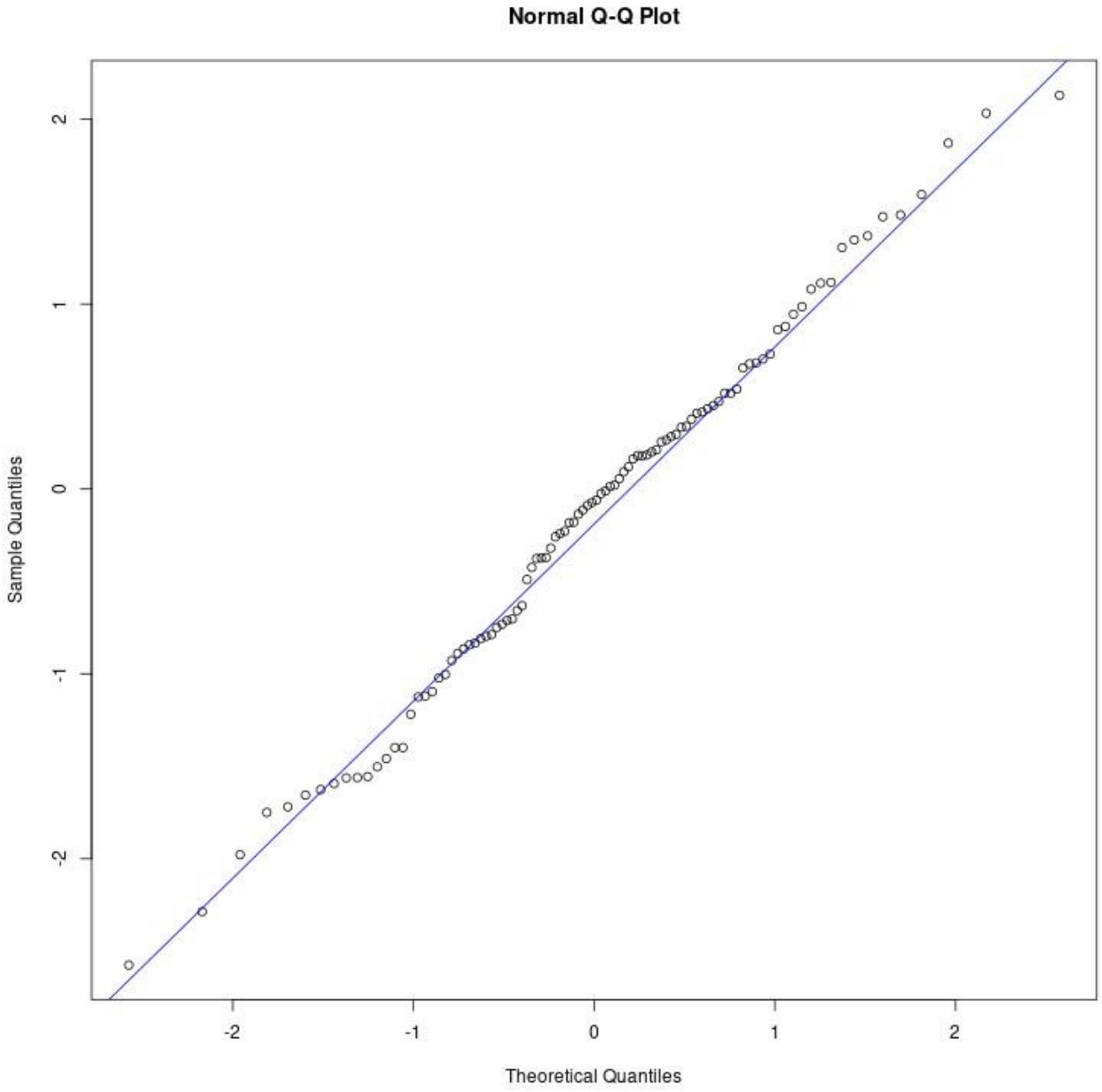


Рисунок 23 Пример диаграммы квантиль-квантиль, демонстрирующий нормальное распределение признака

Контрольные вопросы

1. С помощью какого типа диаграмм принято визуализировать доли и проценты
2. В чём отличие столбчатых диаграмм от гистограмм
3. Каким типом диаграмм можно визуализировать зависимость одного признака от другого
4. С помощью какой базовой функции можно построить круговую диаграмму в R

