Лабораторная работа №2.1

Способы подготовки и отображения данных в R

2.1. Цель работы:

Научиться подготавливать наборы данных для анализа. Усвоить способы загрузки данных. Составить представление о способах отображения.

2.2. Общие сведения

Чтобы получить представление о графических возможностях ***R***, наберите в командной строке ***demo(graphics).*** Некоторые из графиков, которые при этом появятся, представлены на рис. 2.1. Другие демонстрационные наборы графиков можно получить, напечатав ***demo(Hershey)***, ***demo(persp)*** и ***demo(image).*** Для того чтобы увидеть полный набор демонстрационных графиков, введите ***demo()*** без параметров.

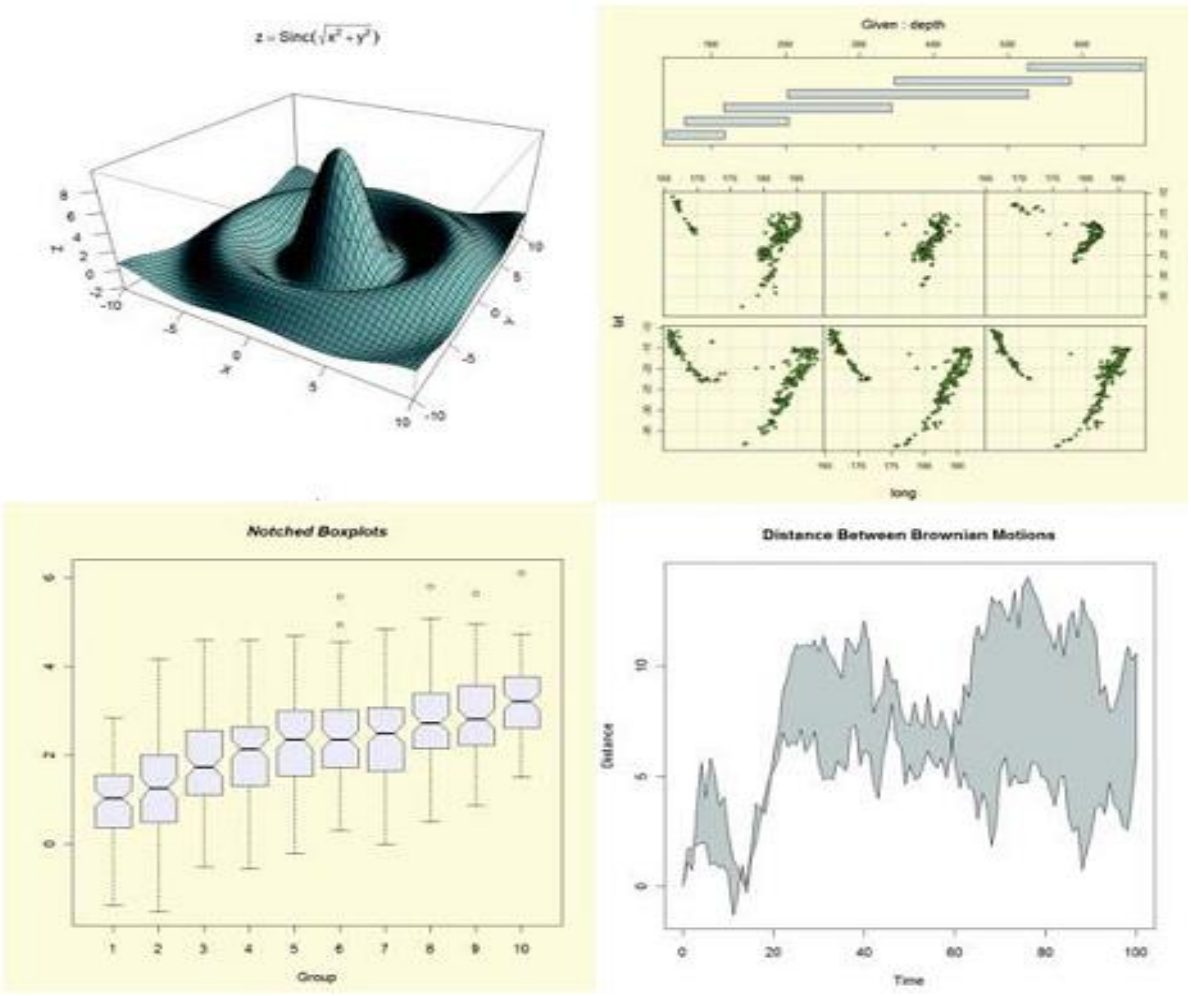


Рис.2.1. Некоторые графики, которые появляются
при вводе функции ***demo()***

Совершенно необходимо иметь под рукой справку, поэтому напоминаю необходимые для работы функции:

