Wprowadzenie do R

v 1.1

Artur Suchwałko, Agnieszka Suchwałko, Adam Zagdański

QuantUp.pl

10.03.2011-11.07.2012

Spis treści

1 Wstęp

Książeczka ta uzupełnienia prowadzone przez nas szkolenia z dziedziny budowy i walidacji modeli scoringowych i predykcyjnych, statystyki, praktycznej analizy danych i data miningu.

Naszym celem podczas niektórych szkoleń nie jest uczenie samego R. R pomaga nam w uczeniu metod analizy danych. Pomijamy więc szczegóły techniczne i dużo informacji dodatkowych o R. Ograniczając te informacje do niezbędnego minimum podczas szkoleń udostępniamy gotowe funkcje i skrypty. Pozwala to nawet zapomnieć, że pracujemy w potężnym narzędziu, jakim jest R. Zalecamy zapoznanie się z tymi materiałami przed szkoleniami oraz samodzielne przećwiczenie podanych przykładów. Podczas innych szkoleń uczymy jednak, jak analizować dane w R.

Zdecydowaliśmy, że chcemy udostępnić te materiały wszystkim zainteresowanym, nie tylko uczestnikom szkoleń.

System R jest oparty na języku programowania. Oznacza to, że bardziej zaawansowani użytkownicy mogą nieograniczenie rozwijać możliwości systemu tworząc własne metody upraszczające wykonywane przez nich analizy danych. Te możliwości nie będą jednak utrudniać pracy początkującym użytkownikom. Ta grupa użytkowników może korzystać z systemu po prostu wywołując odpowiednie funkcje.

Samodzielne nauczenie się R wymaga czasu. Problemami nie należy się przejmować, a najlepsza jest metoda prób i błędów. Biegłość przyjdzie z czasem, a możliwości języka R mogą spowodować, że system R stanie się podstawowym narzędziem do codziennej pracy z danymi.

R zdobywa coraz większe uznanie w bankowości i innych instytucjach finansowych. Nawet jeśli podstawowe narzędzie jest inne, R bardzo często stosowany jest jako uzupełnienie i pomaga w wykonywaniu niestandardowych analiz.

Chcesz wiedzieć, dlaczego warto nauczyć się R i z niego korzystać? Przeczytaj ten krótki tekst: „Dlaczego R?".

W kolejnych rozdziałach opisane są:

- *Wprowadzenie* – pożyteczne informacje na temat systemu R,
- *Pierwszy kontakt z R* – korzystanie z R bez programowania i pierwsze kroki,
- *Podstawy R* – przegląd obiektów w R,
- *Dane w R* – praca z danymi,
- *Odczyt i zapis* – komunikacja R ze światem zewnętrznym,
- *Grafika w R* – tworzenie grafiki,

– *Elementy programowania w R* – podstawy programowania,

– *Elementy statystyki* – statystyka opisowa, wykresy statystyczne,

– *Losowanie, zmienne losowe i symulacje* – jak generować liczby pseudolosowe,

– *Przykładowe sesje z R* – kilka gotowych przykładów do samodzielnego prześledzenia.

Jeśli nie masz czasu, to przeczytaj rozdziały: *Wprowadzenie*, *Pierwszy kontakt z R*, *Dane w R* (tylko pobieżnie) i *Przykładowe sesje z R*.

Jeśli chcesz poznać możliwości R, to w Internecie znajdziesz naprawdę dużo informacji. Liczba książek dotyczących systemu R systematycznie rośnie. Kilka wymieniamy w spisie literatury. Wyczerpujące informacje o systemie R można znaleźć na stronie domowej projektu: http://www.r-project.org. W dziale „Documentation" znajduje się wiele materiałów mogących pomóc w samodzielnej nauce R.

Czego nie ma w tym dokumencie, a czego warto się nauczyć:

– operacje na napisach (łańcuchach znakowych),

– operacje macierzowe i algebra macierzowa,

– wektory indeksów elementów,

– więcej o programowaniu,

– formuły, NaN i NA,

– zmienne typu czynnikowego (factor),

– bardziej zaawansowane operacje na ramkach danych,

– bardziej zaawansowana grafika, dostosowywanie wyglądu wykresów: parametry graficzne,

– tworzenie pakietów.

Jeśli chcesz zaoszczędzić czas, łatwo nauczyć się więcej niż z tej książeczki (na przykład z wymienionych wyżej tematów), przećwiczyć wszystko na komputerze z pomocą doświadczonych użytkowników i programistów R oraz otrzymać solidne materiały i skrypty, to zapraszamy na nasze szkolenia z R. Oferujemy też szkolenia z analizy danych, statystyki i podobnych dziedzin.

Jesteśmy fanami (i fankami) R. Wykorzystujemy go od lat w zastosowaniach biznesowych, naukowych i dydaktycznych. Jesteśmy przekonani, że polubicie system R, a z jego pomocą analiza danych oraz modelowanie okażą się przyjemniejsze niż dotychczas.

Ostrzeżenie
Czytasz właśnie bardzo wczesną i wciąż roboczą wersję tej książeczki (v 1.1). Uznaliśmy jednak, że mimo to może Ci się ona przydać. Przepraszamy za niedoróbki i planujemy aktualizację.

Artur, Agnieszka i Adam

2 Wprowadzenie

Ten rozdział przedstawia specyfikę i wyjątkowość systemu R. Opisane są w nim skrótowo możliwości i dziedziny zastosowania systemu R. Bardzo niecierpliwy czytelnik może pominąć większą część tego rozdziału. Tym czytelnikom zalecamy jednak przeczytanie przynajmniej opisu instalacji. Po zainstalowaniu R mogą oni przejść do przykładów umieszczonych w ostatnim rozdziale.

2.1 Na początek...

Omawiane w tym dokumencie zagadnienia stanowią tylko przegląd użytecznych lub po prostu potrzebnych zastosowań pakietu R przez osoby, które nie są biegłymi programistami. Każdy z zamieszczonych tu przykładów można (czytaj: „należy") wykonać samodzielnie. Oczywiście, trzeba zwrócić przy tym uwagę, że część poleceń nie ma sensu (czyli może nie działać), jeśli nie zostaną wykonane polecenia poprzednie.

2.2 Właściwości R

Ten zagadkowo zatytułowany rozdział przedstawia właściwości systemu R takie jak: integralny język programowania, obszary, w których R jest stosowany, opisuje też możliwości systemu.