Практические задания

1. По данным таблицы *iris* постройте столбчатую диаграмму, показывающую ширину листа разных видов ирисов
2. По данным *HairEyeColor* постройте круговую диаграмму, показывающую долю мужчин с разным цветом глаз.
3. По данным *state.x77* покажите зависимость населения (Population) от индекса низких температур (frost)
4. По данным каталога *AltaGenetics* за август 2024 года покажите зависимость индекса TPI от прибавки по молоку. Изобразите быков рождённых в США и Канаде маркёрами разной формы и разного цвета
5. По данным таблицы *food_kaggle* изобразите содержание протеина в продуктах из курицы, говядины и свинины на коробчатой диаграмме
6. По данным таблицы *cat_traits* изобразите на столбчатой диаграмме среднюю живую массу кошек в зависимости от фенотипа масти. Количество животных в группе должно быть не менее 80.

Список используемой литературы

1. Айвазян С. А. и др. Прикладная статистика. Классификация и снижение размерности. Москва: Финансы и статистика, 1989. 607 с.

2. Брюс П., Брюс Э. Практическая статистика для специалистов Data Science: Пер. с англ. СПб: O`REILLY, 2018. Вып. БХВ-Петербург. 308 с.

3. Дрейпер Н., Смит Г. Прикладной регрессионный анализ. Москва: Финансы и статистика, 1986. Вып. 2-е. 366 с.

4. Камалдинов Е. В. и др. Достоверность данных первичного зоотехнического учёта в молочном скотоводстве // Вестник НГАУ. 2022. Т. 63. № 2. С. 76–83.

5. Камалдинов Е. В. и др. Оценка качества генеалогических данных в племенных предприятиях Западной Сибири // Животноводство и кормопроизводство. 2024. Т. 107. № 4. С. 53–67.

6. Кузнецов В. М. Генетическая оценка молочного скота методом BLUP // Зоотехния. 1995. № 11. С. 8–15.

7. Кузнецов В. М. Разведение по линиям и голштинизация: методы оценки, состояние и перспективы // Проблемы биологии продуктивных животных. 2013. № 3. С. 25–79.

8. Лакин Г. Ф. Биометрия. Москва: Высшая школа, 1990. 352 с.

9. Мастицкий С. Э., Шитиков В. К. Статистический анализ и визуализация данных с помощью R. : ДМК Пресс, 2015. 496 с.

10. Меркурьева Е. К. Биометрия в селекции и генетике сельскохозяйственных животных. Москва: Колос, 1970. 214 с.

11. Никитин С. В. и др. Возрастная динамика многоплодия мини-свиней ИЦиГ СО РАН и ее связь с белой мастью // Известия Тимирязевской сельскохозяйственной академии. 2024. № 6. С. 141–161.

12. Петров А. Ф. и др. Роль фиксированных факторов В изменчивости удоя скота ирменского типа В условиях промышленного комплекса // Вестник НГАУ (новосибирский Государственный Аграрный Университет). 2021. Т. 61. № 4. С. 137–149.

13. Самсонов Т. Е. Визуализация и анализ географических данных на языке R. Москва: Географический факультет МГУ, 2017.

14. Свечин К. Б. Индивидуальное развитие сельскохозяйственных животных. Киев: Урожай, 1976. 288 с.

15. Тьюки Дж. Анализ результатов наблюдений. Разведочный анализ. Москва: Мир, 1981. 346 с.

16. Уикем Х., Гроссер М., Буманн Х. R. К вершинам мастерства. Москва: ДМК Пресс, 2024. 748 с.

17. Филд Д. Регулярные выражения. СПб: Символ Плюс, 2008. Вып. 3. 608 с.

18. Шатохин К. С. и др. Генетические особенности миниатюрных свиней ИЦиГ СО РАН // Вестник НГАУ. 2014. Т. 30. № 1. С. 75–80.

19. Шипунов А. Б. и др. Наглядная статистика. Используем R! : практическое руководство. Москва: ДМК Пресс, 2017. 298 с.

20. Abadi D. J., Boncz P. A., Harizopoulos S. Column-oriented database systems // Proc. VLDB Endow. 2009. Т. 2. № 2. С. 1664–1665.