Таблица 2.1. Функции вызова справки в R

Функция	Действие
`help.start()`	Общая справка
`help ("нечто")` или `?нечто`	Справка по функции *нечто* (кавычки необязательны)
`help.search("нечто")` или `??нечто`	Поиск в справке записей, содержащих *нечто*
`example("нечто")`	Примеры использования функции *нечто* (кавычки необязательны)
`RSiteSearch("нечто")`	Поиск записей, содержащих *нечто* в онлайн-руководствах и заархивированных рассылках
`apropos("нечто", mode="function")`	Список всех доступных функций, в названии которых есть *нечто*
`data()`	Список всех демонстрационных данных, содержащихся в загруженных пакетах
`vignette()`	Список всех доступных руководств по загруженным пакетам
`vignette("нечто")`	Список руководств по теме *нечто*

Таблица 2.2. Функции, использующиеся для управления рабочим пространством в R

Функция	Действие
`getwd()`	Вывести на экран название текущей рабочей директории
`setwd("моя_директория")`	Назначить *моя_директория* текущей рабочей директорией
`ls()`	Вывести на экран список объектов в текущем рабочем пространстве
`rm("список_объектов")`	Удалить один или несколько объектов
`help(options)`	Справка о возможных опциях
`options()`	Посмотреть или установить текущие опции
`history(#)`	Вывести на экран последние # команд (по умолчанию 25)
`savehistory("мой_файл")`	Сохранить историю команд в файл *мой_файл* (по умолчанию `.Rhistory`)
`loadhistory("мой_файл")`	Загрузить историю команд (по умолчанию `.Rhistory`)
`save.image("мой_файл")`	Сохранить рабочее пространство в файл *мой_файл* (по умолчанию `.Rdata`)
`save("список_объектов", file="мой_файл")`	Сохранить определенные объекты в файл
`load("мой_файл")`	Загрузить сохраненное рабочее пространство в текущую сессию (по умолчанию `.Rdata`)
`q()`	Выйти из программы. Появится вопрос, нужно ли сохранить рабочее пространство

Ввод данных из файла

Функция ***source("filename")*** запускает скрипт. Если не прописан путь к файлу, подразумевается, что он находится в текущей рабочей директории. Например, команда source("myscript.R") запускает серию команд R, которые записаны в файле myscript.R. Принято, чтобы файлы скриптов имели расширение .R, но это не обязательное условие.

Текстовый вывод

Функция sink("***Имя файла***") выводит все результаты выполнения команд в файл с названием ***Имя файла***. По умолчанию, если этот файл уже существует, новая версия записывается **поверх старой**. Параметр ***append=TRUE*** позволяет добавлять новый текст в файл, а не записывать его вместо старого текста. Параметр ***split=TRUE*** позволяет выводить результаты и на экран, и в текстовый файл. Выполнение команды ***sink()*** без аргументов восстановит вывод результатов только на экран.

Графический вывод

Хотя команда ***sink()*** управляет выводом текста, она не оказывает никакого воздействия на вывод графики. Для управления выводом изображений используйте возможность экспорта рисунка:

Рис.2.2.а) Экспорт графиков в PDF

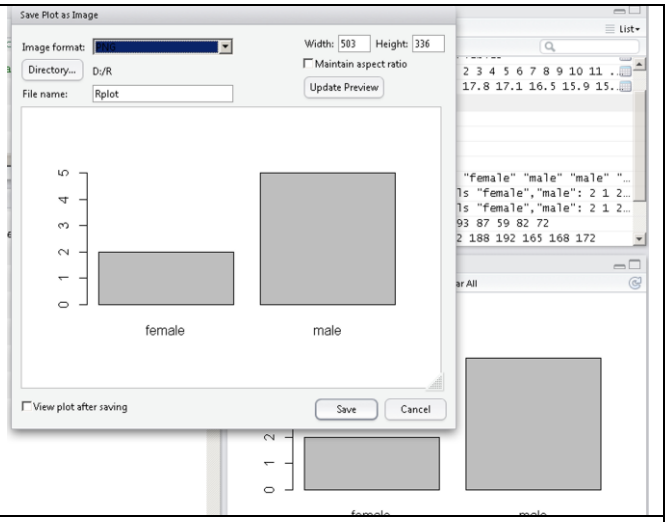


Рис.2.2.б) Экспорт графиков в выбранный графический формат.

2.3. Теперь пример:

1. Заходим **Tools-> Global Options**, устанавливаем **Default working directory**. (Я установила D:/R).
1. Перезапускаем RStudio, проверяем путь по умолчанию: getwd().

Записываем новый скрипт с именем " R_.R":

```
x<-c(174,162,188,192,165,168,172) #данные о росте семи сотрудников
небольшой компании
str(x) # Функция str() выводит данные об объектах
pol<-c("male","female","male","male","female","male","male")
#Формируем вектор "пол" для сотрудников фирмы
is.character(pol)
is.factor(pol)
is.vector(pol)
str(pol)
table(pol)
pol.f<-factor(pol)
is.factor(pol.f)
pol.f
```

```
plot(pol.f)
```

*) Кстати, функция ***ls.str()*** позволяет получить информацию о типах всех актуальных на данный момент объектов.

2. Затем загружаем и запускаем скрипт:

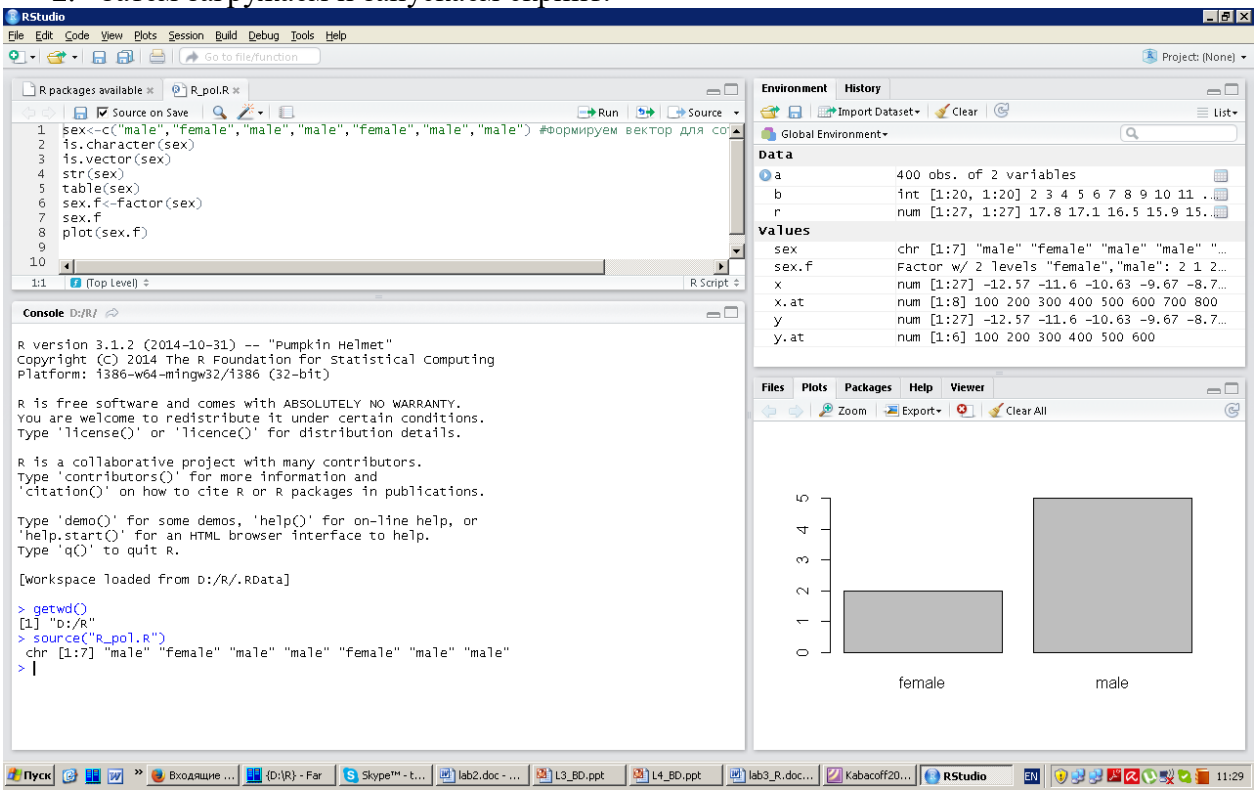
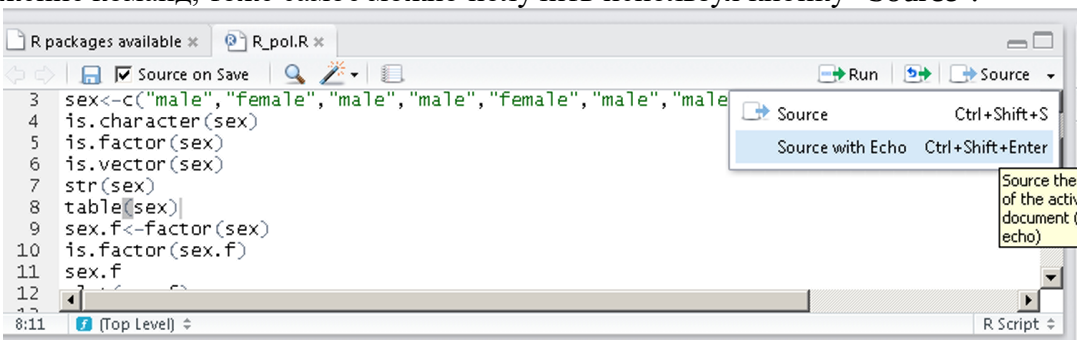