Historia R i S-PLUS

R bazuje na języku S. Pierwsza dyskusja z cyklu spotkań, które doprowadziły do powstania specyfikacji języka S odbyła się w maju 1976 roku w grupie pięciu osób (wśród nich był John Chambers). Były problemy z wyborem nazwy. Często w akronimach powtarzało się „S", więc taką nazwę wybrano. Język S jest zaimplementowany w S-PLUS i R. S-PLUS jest komercyjną implementacją języka S (http://www.tibco.com/). Język S miał być opracowany wyłącznie dla potrzeb różnorodnych analiz wykonywanych przez grupę statystyków z Bell Labs, ważna była zatem elastyczność i możliwość programowania.
Ross Ihaka i Robert Gentleman (nazywani „R & R") napisali pierwszą wersję R w roku 1995 (Auckland, Nowa Zelandia). Zamierzali wykorzystywać ten program do celów edukacyjnych. Nazwa „R" pochodzi od ich inicjałów i nawiązuje do „S".
Od 1997 istnieje R Core Team (17 osób). Zespół ten kieruje rozwojem R oraz bierze aktywny udział w rozwijaniu systemu. W grupie tej znajduje się również John Chambers, twórca S. Udział w tworzeniu R bierze wiele osób, a pakiety / biblioteki tworzą użytkownicy systemu oraz naukowcy z całego świata.

Czym jest R?

R jest środowiskiem, w którym są zaimplementowane metody statystyczne oraz metody analizy i wizualizacji danych. Właśnie tak charakteryzują R jego twórcy. Określenie R mianem środowiska, w którym są zaimplementowane pewne metody podkreśla uniwersalność systemu (inaczej R byłby opisywany na przykład jako środowisko analiz statystycznych).
W ponad tysiącu pakietów dodatkowych udostępniona jest większość metod klasycznej i współczesnej statystyki. System R działa praktycznie na wszystkich platformach, a użytkownicy mogą wybierać między kilkoma opcjonalnymi interfejsami użytkownika. System R oraz prawie wszystkie jego pakiety dodatkowe są darmowe do wszelkich zastosowań (licencja GNU GPL). Szczególnie chętnie jest używany przez naukowców język R jest standardem dla współczesnej statystyki. Autorzy artykułów metodologicznych często uzupełniają je o biblioteki stworzone dla środowiska R.
Liczba użytkowników szacowana nawet na kilkaset tysięcy. Popularność systemu R w Polsce wciąż rośnie i jest on coraz częściej wykorzystywany w zastosowaniach komercyjnych.
Niezmiernie ważny jest fakt, że R jest oparty na języku programowania. Oznacza to ogromną elastyczność systemu. Żeby korzystać z R nie trzeba jednak ani być programistą, ani nawet umieć programować.

Co potrafi R?

R jest bardzo wszechstronnym systemem i długo można opowiadać o jego możliwościach. W związku z tym w naszym krótkim opracowaniu wyliczymy jedynie jego kluczowe zalety:

- obsługa danych praktycznie dowolnego rodzaju (np. MySQL, Oracle, ODBC, XML, SAS transport file),
- ogromna liczba gotowych do użycia narzędzi statystyki i analizy danych,
- ogromne możliwości graficzne (grafika w pełni modyfikowalna),
- grafika o jakości prezentacyjnej (do raportów, prezentacji i publikacji),
- operacje macierzowe (wraz z macierzami rzadkimi),
- prosty i efektywny język programowania R (w tym elementy programowania obiektowego),
- możliwość integracji praktycznie ze wszystkimi językami programowania i z wieloma innymi narzędziami.

Kto wykorzystuje R?

System R jest wykorzystywany przede wszystkim w środowisku naukowym, biznesie i w nauczaniu. Stopniowo język R stał się uniwersalnym językiem współczesnej statystyki.
W zastosowaniach biznesowych szczególnie popularny stał się wśród ludzi zajmujących się modelowaniem statystycznym i inżynierią finansową. System R ujawnia także ogromne zalety, gdy wykorzystuje się go jako pomoc w nauczaniu statystyki i analizy danych. Wymaga zrozumienia wykonywanej analizy, a z drugiej strony nie jest ogromnym systemem, którego nauka wymaga wielu lat pracy. Kolejny ogromny walor dydaktyczny to dostępność kodów źródłowych wszystkich metod zaimplementowanych w R. Istnieją podręczniki do statystyki oparte wyłącznie na R.

Dziedziny zastosowania R

System R może być stosowany wszędzie tam, gdzie pojawia się potrzeba analizy danych, budowy modeli statystycznych i stochastycznych, czy graficznej prezentacji informacji. Wymieńmy niektóre z obszarów zastosowań:

- ekonometria, finanse,
- nauki społeczne,
- analiza danych ekologicznych, ankietowych, przestrzennych,
- analiza szeregów czasowych,
- testowanie hipotez (również testy dokładne),
- modelowanie stochastyczne,
- wnioskowanie bayesowskie,
- modele regresyjne, modele mieszane,
- uczenie maszyn (w tym sieci neuronowe), data mining,
- statystyka wielowymiarowa (analiza skupień, klasyfikacja).

Popatrz, które pakiety wykorzystać do jakich analiz: CRAN Task Views.

Specyfika i wyjątkowość R

Przyjrzyjmy się temu, co jest specyficzne dla systemu R i co sprawiaja, że jest on wyjątkowym narzędziem.

- R jest darmowy do wszelkich zastosowań (licencja GPL)
- język R:
  - w przeciwieństwie do R, systemy statystyczne dostarczają często zamknięty zbiór metod, co ogranicza ich zastosowania,
  - dzięki językowi R możliwa jest dowolna rozbudowa środowiska.
- R działa w interaktywnym trybie komend:
  - narzuca użytkownikowi dyscyplinę w sposobie wykonywania analiz,
  - sprzyja wykonywaniu analiz ze zrozumieniem,
  - procentuje wykształceniem dobrych nawyków w analizie danych,
  - trudniej niż w innych narzędziach jest zrobić w R coś, czego się nie rozumie,
  - jeśli zrobiło się w R coś, czego się nie rozumie, łatwiej to później zrozumieć niż gdybyśmy pracowali w innych narzędziach,
  - powszechne używanie skryptów,
  - uzyskiwanie powtarzalnych rezultatów dzięki skryptom (tzw. reproducible research),
  - można operować na wynikach działania metod; to jest pozornie techniczna sprawa, ale okazuje się ważna podczas tworzenia większych programów do analizy danych,

– stroma krzywa uczenia:

  - oznacza to, że konieczny jest większy nakład pracy (inwestycja) na początku niż w przypadku narzędzi „okienkowych",
  - zyskujemy jednak lepsze i głębsze zrozumienie wykonywanych analiz oraz nieograniczone możliwości uczenia się.

– podstawowa różnica między R i innymi systemami:

  - R przechowuje wyniki w obiektach, na których można wykonywać dowolne dalsze operacje,
  - wyjście na ekran jest ograniczone i R nie zasypuje nas wynikami, jeśli tego nie chcemy.