21. Abdul-Aziz A. R., Luguterah A., Saeed B. I. I. Using residual estimators to detect outliers and potential controlling observations in structural equation modelling: qq plot approach // Open Journal of Statistics. 2020. № 10. С. 905–914.

22. Alcantara L. M. и др. Conformation traits of Holstein cows and their association with a Canadian economic selection index // Can. J. Anim. Sci. 2022. Т. 102. № 3. С. 490–500.

23. Bailey D. W. The Inheritance of Maternal Influences on the Growth of the Rat // 1953.

24. Barry T. Collections in R: Review and Proposal // The R Journal. 2018. Т. 10. № 1. С. 455.

25. Bloomfield V. A. Using R for Numerical Analysis in Science and Engineering. Boca Raton, London, New York: CRC Press/Taylor & Francis Group, 2014.

26. Braun W. J. Modelling and Simulation in R. Vancouver, BC Canada: University of British Columbia, 2021. 242 с.

27. Campbell N., Mahon R. A multivariate study of variation in two species of rock crab of the genus Leptograpsus // Aust. J. Zool. 1974. Т. 22. № 3. С. 417.

28. ÇeliKyürek H. и др. Database Usage and Its Importance in Livestock // 2019. Т. 9. № 2. С. 117–121.

29. Chambers J. M. и др. Graphical Methods for Data Analysis. New York: Chapman and Hall/CRC, 1983. 410 с.

30. Chambers J. M. Facets of R // The R Journal. 2009. Т. 1. № 1.

31. Chang W. R graphics cookbook. Beijing Cambridge Farnham Köln Sebastopol Tokyo: O'Reilly, 2013. 396 с.

32. Cole J. B., VanRaden P. M. Symposium review: Possibilities in an age of genomics: The future of selection indices // Journal of Dairy Science. 2018. Т. 101. № 4. С. 3686–3701.

33. Cosentino V., Cánovas Izquierdo J. L., Cabot J. A Systematic Mapping Study of Software Development With GitHub // IEEE Access. 2017. Т. 5. С. 7173–7192.

34. Crawley M. J. The R book. Chichester (GB): J. Wiley, 2007.

35. Crowder M. J., Hand D. J. Analysis of repeated measures. LONDON ■ NEW YORK • TOKYO • MELBOURNE • MADRAS: CRC Press, 1990. 272 с.

36. Escamilla E. и др. The Rise of GitHub in Scholarly Publications // Linking Theory and Practice of Digital Libraries / под ред. G. Silvello и др. Cham: Springer International Publishing, 2022. С. 187–200.

37. Fieller N. Basics of Matrix Algebra for Statistics with R. University of Sheffield, UK: CRC Press/Taylor & Francis Group, 2016. 239 с.

38. Fisher R. A. The use of multiple measurements in taxonomic problems // Annals of Eugenics. 1936. Т. 7. № II. С. 179–188.

39. Gauthier J. и др. A brief history of bioinformatics // Briefings in Bioinformatics. 2019. Т. 20. № 6. С. 1981–1996.

40. Gentleman R., Ihaka R. Lexical Scope and Statistical Computing // Journal of Computational and Graphical Statistics. 2000. Т. 9. № 3. С. 491–508.

41. Giorgi F. M., Ceraolo C., Mercatelli D. The R Language: An Engine for Bioinformatics and Data Science // Life. 2022. Т. 12. № 5.

42. Gousios G., Pinzger M., Deursen A. van. An exploratory study of the pull-based software development model // Proceedings of the 36th International Conference on Software Engineering ICSE 2014. New York, NY, USA: Association for Computing Machinery, 2014. С. 345–355.

43. Grolemund G., Wickham H. Dates and Times Made Easy with **lubridate** // J. Stat. Soft. 2011. Т. 40. № 3. С. 1–25.

44. Hesterberg T. Bootstrap // WIREs Computational Stats. 2011. Т. 3. № 6. С. 497–526.

45. Hornik K. The Comprehensive R Archive Network // WIREs Computational Stats. 2012. Т. 4. № 4. С. 394–398.

46. Ibarz J. и др. How to train your robot with deep reinforcement learning: lessons we have learned // The International Journal of Robotics Research. 2021. Т. 40. № 4–5. С. 698–721.