Рис.2.2. Написание скрипта, загрузка его на выполнение.

3. Если установим ***source('D:/R/R_.R', echo=TRUE)***, то получим пошаговое отображение команд, тоже самое можно получить используя кнопку "Source":



А использование кнопки "Run" позволяет выполнить скрипт пошагово.

4. Поясним фрагмент кода:

```
pol.f<-factor(pol)
is.factor(pol.f)
pol.f
plot(pol.f)
```

Команда plot(), увы, не умеет ничего хорошего сделать с таким вектором. И это, в общем-то, правильно, потому что программа ничего не знает про свойства пола человека. В таких случаях пользователь сам должен проинформировать R, что его надо рассматривать как категориальный тип данных. Делается это так:

К примеру, у нас есть вектор

```
pol<-c("male","female","male","male","female","male","male")
```

Команда `pol.f<-factor(pol)`

устанавливает внутреннее соответствие 1=Type1 и 2=Type2 (присвоение числовых значений происходит в алфавитном порядке). Любой анализ, который вы будете проводить

с вектором ***pol***, будет воспринимать эту переменную как номинальную, и выбирать статистические методы, подходящие для этого типа данных.

5. Поэтому теперь при попытке тестирования типа `pol.f` мы получим:

```
> is.factor(pol.f)
[1] TRUE
> is.character(pol.f)
[1] FALSE
> str(pol.f)
 Factor w/ 2 levels "female","male": 2 1 2 2 1 2 2
```

6. Кроме того, факторы в отличие от текстовых векторов можно легко преобразовать в числовые значения:

```
>as.numeric(pol.f)
[1]2122122
```

7. Зачем это нужно, становится понятным, если рассмотреть вот такой пример: положим, кроме роста, у нас есть ещё и данные по весу сотрудников, и мы хотим построить такой график, на котором были бы видны одновременно рост, вес и пол. Вот как это можно сделать:

```
>#Вектор веса
>w<-c(69,68,93,87,59,82,72)
>#Построение графика
>plot(x,w,pch=as.numeric(pol.f),col=as.numeric(pol.f))// x - возраст, см. выше
>legend("topleft",pch=1:2,col=1:2,legend=levels(pol.f))
```