Platformy i graficzne interfejsy użytkownika

System R jest dostępny na praktycznie wszystkie platformy (możliwe jest nawet uruchomienie R na PocketPC). Należą do nich oczywiście: Windows, Linux, MacOS. Na platformie linuksowej system R działa zauważalnie szybciej.
System R posiada graficzny interfejs użytkownika o dosyć ograniczonych możliwościach (tylko na platformie Windows). Dla większości użytkowników nie jest to przeszkodą (po czasie potrzebnym na przyzwyczajenie się). Pozostali użytkownicy mogą skorzystać z wielu dostępnych „nakładek graficznych" na R:

– RStudio (http://www.rstudio.com/ide/download/),

– RCommander (http://socserv.mcmaster.ca/jfox/Misc/Rcmdr/),

– Jaguar (http://rosuda.org/JGR/),

– TINN-R (http://www.sciviews.org/Tinn-R/),

– Emacs (Emacs Speaks Statistics) (http://ess.r-project.org/).

Programowanie w R

R bazuje na kompletnym, wygodnym i efektywnym języku programowania. Praktycznie wszystkie pakiety dostępne są w postaci kodów źródłowych. Istnieje sporo książek poświęconych programowaniu w języku S i R.
Nie będziemy poświęcać programowaniu w R dużej uwagi, ale możliwości dla programistów R są wręcz nieograniczone.
Możliwa jest też integracja z C, C++, czy Fortranem. Kody w C mogą korzystać bezpośrednio z obiektów R. Dzięki serwerowi i klientowi (D)COM (platforma Windows) możliwa jest integracja R ze wszystkimi językami programowania oraz na przykład z oprogramowaniem Microsoft Office.

2.3 Instalacja R

Instrukcja instalacji napisana została wyłącznie dla użytkowników systemu Windows, użytkownicy Linux'a sami sobie poradzą, a użytkownicy MacOS przywykli, że się ich zazwyczaj pomija. Wymagania sprzętowe i programowe dla R są naprawdę minimalne, w zasadzie działa na każdym komputerze:

– Windows 95 lub nowszy,

– obsługa długich nazw plików,

– przynajmniej 18 MB miejsca na dysku.

Oto krótki opis sposobu instalacji systemu R dla Windows:

– Pobieramy plik instalacyjny: http://www.r-project.org → „CRAN" → Wybór mirrora → „Download R for Windows" → „base" → „Download R 3.0.3 for Windows"

– Instalacja: uruchomienie ściągniętego pliku

– Zalecamy zmianę podczas instalacji języka komunikatów R na angielski. Dzięki temu będzie można dużo łatwiej znaleźć pomoc, jeśli pojawią się komunikaty o błędach.

– Uruchomienie: menu Windows „Start"

– Instalacja dodatkowych pakietów: menu „Packages" → „Install package(s)..." (można też instalować pakiety z plików lokalnych)

– Aktualizacja pakietów: menu „Packages" → „Update packages..."

– Podstawowe komendy: q(), help(), help.start()

Można zainstalować R na CD lub pamięci USB (PenDrive) po prostu przenosząc instalację z dysku twardego. System R nie zapisuje nic w rejestrach systemowych, instalację można przenosić więc bez ograniczeń.

2.4 Dodatkowe pakiety do R

Po zainstalowaniu R mamy do dyspozycji już spore możliwości, ponieważ dostarczany jest on z kilkudziesięcioma podstawowymi pakietami. To jest dopiero początek. O prawdziwej potędze R decydują dodatkowe pakiety, które można w miarę potrzeb zainstalować. Ile jest pakietów, tego dokładnie nikt nie wie, ale wiadomo, że liczba oficjalnie dystrybuowanych bibliotek przekracza 4.000 (maj 2012). Poza oficjalną dystrybucją istnieje wiele specjalizowanych pakietów, które nie uzyskały tak dużej popularności.
Instalacja pakietów jest łatwa. Aby zainstalować dodatkowy pakiet należy wybrać z menu **Packages** opcję **Install package(s)...**. Przy instalacji R zapyta z jakiego serwera chcemy skorzystac (można wybrać dowolny, ale najlepiej zdecydować się na położony w Polsce lub niedaleko), a nastepnie wyswietli liste pakietów gotowych do zainstalowania. Wystarczy wybrać pakiety i kliknąć **OK**. Automatycznie zostaną zainstalowane pakiety, które są niezbędne do poprawnego działania wybranego przez nas pakietu, tzw. dependencies czyli „zależności".
Bardziej zaawansowani użytkownicy lub osoby, które nie korzystają z GUI (na przykład pracujący pod Linuxem) mogą posłużyć się następującą komendą do uzyskania identycznego efektu:

```
install.packages(nazwa.pakietu, dependencies=T)
```

Jeśli chcemy korzystać z pakietu, to po zainstalowaniu trzeba go załadować. Do tego celu służy polecenie:

```
library(nazwa.pakietu)
```

Nazwa polecenia jest nieco myląca, ale właściwie każdy pakiet jest również w sensie programistycznym biblioteką, więc trzeba po prostu do tego przywyknąć.

3 Pierwszy kontakt z R

Ten rozdział przedstawia, jak korzystać z systemu R bez programowania. Nawet najbardziej niecierpliwy czytelnik powinien zapoznać się z tym rozdziałem.

3.1 R nie jest trudny

Mając kontakt z R po raz pierwszy można pomyśleć, że jest on programem niezbyt przyjaznym i bardzo trudnym do nauczenia się. Opanowanie podstaw R nie jest jednak trudne. Właśnie o tym chcielibyśmy przekonać czytelników w tym rozdziale.
Oczywiście, aby zostać ekspertem systemu R potrzeba dużo praktyki i sporo czasu spędzonego przed komputerem.

Przykład na początek

Zaczniemy od klasycznego przykładu: budowa modelu regresji liniowej. W wielu pakietach statystycznych model regresyjny buduje się kilkoma lub kilkunastoma kliknięciami. W R wymaganych jest zaledwie kilka prostych poleceń.

```
# wczytanie biblioteki
library(MASS)

# wczytanie danych
data(iris)

# budowa modelu regresyjnego
model.reg <- lm(Petal.Length~Sepal.Length, data=iris)

# tak wyglada model
summary(model.reg)

# zestaw wykresow diagnostycznych
plot(model.reg)
```

Zwróćmy uwagę na rozbudowany zestaw wykresów diagnostycznych
Siła systemu R objawi się podczas wykonywania skomplikowanych analiz oraz podczas ich automatyzacji. Konieczność wykonania skomplikowanej obróbki wstępnej danych ujawni możliwości systemu R.