47. Ihaka R. The R Project: A Brief History and Thoughts About the Future // 2022.

48. Ihaka R., Gentleman R. R: A Language for Data Analysis and Graphics // Journal of Computational and Graphical Statistics. 1996. Т. 5. № 3. С. 299–314.

49. Jiang J. и др. Why and how developers fork what from whom in GitHub // Empirical Software Engineering. 2017. Т. 22. № 1. С. 547–578.

50. Jordon J. и др. Synthetic Data -- what, why and how? // 2022.

51. Kietzmann P., Schmidt T. C., Wählisch M. A Guideline on Pseudorandom Number Generation (PRNG) in the IoT // ACM Comput. Surv. 2021. Т. 54. № 6.

52. Lawrence M. и др. Software for Computing and Annotating Genomic Ranges // PLoS Comput Biol. 2013. Т. 9. № 8. С. e1003118.

53. Leboucher G., Lowitz G. E. What a histogram can really tell the classifier // Pattern Recognition. 1978. Т. 10. № 5. С. 351–357.

54. Little R. J. A. A Test of Missing Completely at Random for Multivariate Data with Missing Values // Journal of the American Statistical Association. 1988. Т. 83. № 404. С. 1198–1202.

55. MacLennan B. J. Values and objects in programming languages // SIGPLAN Not. 1982. Т. 17. № 12. С. 70–79.

56. Maindonald J. H. Using R for data analysis and graphics introduction, code and commentary. Canberra, Australia: Centre for Mathematics and Its Applications, Australian National University, 2008. 96 с.

57. Martin A. D., Quinn K. M., Park J. H. **MCMCpack** : Markov Chain Monte Carlo in *R* // J. Stat. Soft. 2011. Т. 42. № 9.

58. Martin R. C. Clean code: a handbook of agile software craftsmanship. Upper Saddle River, NJ Munich: Prentice Hall, 2012. Вып. Repr. 431 с.

59. McGill R., Tukey J. W., Larsen W. A. Variations of Box Plots // The American Statistician. 1978. Т. 32. № 1. С. 12–16.

60. Meher P. K., Rustgi S., Kumar A. Performance of Bayesian and BLUP alphabets for genomic prediction: analysis, comparison and results // Heredity. 2022. Т. 128. № 6. С. 519–530.

61. Miglior F., Muir B. L., Van Doormaal B. J. Selection Indices in Holstein Cattle of Various Countries // Journal of Dairy Science. 2005. Т. 88. № 3. С. 1255–1263.

62. Mrode R. A., Pocrnic I. Linear Models for the Prediction of the Genetic Merit of Animals. Boston, USA: CABI, 2023. 341 с.

63. Neves K., Amaral O. B. Addressing selective reporting of experiments through predefined exclusion criteria. // Elife. 2020. Т. 9.

64. Nikitin S. V. и др. Breeding and selection of mini-pigs in the ICG SB RAS // Vestn. VOGiS. 2018. Т. 22. № 8. С. 922–930.

65. Nikitin S. V. и др. ‘Genetic load’ and changes in the chronology of early mortality in mini-pigs of ICG SB RAS // Agronomy Research. 2020. Т. 18. № 3. С. 2156–2165.

66. Peng R. D. R Programming for Data Science. , 2022. 147 с.

67. Pfeiffer A., Pia M. G. Data analysis with R in an experimental physics environment // 2013.

68. Pleil J. D. QQ-plots for assessing distributions of biomarker measurements and generating defensible summary statistics // Journal of Breath Research. 2016. Т. 10. № 3. С. 035001.

69. Ren L., Zhou S., Kästner C. Forks insight: providing an overview of GitHub forks // Proceedings of the 40th International Conference on Software Engineering: Companion Proceeedings ICSE ’18. New York, NY, USA: Association for Computing Machinery, 2018. С. 179–180.

70. Revelle W. psych (version 2.5.6). Procedures for Psychological, Psychometric, and Personality Research. , 2025.

71. Reynolds P. S. Time-series analyses of beaver body temperatures // Case studies in biometry. New York: John Wiley and Sons, 1994. С. 211–218.

72. Salgado C. M. и др. Missing Data. // Secondary Analysis of Electronic Health Records. Cham (CH): Springer, 2016. С. 143–162.

73. Salinas Ruíz J. и др. Generalized Linear Mixed Models with Applications in Agriculture and Biology. Cham: Springer International Publishing, 2023.