Тут, разумеется, нужно кое-что объяснить. Параметры ***pch*** и ***col*** предназначены для определения соответственно типа значков и их цвета на графике. Таким образом, в зависимости от того, какому полу принадлежит данная точка, она будет изображена кружком или треугольником и чёрным или красным цветом, соответственно. При условии, разумеется, что все три вектора соответствуют друг другу. Ещё надо отметить, что изображение пола при помощи значка и цвета избыточно, для ¾ нормального графика хватит и одного из этих способов. Попробуйте, например, так:

```
plot(x,w,pch=(7:8), col=c("magenta","green"))
legend("topleft",pch=7:8, col=c("magenta","green"),legend=levels(pol.f))
```

Рис.2.4. Символы, назначаемые при помощи параметра pch

8. Факторы также можно упорядочивать, превращая их в некое подобие числовых данных. Введём четвёртую переменную: размер маек для тех же самых гипотетических восьмерых сотрудников:

```
> m<-c("L","S","XL","XXL","S","M","L")
> m.f<-factor(m)
> m.f
[1] L   S   XL  XXL S   M   L
Levels: L M S XL XXL
```

Как видно из примера, уровни расположены просто по алфавиту, а нам надо, чтобы

"S"(small) шёл первым. Кроме того, надо как-то сообщить R, что перед нами не просто категориальные, а упорядочиваемые категориальные данные. Делается это так:

```
m.o<-ordered(m.f,levels=c("S","M","L","XL","XXL"))
> m.o
[1] L   S   XL  XXL S   M   L
Levels: S < M < L < XL < XXL
```

2.4. Пропущенные данные

В дополнение к векторам из чисел и текстовым векторам. R поддерживает ещё и логические вектора, а также специальные тины данных, которые бывают очень важны для статистических расчётов. Прежде всего, это пропущенные или отсутствующие данные, которые обозначаются как `NA`. Такие данные очень часто возникают в реальных полевых и лабораторных исследованиях, опросах, тестированиях и т. д. При этом следует осознавать, что наличие пропущенных данных вовсе не означает, что данные в целом некачественны. С другой стороны, статистические программы должны как-то работать и с такими данными.

Пример: предположим, что у нас имеется результат опроса тех же самых семи сотрудников. Их спрашивали: сколько в среднем часов они спят, при этом один из опрашиваемых отвечать отказался, другой ответил «не знаю», а третьего в момент опроса просто не было в офисе. Так возникли пропущенные данные:

```
h <- c(8, 10, NA, NA, 8, NA, 8)
h
[1] 8 10 NA NA 8 NA 8
```

Из примера видно, что NA надо вводить без кавычек, а R нимало не смущается, что среди цифр находится «вроде бы» текст. Отметим, что пропущенные данные очень часто столь же разнородны, как и в нашем примере. Однако кодируются они одинаково.

Теперь о том, как надо работать с полученным вектором ***h***. Если мы просто попробуем посчитать среднее значение (функция ***mean ()***), то получим:

```
> mean(h) [1] NA
```

Чтобы высчитать среднее от ¾ непропущенной части вектора, можно поступить одним из двух способов:

```
>mean(h,na.rm=TRUE)
[1]8.5
>mean(na.omit(h))
[1]8.5
```

Часто возникает ещё одна проблема: как сделать подстановку пропущенных данных, скажем, заменить все NA на среднюю по выборке. Распространённое решение примерно следующее:

```
>h[is.na(h)]<-mean(h,na.rm=TRUE)
>h
[1]8.010.08.58.58.08.58.0
```

В левой части первого выражения осуществляется индексирование, то есть выбор нужных значений ***h*** таких, которые являются пропущенными (***is.na ()***). После того, как выражение выполнено, «старые» значения исчезают навсегда.