Ilustracja możliwości graficznych

Grafice w R poświęcona jest świetna książka P. Murrela [6] oraz interesująca strona internetowa [7]. Jednak najprostszym sposobem na zapoznanie się z możliwościami systemu R jest przyjrzenie się tworzonym przez niego wykresom. Do tego wystarczy jedna instrukcja:

```
demo(graphics) # polecenie uruchamia demonstracje mozliwosci graficznych
```

3.2 Pomoc w R

Jest bardzo wiele sposobów uzyskiwania pomocy dotyczącej systemu R. Przedstawiamy skrótowo niektóre z nich. Pomoc na temat konkretnej funkcji:

– help(„nazwa”)

– help(nazwa)

– ?nazwa

Pomoc w formacie HTML:

– help.start()

Zawansowane wyszukiwanie (można korzystać z wyszukiwania przybliżonego i z wyrażeń regularnych)

– help.search()

Przykładowe zastosowanie funkcji

– example(nazwa)

Pakiet, w którym dostępna jest dana funkcja

– find(nazwa)

Z pewnością na szczególną uwagę zasługuje system pomocy w HTML uruchamiany instrukcją „help.start()” i pojawiający się po chwili w przeglądarce internetowej.
Wyszukiwanie informacji na temat R w Internecie może sprawiać trudność, ponieważ nazwa systemu jest jednoliterowa. Rozwiązaniem jest wykorzystywanie dedykowanej R wyszukiwarki RSeek (http://www.rseek.org).

3.3 Warto zapamiętać

Poniżej wymieniamy ważne informacje praktyczne: ułatwienia w korzystaniu z R, wybrane ważne polecenia R oraz opisujemy wszystkie opcje menu programu.

Ułatwienia w korzystaniu z R

– strzałka w górę – przywoływanie wykonanych wcześniej komend,

– CTRL + L – czyszczenie ekranu,

– ustawianie katalogu roboczego – polecenie `setwd()` lub „Change dir...” z menu „File”,

– można zapisać historię pracy (menu „File”, opcja „Save history”),

– można zapisać tak zwaną przestrzeń roboczą (workspace); dzięki temu możemy przerwać analizę i nic nie tracąc powrócić do niej w innym momencie (menu „File”, opcja „Save workspace”),

– praca ze skryptami, uruchamianie skryptów.

Ważne polecenia R

Poniżej wymieniamy naszym zdaniem najważniejsze polecenia systemu R.

`help()` – wywołuje pomoc,

`q()` – zamykamy R,

`library(nazwa)` – wczytanie pakietu o wskazanej nazwie,

`data(nazwa)` – wczytanie zbioru danych,

`search()` – ścieżka poszukiwań R (również lista załadowanych pakietów),

`ls(), ls(2)` – dostępne obiekty ,

`getwd(), setwd()` – ustawienie i sprawdzenie katalogu roboczego,

`source("skrypt.R")` – wczytanie kodu R z pliku,

`history()` – przejrzenie historii wykonanych poleceń.

Pożyteczne funkcje dostępne z menu

Poniżej prezentujemy przegląd możliwości graficznego menu systemu R dostępnego dla platformy Windows. Nie jest ono zbyt rozbudowane, ale funkcje są przydatne i warto poznać wszystkie możliwości dostępne z menu.

Zakładka	**Opcja**	**Działanie**
File	Source R code	wczytanie kodu źródłowego
	Load workspace / Save workspace	odczyt / zapis przestrzeni roboczej
	Load history / Save history	odczyt / zapis historii
	Change dir	zmiana katalogu roboczego
	Exit	wyjście z R
Edit	GUI preferences	preferencje
Misc	Remove all objects	usunięcie wszystkich obiektów
Packages	Load package	wczytywanie pakietów
	Install package(s)	instalowanie pakietów z repozytorium
	Update packages	automatyczna aktualizacja pakietów
	Install packages from local zip files	instalacja pakietów z lokalnych plików zip
Help	FAQ on R	najczęściej zadawane pytania
	FAQ on R for Windows	najczęściej zadawane pytania (R dla Windows)
	Manuals (in PDF)	podręczniki
	Html help	pomoc w formacie html
	Search help	przeszukiwanie pomocy

3.4 Pierwszy kontakt

Proponujemy zobaczyć, jaki będzie rezultat wprowadzenia do R poniżej wypisanych komend. Ilustrują one najbardziej podstawowe właściwości R: obliczenia, zmienne oraz pracę z obiektami w pamięci R.

Podstawowe polecenia

`2 + 2`	suma
`exp(2)`	e^2
`rnorm(10)`	próba dziesięcioelementowa z rozkładu normalnego
`x <-2`	przypisanie, równoważne zapisowi $x = 2$
`x + 3`	5, ale wynik nie zostanie nigdzie zapisany
`napis <-"to jest napis"`	łańcuch znaków
`objects()`	wypisuje obiekty przechowywane aktualnie w pamięci
`ls()`	to samo co objects()
`rm(x)`	usunięcie obiektu „x”

4 Podstawy R

Korzystanie w zakresie podstawowym z systemu R nie wymaga dużej znajomości języka R. Są jednak pewne ważne elementy, o których warto wspomnieć. Należą do nich operacje na wektorach, macierzach i indeksach. Posiadanie takiej wiedzy ułatwi zrozumienie działania systemu oraz dalszą naukę. W tym rozdziale przedstawiamy podstawowe zasady pracy z R, trochę wskazówek praktycznych oraz elementarne informacje o wykorzystywanych przez R obiektach.
Ponad 90% czynności wykonywanych podczas analizy danych przez początkującego użytkownika R będzie sprowadzało się do operacji na ramkach danych (data.frame). Dlatego właśnie te obiekty uznajemy za najważniejsze.
Niecierpliwy czytelnik może pominąć ten rozdział i przejść do rozdziału dotyczącego ramek danych. Serdecznie zachęcamy jednak do zapoznania się z tym krótkim rozdziałem w całości. Ułatwi to dalszą naukę i zapewni zrozumienie tego, co się robi.

4.1 Proste operacje na danych

Na początek podajemy kilka najbardziej podstawowych operacji na ramkach danych. Dla lepszego zrozumienia rozdziału poświęconego danym zachęcamy do przeczytania fragmentów poświęconych obiektom i podstawom pracy z R. Bez znajomości tych kwestii trudno będzie wykonywać bardziej zaawansowane operacje na danych.

```
# wczytujemy przykładowe dane
library(MASS)
data(Cars93)

edit(Cars93) # oglądamy dane

dim(Cars93) # wielkosc zbioru danych

names(dane) # lista zmiennych

Cars93$Price # dostep do zmiennej

Cars93[17,5] # dostep do pojedynczych elementow

Cars93[,5] # dostep do kolumn danych

Cars93[17,] # dostep do wierszy danych
```

4.2 Obiekty w R

W R wszystko jest obiektem. Każdy obiekt ma swoje właściwości. Obiekty w pakiecie R to:

– tablice liczb (liczba jest tablicą jednoelementową),

– tablice znaków,

– funkcje,

– bardziej złożone struktury danych, na przykład modele statystyczne albo dane, które analizujemy (te struktury zbudowane są z powyższych elementów).

Przestrzeń robocza (workspace), to zbiór aktualnie przechowywanych obiektów. Dodatkowa zaletą jest możliwość zapisania przestrzeni roboczej, dzięki czemu łatwo będzie można wrócić do przerwanej analizy.
R odróżnia duże i małe litery.
Można nazwać zmienne używając kropki wewnątrz nazwy: „taka.nazwa.zmiennej".