74. Scott D. W. Histogram // WIREs Computational Statistics. 2010. Т. 2. № 1. С. 44–48.

75. Sepulveda J. L. Using R and Bioconductor in Clinical Genomics and Transcriptomics // The Journal of Molecular Diagnostics. 2020. Т. 22. № 1. С. 3–20.

76. Shatokhin K. и др. Effect of KIT gene polymorphism on the reproductive ability of mini pigs // Int. J. Bio. Chem. Sci. 2025. Т. 19. № 1. С. 142–156.

77. Singh D., Garg R. R, Language for Data Analytics // SSRN Electronic Journal. , 2019. С. 757–762.

78. Snee R. D. Graphical Display of Two-way Contingency Tables // The American Statistician. 1974. Т. 28. № 1. С. 9–12.

79. Stanstrup J. и др. The metaRbolomics Toolbox in Bioconductor and beyond // Metabolites. 2019. Т. 9. № 10. С. 200.

80. Tierney N. Getting started with naniar // https://cran.r-project.org/web/packages/naniar/vignettes/getting-started-w-naniar.html. 2024.

81. Vanraden P. Selection of dairy cattle for lifetime profit // 2002.

82. Venables W. N., Ripley B. D. Modern Applied Statistics with S. New York, NY: Springer New York, 2002. 501 с.

83. Wickham H. The Split-Apply-Combine Strategy for Data Analysis // J. Stat. Soft. 2011. Т. 40. № 1.

84. Wickham H., Grolemund G. R for Data Science. Gravenstein Highway North, Sebastopol, CA: O’Reilly, 2017. 520 с.

85. Wiernik B. M., Dahlke J. A. Obtaining Unbiased Results in Meta-Analysis: The Importance of Correcting for Statistical Artifacts // Advances in Methods and Practices in Psychological Science. 2020. Т. 3. № 1. С. 94–123.

86. Zhao Y. Data Mining Applications with R | ScienceDirect. : Academic Press, 2013. 470 с.

87. Zou W. и др. Branch Use in Practice: A Large-Scale Empirical Study of 2,923 Projects on GitHub // 2019 IEEE 19th International Conference on Software Quality, Reliability and Security (QRS). , 2019. С. 306–317.

Приложение 1. Примерные задания для выполнения контрольных работ

1. Создайте вектор с заданным нормальным распределением. Параметры: n = 100, sd = 500, mean = 9764
2. Создайте матрицу 6:5, заполнив последовательностью чисел от 1 до 60 с интервалом 2.
3. Создайте матрицу 4:5, заполнив последовательностью из 20 чисел начиная с 0.23 с интервалом 0.17
4. Объедините матрицы из п. 2 и п.3 с увеличением числа строк
5. Из матрицы state.x77 извлеките строки, где Population превышает среднее значение.
6. Скачайте таблицы Alta202408 и Alta202412, создайте список быков, встречающихся в обоих каталогах
7. По данным таблицы WWS создайте столбец, содержащий информацию о стране происхождения быка
8. Постройте круговую диаграмму, отражающую долю стран происхождения быков (п. 7)
9. Найдите медиану с 1-го по 4-й столбец таблицы iris
10. Рассчитайте стандартное отклонение для каждой категории инсектицидов (InsectSprays)
11. Создайте вектор с заданными параметрами: n = 100, min = 500, max = 5000
12. Создайте матрицу 5:5, заполнив последовательностью чисел от 1 до 25 с интервалом 1
13. Создайте матрицу 5:5, заполнив последовательностью из 25 чисел начиная с 0.35 с интервалом 0.18
14. Объедините матрицы из п. 2 и п.3 с увеличением числа столбцов
15. Загрузите таблицу Crabs, извлеките строки, в которых цвет панциря = B.
16. По данным таблицы Crabs создайте ключевой столбец Key, содержащий записи о цвете панциря (sp) и гендерной принадлежности (sex) крабов.
17. Представьте данные длины панциря (CL) в виде широкой таблицы. В качестве заголовков используйте столбец Key
18. Рассчитайте медиану по каждому столбцу в получившейся таблице
19. Рассчитайте стандартное отклонение ширины панциря спереди (FL) в зависимости от категорий столбца Key
20. Изобразите ранжированный ряд ширины панциря сзади (RW) на гистограмме
21. Создайте вектор с заданными параметрами: n = 100, min = 500, max = 5000
22. Создайте матрицу 5:5, заполнив последовательностью чисел от 1 до 25 с интервалом 1