Чтобы очистить всю таблицу от строк с NA используйте один из вариантов:

```
df2<-df[rowSums(is.na(df)) == 0,] # убираем все NA в таблице
df3<-df %>% na.omit # нужна library(dplyr)
```

2.5. Таблицы данных

Наконец подошли к самому важному типу данных—к таблицам данных (data frames). Именно таблицы данных больше всего похожи на электронные таблицы Excel и аналогов, и поэтому с ними работают чаще всего. Особенно это касается начинающих пользователей R. Таблицы данных — это гибридный тип представления. *одномерный список из векторов*

одинаковой длины. Таким образом, каждая таблица данных — это список колонок, причём внутри одной колонки все данные должны быть одного типа. Проиллюстрируем это на примере созданных ранее (файл ***"R_.R"***) векторов:

```
names(w)<-c("Коля","Женя","Петя","Саша","Катя","Вася","Жора")
> d<-data.frame(weight=w,height=x,size=m.o,pol=pol.f)
> d
      weight height size    pol
Коля      69    174    L   male
Женя      68    162    S female
Петя      93    188   XL   male
Саша      87    192  XXL   male
Катя      59    165    S female
Вася      82    168    M   male
Жора      72    172    L   male

>str(d)
'data.frame':7 obs. of  4 variables:
 $ weight: num  69 68 93 87 59 82 72
 $ height: num  174 162 188 192 165 168 172
 $ size  : Ord.factor w/ 5 levels "S"<"M"<"L"<"XL"<..: 3 1 4 5 1 2 3
 $ pol   : Factor w/ 2 levels "female","male": 2 1 2 2 1 2 2
```

Поскольку таблица данных является списком, к ней применимы методы индексации списков. Кроме того, таблицы данных можно индексировать и как двумерные матрицы. Вот несколько примеров:

```
> d$weight
[1] 69 68 93 87 59 82 72
d[[1]]
[1] 69 68 93 87 59 82 72
>d[,1]
[1] 69 68 93 87 59 82 72
>d[,"weight"]
[1] 69 68 93 87 59 82 72
```

Как, например, отобрать из нашей таблицы только данные, относящиеся к женщинам? Вот один из способов:

```
d[d$pol=="female",]
      weight height size    pol
Женя      68    162    S female
Катя      59    165    S female
```

Чтобы отобрать нужные строки, поместим перед запятой логическое выражение, ***d$pol==female.*** Его значением является логический вектор:

```
> d$pol=="female"
[1] FALSE  TRUE FALSE FALSE  TRUE FALSE FALSE
```

Более сложным случаем селекции является сортировка таблиц данных. Для сортировки вектора достаточно применить команду ***sort()***, а вот если нужно, скажем, отсортировать наши данные сначала по полу, а потом по росту, приходится применить операцию посложнее:

```
d[order(d$pol,d$height),]
      weight height size    pol
Женя      68    162    S female
Катя      59    165    S female
Вася      82    168    M   male
Жора      72    172    L   male
Коля      69    174    L   male
Петя      93    188   XL   male
Саша      87    192  XXL   male
```

Команда ***order()*** создаёт не логический, а числовой вектор, который соответствует будущему порядку расположения строк. Подумайте, как применить команду ***order()*** для того чтобы отсортировать колонки по алфавиту.

2.6. Задания к лабораторной работе

Выполнить мини соцопрос в своей группе по темам по шкале от 0 до 10 или дробная шкала 0-1, для этого сделать таблицу в Google Docs:

1. Любимый фильм;
2. Любимая книга;
3. -//- вид спорта;
4. -//- времяпрепровождение;
5. -//- компьютерный бренд;
6. -//- бренд мобильных устройств;
7. -//- футбольная команда;
8. -//- литературный жанр;
9. -//- предмет в ВУЗе;
10. -//- компьютерная игра;
11. Наиболее часто посещаемые страницы интернет.
12. Любимый Браузер (поисковик).
13. Предложить свое исследование.

Таблица заполняется степенью предпочтения каждого студента из вашей группы, например:

Компьютерный бренд:

		ASUS	ACER	SAMSUNG	…
1	Иванов	0.8	0.7	0.9	
2	Петров	1	0.5	0.8	
…					
20					

Должно быть не менее 10 колонок. У каждого участника опроса может быть 1-2 предпочтения. Допускается до 10 ячеек с пропущенными данными (NA).

По результатам необходимо

- вычислить max, min, mean по каждому столбцу;
- подсчитать количество людей, отдавших предпочтение выбранному элементу >0.7 и <0.3 (составить вектор);
- вывести рейтинг фильмов (книг...или что у вас там) в списке по убыванию;
- построить столбчатую диаграмму оценок (нужно сделать разными способами);
- **оформить отчет (отчет по ЛР 1-2 необходимо предоставить в случае, если вы отстаете от графика сдачи Лабораторных работ)**.