Funkcje obsługujące cechy obiektów

`class()`	–	klasa (rodzaj) obiektu, dodatkowy atrybut uzupełniający „mode",
`length()`	–	długość obiektu,
`attributes()`	–	atrybuty obiektu,
`attr()`	–	dostęp do atrybutów obiektu,
`object.size()`	–	wielkość obiektu w bajtach,
`mode()`	–	wewnętrzna reprezentacja obiektu,
`str()`	–	szczegóły wewnętrznej reprezentacji obiektu.

4.3 Wywoływanie funkcji

W dużej mierze analiza danych będzie sprowadzała się do wywoływania gotowych funkcji. W systemie R wywołania funkcji są specyficzne: pozwalają nazywać argumenty, zmieniać ich kolejność i korzystać z parametrów domyślnych. Przedstawimy te własności na podstawie funkcji „seq", która tworzy wektor składający się z równoodległych liczb.
Kolejnym ważną cechą jest to, że funkcje w R są przeciążone (polimorficzne). Oznacza to, że pewne funkcje (na przykład `print`, `summary`, czy `plot`) mogą być wywoływane dla różnych obiektów: ramek danych, wektorów, tablic kontyngencji, a nawet zbudowanych modeli statystycznych. Dzięki temu zapamiętanie podstawowych funkcji jest łatwiejsze.

Przykład

`?seq`	–	pomoc na temat funkcji seq
`seq(1,5)`	–	liczby od 1 do 5 (krok równy 1)
`seq(1,5,by=0.5)`	–	liczby od 1 do 5 z krokiem 0.5
`seq(by=0.5,to=5,from=1)`	–	wynik identyczny jak poprzednio, ale inna kolejność parametrów wejściowych funkcji
`seq()`	–	wywołanie funkcji z parametrami domyślnymi

Ciekawostka: wywołanie: `args(seq.default)` zwraca listę argumentów przyjmowanych przez funkcję „`seq.default`". `args()` jest również funkcją.

4.4 Zmienne i operatory logiczne

`TRUE (T)`	–	wartość liczbowa == 1
`FALSE (F)`	–	wartość liczbowa == 0
`NA` („not available”)	–	kod brakującej obserwacji – ważne w statystyce
`NaN` („Not a Number”)	–	wyrażenie nie jest wartością liczbową np. dzielenie przez zero
`x<y`	–	prawda jeśli x mniejsze od y
`x<=y`	–	prawda jeśli x mniejsze lub równe y
`x>y`	–	prawda jeśli x większe od y
`x>=y`	–	prawda jeśli x większe lub równe y
`x==y`	–	prawda jeśli x równe y
`x!=y`	–	prawda jeśli x różne od y
`&`	–	logiczny „and”
`|`	–	logiczny „or”
`!A`	–	negacja wyrażenia logicznego A

Przykład

```
x <- TRUE   # zmienna x = 1
y <- x & T  # zmienna y = 1
z <- x & F  # zmienna z = 0
```

4.5 Wektory i operacje na wektorach

Wektory służą do zapisywania liczb, ale mogą także przechowywać inne, dowolne, obiekty. W wektorze może być na przykład zapisany wiek pewnej liczby osób. Możliwe są praktycznie wszystkie operacje arytmetyczne oraz logiczne na wektorach. Oto przykłady:

`w <-c(1,2,3)`	–	w jest wektorem o elementach 1, 2 i 3
`c(1,2,3)-> w`	–	to samo
`w <-1:3`	–	to samo
`w = 1:3`	–	to samo
`length(w)`	–	zwraca ilość elementów wektora „w”
`sin(w)`	–	liczy sinus dla wszystkich elementów wektora
`min(w)`	–	znajduje najmniejszy element wektora w i zwraca go
`max(w)`	–	znajduje największy element wektora w i zwraca go
`sum(w)`	–	sumuje wszystkie elementy wektora w i zwraca wynik
`prod(w)`	–	mnoży wszystkie elementy wektora w i zwraca wynik
`w.l <-c(T,T,F)`	–	wektor„w.l” składa się z elementów 1, 1, 0
`w.t <-c("jeden","dwa","trzy")`	–	wektor „w.t” to wektor zawierający łańcuchy znakowe
`sort(w.t)`	–	ponieważ sortujemy łańcuchy, więc porządek jest leksykograficzny (czyli alfabetyczny)
`length(w.t)`	–	wektor ma trzy elementy
`w.t[2]`	–	tak dostajemy się do drugiego elementu wektora „w.t”; należy pamiętać, że R numeruje od 1!!!
`w.t[c(3,2)]`	–	zwrócone zostaną elementy wektora „w.t” znajdujące się na pozycjach będących składowymi wektora z nawiasów kwadratowych, czyli: „trzy”, „dwa”
`zakupy<-c(ser = 3, woda = 5, chleb = 1)`	–	utworzony został wektor z nazwanymi elementami, takie wektory lub listy często są zwracane jako wyniki działania wbudowanych funkcji R.

Operacje na wektorach wartości logicznych

Wektory wartości logicznych będą przydatne podczas wybierania podzbiorów ze zbiorów danych. To będzie nasze podstawowe zastosowanie dla tego rodzaju obiektów. Wydawać by się mogło, że to sprawa bardzo techniczna, ale warto przećwiczyć i zrozumieć, jak pracować na wektorach wartości logicznych.

Funkcja	**Działanie**
`TRUE, FALSE`	wartości logiczne
`&`	operator „and”
`|`	operator „or”
`==`	porównanie
`any`	czy przynajmniej jedna wartość wektora to `TRUE`
`all`	czy wszystkie wartości wektora to `TRUE`
`which`	indeksy elementów, w których jest `TRUE`

Jest duża różnica między operatorami logicznymi `&` i `&&` dla wektorów wartości logicznych. Rezultatem działania pierwszego z nich jest wektor, a operacje wykonywane są dla kolejnych

elementów przekazanych wektorów i umieszczane w kolejnych elementach wektora wynikowego. Drugi z operatorów w wyniku daje tylko jedną wartość logiczną. Działanie innych operatorów logicznych jest analogiczne. Najlepiej zrozumieć różnicę analizując poniższy przykład.

```
> a <- c(T,T,F); b <- c(T,F,T)
> a&b
[1] TRUE FALSE FALSE # wynikiem jest caly wektor
> a&&b
[1] TRUE # w wyniku dostajemy tylko jedna wartosc logiczna
>
```

Podsumowanie operacji na wektorach

Funkcja	Działanie
`vector`	tworzenie wektora
`c`	tworzenie wektora z podanych elementów
`:, seq`	tworzenie ciągów
`rep`	powielanie wektora
`length`	długość wektora
`[ ]`	pobieranie i zmiana elementów wektora
`+,-,*,/,<, <=, sin`	operacje na wektorach
`sort`	sortowanie wektora (także z indeksem)
`rev`	zmiana kolejności elementów wektora
`sum, cumsum`	suma elementów wektora
`prod, cumprod`	iloczyn elementów wektora
`max, min`	największy i najmniejszy element wektora
`which.max`	indeks największego elementu wektora