23. Создайте матрицу 5:5, заполнив последовательностью из 25 чисел начиная с 0.35 с интервалом 0.117
24. Объедините матрицы из п. 2 и п.3 с увеличением числа столбцов
25. Загрузите таблицу Crabs, извлеките строки, в которых цвет панциря = B.
26. По данным таблицы Crabs создайте ключевой столбец Key, содержащий записи о цвете панциря (sp) и гендерной принадлежности (sex) крабов.
27. Представьте данные длины панциря (CL) в виде широкой таблицы. В качестве заголовков используйте столбец Key
28. Рассчитайте медиану по каждому столбцу в получившейся таблице
29. Рассчитайте стандартное отклонение ширины панциря спереди (FL) в зависимости от категорий столбца Key
30. Изобразите ранжированный ряд ширины панциря сзади (RW) на гистограмме
31. Создайте вектор с заданными параметрами: начало = 1, интервал = 2.257, n = 80
32. Создайте матрицу 6:6, заполнив последовательностью чисел от 1 до 30 с интервалом 1.
33. Создайте матрицу 6:6, заполнив последовательностью из 30 чисел начиная с 0.38 с интервалом 0.13
34. Объедините матрицы из п. 2 и п.3 с увеличением числа столбцов
35. Из столбца weight (таблица ChickWeigh) извлеките записи в интервале от ±2 сигмы.
36. Из таблицы Semex_202212 Holstein_Genomax извлеките записи о быках, рождённых в Канаде
37. Загрузите таблицу cat_traits, отсортируйте по возрасту (Age) по возрастанию
38. Рассчитайте средний вес кошек (Cat$Weight) в зависимости от пола (*Gender)*
39. Рассчитайте стандартное отклонение TPI, Milk, % Fat, % Protein по данным таблицы Semex_202212 Holstein_Genomax
40. Изобразите на графике динамику Stature (о данным таблицы Semex_202212 Holstein_Genomax) в зависимости от года рождения быка (извлечь из столбца **Date of Birth**)

Приложение 2. Список используемых в учебном пособии данных, взятых из открытых ресурсов с указанием ссылок на оригинальные источники

Название файла	Описание	Источник
ABS_202212.csv*	Каталог быков-производителей компании ABS за декабрь 2022	https://absbullsearch.absglobal.com/BullList?VisibilityCountryCode=RUS&BreedTypeCode=D&BreedCode=HO&ProofCode=USA&comp=&o=NetMerit|true
alta_202408.xlsx	Каталог быков-производителей компании AltaGenetics за декабрь 2022	https://bullsearch.altagenetics.com/us/BS/List
Semex_202212.csv*	Каталог быков-производителей компании Semex за декабрь 2022	https://www.semex.com/us/i?lang=en&page=home
cat_traits.csv	Набор данных, полученный Эмирханом Булутом. Предоставляет собой таблицу с данными о различных характеристиках кошек в разных регионах	https://www.kaggle.com/datasets/emirhanai/global-street-cats-statistics-for-ai-research?resource=download
food_kaggle.csv	Набор данных о питательности пищевых продуктов, скачан из открытого доступа	https://www.kaggle.com/datasets/shrutisaxena/food-nutrition-dataset
ST.scv*	Каталог быков-производителей компании ST Genetics за декабрь 2022	https://www.stgen.com/sire-directory/dairy-proof.aspx?proof=usa&code=holstein&language=english&title=dairy-cattle
ВНИИ_ПЛЕМ_Дюрок_хряки.xlsx	База данных национального генофонда свиней породы дюрок за 2022 год	hhttps://vniiplem.ru/l/gisc/bd_ng/db-ng-svin/
allobgepa2408_ho.csv	База данных племенной ценности быков голштинской породы	https://lactanet.ca/genetique/evaluations-genetiques/
heroes_information	Данные о вымышленных супергероях	https://github.com/pbiecek/TechnikiWizualizacjiDanych2018/blob/master/Projekt1/WachulecPysiakMakowski/dataset/heroes_information.csv

* - каталоги быков обновляются трижды в год