Ważny przykład – wektoryzacja operacji

To jest naprawdę ważny przykład. Większość operacji na wektorach albo macierzach może być wykonywana tak, jak na liczbach. Możemy na przykład dodawać albo odejmować całe wektory. To jest właśnie tytułowa „wektoryzacja operacji". Zalecamy taki sposób pracy na wektorach.
Podstawowe korzyści są dwie: zapis jest bardziej czytelny oraz tak napisane polecenia wykonywane są szybciej niż jakby zapisać je przy wykorzystaniu pętli.
Uwaga dla czytelników mających jakiekolwiek doświadczenie programistyczne: R jest językiem interpretowanym, a nie kompilowanym, wobec tego pętle wykonywane są krok po kroku, a operacje na całych wektorach wykonywane są bezpośrednio przez kod w C, w którym napisany jest R.
W poniższym przykładzie wyznaczymy dla ilustracji operacji wektorowych indeks BMI (Body-Mass Index) dla fikcyjnych danych dotyczących kilku osób. Później wyznaczymy dla wagi jej średnią i odchylenie standardowe z pomocą elementarnych operacji na wektorach. Wyniki porównamy z rezultatami uzyskanymi z pomocą wbudowanych funkcji „mean" i „sd".

`waga <-c(60,72,57,90,95)`	– tworzony jest wektor wartości liczbowych
`waga`	– wypisanie zawartości wektora na ekran
`wzrost <-c(1.72,1.80,1.65,1.90,1.74)`	– kolejny wektor wartości
`wzrost`	– wypisanie zawartości wektora
`bmi <-waga /wzrost^2`	– deklaracja wektora „bmi" i przypisanie do niego wartości będących wynikiem operacji na dwóch przed chwilą utworzonych wektorach
`bmi`	– wypisanie zawartości na ekran
`sum(waga)`	– suma elementów wektora „waga"
`sum(waga)/length(waga)`	– suma elementów wektora „waga" podzielona przez liczbę elementów tego wektora
`sr.waga <-sum(waga)/length(waga)`	– jak wyżej, ale przypisanie wyliczonej wartości do zmiennnej „sr.waga" (śradnia waga)
`waga - sr.waga`	– wektor odchyleń od średniej dla wektora „waga"
`(waga - sr.waga)^2`	– wektor kwadratów odchyleń dla wektora „waga"
`sum((waga - sr.waga)^2)`	– suma kwadratów odchyleń
`sqrt(sum((waga - sr.waga)^2)/(length(waga)- 1))`	– odchylenie standardowe wektora „waga"
`mean(waga)`	– średnia wektora „waga"
`sd(waga)`	– odchylenie standardowe (identyczna wartość otrzymana została dwie linie wyżej)
`summary(waga)`	– podsumowanie dla wektora „waga" (minimum, maksimum, średnia, kwantyle)
`bmi > 25`	– wynikiem działania polecenia jest wektor o długości takiej, jak wektor „bmi" i wartościach logicznych będących wynikiem działania warunku na poszczególne elementy wektora „bmi"

4.6 Macierze

Macierze wykorzystywane są w bardzo wielu dziedzinach matematyki (przykład: odwzorowania liniowe) i statystyki (przykłady: macierz kowariancji, macierz eksperymentu, wspolczynniki macierzowe modeli statystycznych). Trudno w związku z tym wyobrazić sobie analizę danych bez operacji macierzowych.
Oczywiście, typowy początkujący użytkownik z takich operacji korzystał nie będzie bezpośrednio. Warto jednak wiedzieć, że ramki danych (o których więcej powiemy później) często zachowują się tak, jak macierze.

Poniżej przedstawiamy podstawowe operacje macierzowe.

```
a <-matrix(1,4,3)
```
– macierz 4×3 wypełniona jedynkami

```
a[1,2]
```
– pobieranie elementu z pierwszego wiersza i drugiej kolumny

```
a[,2]
```
– druga kolumna macierzy

```
a[1,]
```
– piewszy wiersz macierzy

```
a %*%t(a)
```
– mnożenie macierzowe macierzy „a" i jej macierzy transponowanej

```
a.1<-solve(a)
```
– odwrócenie macierzy „a" i zapisanie macierzy odwrotnej w zmiennej „a.1"

```
a *a.1
```
– mnożenie macierzy element po elemencie

```
diag(a)
```
– przekątna macierzy „a"

```
rbind(c(1,2),c(3,4))
```
– tworzenie macierzy przez podanie jej kolejnych wierszy jako wektorów

```
cbind(c(1,2),c(3,4))
```
– tworzenie macierzy przez podanie jej kolejnych kolumn jako wektorów,

```
m <-matrix(1:9,3)
```
– tworzona jest macierz o trzech wierszach, wypełniona elementami od 1 do 9,

```
dimnames(m)<-list(c("x1","x2","x3"),c("y1","y2","y3"))
```
– nadane zostaną nazwy: wierszom „x1", „x2" , „x3", a kolumnom „y1", „y2", „y3",

```
m["x2","y3"]
```
– teraz w inny sposób można się dostać do potrzebnych elementów

Macierze – podsumowanie

Funkcja	Działanie
`matrix`	tworzenie macierzy
`rbind`	łączenie macierzy wierszami
`cbind`	łączenie macierzy kolumnami
`dim`	wymiar macierzy
`dimnames`	nazwy wymiarów macierzy
`t`	transpozycja
`[ ]`	pobieranie i zmiana elementów macierzy
`+,-,*,/,<,<=, sin`	operacje na macierzach
`sum`	suma elementów macierzy
`colSums, rowSums`	sumy elementów w kolumnach i wierszach
`%*%, solve`	mnożenie macierzowe i odwracanie macierzy
`diag`	przekątna lub tworzenie macierzy przekątniowej
`apply`	automatyzacja operacji na macierzy

4.7 Listy

Lista jest obiektem zawierającym obiekty dowolnych typów. Elementy listy mogą posiadać nazwy. Listy często wykorzystywane są przez funkcje do zwracania wyników. Ramka danych (ważny typ danych) też jest listą!

Przykład

```
student <-list(imie="Jan", wiek=22, zonaty=F)
```
– zmienna „student” jest obiektem typu lista

```
student$imie
```
– zwróci „Jan”; do składowych zmiennej student odwołujemy się przy pomocy znaku $

```
student$wiek
```
– analogicznie

```
student$zonaty
```
– analogicznie

```
student[[1]]
```
– działa identycznie jak `student$imie`

```
lista <-c(list(1,2,3),list(4,5,6))
```
– przykład łączenia dwóch list

```
lapply(lista,mean)
```
– na każdy element listy „lista” zostanie nałożona funkcja mean

Listy – podsumowanie

Funkcja	Działanie
`list`	tworzenie listy
`c`	tworzenie listy z podanych elementów / łączenie list
`length`	długość listy
`[ ], [[ ]]`	dostęp do elementów
`lapply`	automatyzacja operacji na listach

4.8 Skrypty

Analiza danych w R w dużej mierze polega na programowaniu. Nie jest to programowanie zaawansowane, ale przy okazji każdej analizy użytkownik R pisze sekwencję instrukcji, które analizę wykonują.
Zapisanie tych instrukcji do pliku tekstowego umożliwi powtórzenie analizy albo ułatwi wykonanie tej samej analizy na innych danych.
Instrukcje z takiego pliku tekstowego można wklejać do R za pośrednictwem schowka.
Druga opcja to wykorzystanie polecenia „Source" w menu „File" systemu R.
Kolejna opcja to wykorzystanie instrukcji `source(nazwa.pliku)`
Każdy użytkownik systemu R – prędzej czy później – poczuje potrzebę tworzenia własnych skryptów wykonujących czy automatyzujących analizy danych. Można pisać je w dowolnym edytorze tekstowym, system R posiada własny, prosty edytor. My polecamy jednak Notepad++ (http://http://notepad-plus-plus.org/) Crimson Editor (http://www.crimsoneditor.com) albo Tinn-R (http://www.sciviews.org/Tinn-R/) dla OS Windows. Wszystkie te edytory wspomniane potrafią podświetlać składnię języka R, co zdecydowanie upraszcza pracę. Tinn-R dodatkowo potrafi wysyłać kod do R, posiada kilka innych naprawdę interesujących możliwości. Wszystkie narzędzia są darmowe.
Użytkownicy innych sytemów raczej nie będą mieli trudności z odnalezieniem odpowiedniego dla siebie edytora, dla Linuxa polecamy jednak edytor Kate (http://kate-editor.org/).
Programiści mogą korzystać z dodatku StatET do Eclipse (http://www.walware.de/goto/statet).

5 Dane w R

5.1 Dane w R

Wczytywanie danych do R w praktycznie dowolnej postaci, tekstowej czy binarnej, nie stanowi problemu. Stosunkowo łatwo jest wczytać nawet bardzo specyficzne dane. Obsługa danych w R jest wyjątkowo prosta i dodatkowo R radzi sobie z brakami danych. Szczególnie przydatną i wartą zapamiętania jest funkcja `read.table`. Rozpoczniemy jednak od pracy z danymi wbudowanymi oraz przedstawienia najprostszych operacji na tych danych.

Przykład

`data(iris)`	–	ten zestaw danych jest wbudowany w R; polecenie ładuje zestaw danych do pamięci i pozwala z tych danych korzystać
`dane <-iris`	–	w ten sposób zestaw danych „iris" został przypisany do zmiennej „dane"
`head(dane)`	–	wyświetla pięć pierwszych elementów zbioru „dane"
`summary(dane)`	–	wyświetla minimum, maksimum, średnią i kwantyle dla każdej kolumny
`row.names(dane)`	–	wyświetla nazwy wierszy, w tym przypadku są to numery
`dim(dane)`	–	zwraca wymiar, w tym przypadku 150×5
`class(zmienna)`	–	klasa, jaką reprezentuje obiekt; ważne w programowaniu obiektowym

5.2 Ramki danych

Ramka danych (data.frame) jest podstawowym obiektem, z którego będziemy korzystać pracując z R. Dla wielu użytkowników może to być jedyny rodzaj obiektów, który będą wykorzystywać. Ramka danych składa się ze zmiennych (kolumny) i przypadków (wiersze). Proponujemy najpierw wczytać przykładowe dane wbudowane:

```
library(MASS)
data(Cars93)
```

oraz obejrzeć dane w następujący sposób:

```
edit(Cars93)
```

Podstawowe operacje, czyli wybór zmiennych, wybór przypadków oraz poszczególnych elementów zostaną opisane w tym rozdziale.

Ważne: ramka danych zachowuje się jak macierz.

Ramki danych (`data.frame`) są wykorzystywane do reprezentacji typowych tabel (arkuszy) zawierających dane. Ramka danych jest tablicą dwuwymiarową, w której dane w określonej kolumnie są tego samego typu, ale różne kolumny mogą zawierać dane różnych typów. Ramki

danych obsługują różne typy zmiennych: ciągłe, dyskretne czy znakowe. Bardzo ułatwiają pracę na danych umożliwiając wygodne wykonywanie wielu operacji. Technicznie mówiąc, ramka danych to lista z nadaną klasą `data.frame`.
Tworzenie ramek w środowisku R może odbywać się na różne sposoby:

– z linii poleceń R (czyli z pomocą języka R),

– interaktywnie, z pomocą wbudowanego edytora (funkcje `edit()`, `fix()`),

– tworząc ramki danych z innych obiektów, na przykład konwertując macierze,

– wczytywanie z różnego rodzaju plików.

Przykład

Pokażemy, jak tworzyć proste dane z wykorzystaniem linii poleceń R. Wyobraźmy sobie małe zoo, które zapiszemy w R jako ramkę danych w następujący sposób:

```
zwierzeta <- data.frame(
 gatunek =c("bobr", "morswin", "swistak", "leszcz", "kielb", "turkuc podjadek"),
 gromada =c("ssaki", "ssaki", "ssaki", "ryby", "ryby", "owady"),
 srodowisko=c("ladowe", "wodne", "ladowe", "wodne", "wodne", "ladowe"),
 wielkosc =c("sredni", "duzy", "sredni", "sredni", "maly", "maly"),
 liczba =c( 2, 4, 1, 3, 4, 10))
```

Zwróćmy uwagę na typy zmiennych:

gatunek zmienna znakowa; każde z naszych zwierząt należy do innego gatunku; można traktować ją jako identyfikator zwierzęcia,

gromada zmienna kategoryczna; każdy gatunek należy do pewnej gromady; gromad jest mniej niż gatunków,

srodowisko podobnie jak *gromada*,

wielkosc zmienna kategoryczna; podobnie jak *gromada* i *srodowisko*; różnica jest taka, że wielkości można ze sobą porównywać (mimo, że opisane są słowami), dlatego definiujemy zmienną jako kategoryczną uporządkowaną,

liczba zwykła zmienna numeryczna (ale przyjmuje tylko wartości naturalne).

Nadajemy zmiennym typy. To jest operacja raczej techniczna, ale ważna jest nawet sama świadomość, że zmienne w ramce danych mają typy i że można je zmieniać:

```
zwierzeta$gatunek <- as.character(zwierzeta$gatunek)
zwierzeta$gromada <- factor(zwierzeta$gromada)
zwierzeta$srodowisko <- factor(zwierzeta$srodowisko)
zwierzeta$wielkosc <- ordered(zwierzeta$wielkosc,levels=c("maly","sredni","duzy"))
```

Liczba zwierząt jest zmienną numeryczną, sprawdźmy to:

```
class(zwierzeta$liczba)
```

Uwaga: podobne modyfikacje typów cech na ogół trzeba wykonywać po wczytaniu danych z plików tekstowych. Podczas analizy musimy wiedzieć, jakiego typu są poszczególne zmienne. Brak znajomości typu zmiennej może doprowadzić do niezrozumiałych błędów.

Zmienne typu factor

Aby łatwiej zrozumieć czym są zmienne typu `factor` (zmienne kategoryczne) prześledźmy taką sesję:

```
> zwierzeta$wielkosc
[1] sredni duzy sredni sredni maly maly
Levels: maly < sredni < duzy
>
> levels(zwierzeta$wielkosc)
[1] "maly" "sredni" "duzy"
>
> zwierzeta$srodowisko
[1] ladowe wodne ladowe wodne wodne ladowe
Levels: ladowe wodne
```

Teraz już łatwo zrozumieć, że `wielkosc` jest zmienną kategoryczną. Oznacza to, że przyjmuje tylko pewną liczbę z góry określonych poziomów. Poziomy te zostały wypisane przy pomocy polecenia
`levels(zwierzeta$wielkosc)`. Możliwe poziomy mogą być uporządkowane, mogą być też nieuporządkowane, jak w przypadku zmiennej `srodowisko`.

Dostęp do elementów ramki danych

Dostęp do elementów ramki danych można uzyskać tak samo, jak do elementów macierzy, czyli w następujący sposób:

`zwierzeta$gatunek`	–	wypisze zawartość zmiennej
`zwierzeta$gatunek[3]`	–	wypisany zostanie trzeci element zmiennej
`zwierzeta[1,"gatunek"]`	–	dostęp do elementu na przecięciu pierwszego wiersza i kolumny „gatunek”
`zwierzeta[-(1:4),]`	–	otrzymamy „zwierzeta” bez pierwszych czterech wierszy i ze wszystkimi kolumnami
`zwierzeta$liczba[1] <-3`	–	element `zwierzeta$liczba[1]` zmienił wartość z 2 na 3
`zwierzeta$wielkosc <-NULL`	–	usuwanie zmiennej

Warto też pamiętać o funkcji `table()`. Używa się jej tak:

```
> table(zwierzeta$gromada)

owady ryby ssaki
   1 2 3

>table(zwierzeta$gromada,zwierzeta$srodowisko)

       ladowe wodne
  owady 1 0
  ryby 0 2
  ssaki 2 1
```

Bezpośredni dostęp do zmiennych

Do zmiennych można uzyskać bezpośredni dostęp. Bezpośredni dostęp przyspiesza pracę z poziomu linii komend i skraca zapis. Skrócenie zapisu można uzyskać też z wykorzystaniem funkcji `with()` (jak w języku Pascal!):

```
attach(zwierzeta)
gatunek[3]  # teraz nie jest juz potrzebny zapis zwierzeta$gatunek[3]
detach(zwierzeta)
```

Przykład

Jakich zwierząt jest najwięcej?

```
> zwierzeta[which.max(zwierzeta$liczba),]
         gatunek gromada srodowisko wielkosc liczba
6 turkuc podjadek owady ladowe maly 10
```

Przykład – wybieramy podzbiór danych

```
> subset(zwierzeta,gromada=="ssaki")
  gatunek gromada srodowisko wielkosc liczba
1 bobr ssaki ladowe sredni 3
2 morswin ssaki wodne duzy 4
3 swistak ssaki ladowe sredni 1
```

Operacje na ramkach danych – podsumowanie

Funkcja	Działanie
`data.frame`	tworzenie ramki danych
`[ ]`	dostęp do elementów
`$`	dostęp do zmiennych
`dim`	wymiar danych
`attach, detach`	dostęp do zmiennych jak do niezależnych obiektów
`head, tail`	początek i koniec danych
`names`	nazwy zmiennych
`row.names`	nazwy przypadków
`subset`	wybór podzbioru
`split`	podział na ramki danych według zmiennej kategorycznej
`levels`	poziomy zmiennej `factor`
`table`	tablica kontyngencji

5.3 Sprawdzenie i konwersja typu

Typ zmiennej czy obiektu wydawać się to może nieistotny, ale niedopasowanie typu albo zmiana (inaczej: konwersja) typu dokonana automatycznie przez R może spowodować błędy w działaniu funkcji R.
Konwersje są przydatne przy pisaniu własnych funkcji i sprawdzaniu typu argumentów. Dzięki świadomym konwersjom można pracować dużo bardziej efektywnie. Czasami są konieczne po to,

żeby dopasować typ obiektu do funkcji, z której chcemy skorzystać.
Prześledźmy następujący przykład konwersji:

```
> d <- factor(c("A", "B", "C", "C")) # tworzymy zmienna kategoryczna
> d
[1] A B C C
Levels: A B C
> is.factor(d) # czy to zmienna typu factor?
[1] TRUE
> is.numeric(d) # czy to zmienna numeryczna?
[1] FALSE
> as.numeric(d) # zrobmy konwersje na zmienna numeryczna
[1] 1 2 3 3
> # dwie kolejne konwersje: najpierw na zmienna numeryczna, pozniej na factor
> as.factor(as.numeric(d))
[1] 1 2 3 3
Levels: 1 2 3
> class(d) # jakiej klasy to jest obiekt?
[1] "factor"
>
```

Sprawdzenie typu	**Konwersja**
`is.numeric()`	`as.numeric()`
`is.integer()`	`as.integer()`
`is.double()`	`as.double()`
`is.complex()`	`as.complex()`
`is.logical()`	`as.logical()`
`is.character()`	`as.character()`
`is.factor()`	`as.factor()`
`is.na()`	–

6 Odczyt i zapis

W tym rozdziale opisujemy krótko sposoby komunikacji systemu R ze światem zewnętrznym. Znajduje się w nim opis komunikacji z użytkownikiem, opis zapisu i odczytu danych w formatach tekstowych oraz wskazówki dotyczące importowania danych z formatów różnych systemów statystycznych.
Początkujący użytkownik będzie prawdopodobnie wykorzystywał tylko odczyt i zapis plików w formacie CSV (Comma Separated Values) oraz odczyt i zapis plików binarnych R.

6.1 Klawiatura i ekran

Klawiatura i ekran to podstawowy sposób komunikacji z użytkownikiem. Na ogół nie będziemy pobierać parametrów od użytkownika, a raczej będziemy samodzielnie wpisywać parametry do odpowiednich obiektów. Z pewnością będziemy jednak często informować użytkownika o wynikach analizy, czyli wyświetlać informacje na ekranie.

Przykład – wejście

```
dane <-scan()                                          – enter kończy wczytywanie
dane <-scan(what="",sep="\n")                          – wczytywanie tekstów
dane <-scan(what = list(flag = "", x = 0, y = 0))      – wczytywanie rekordów; pierwsze
                                                         pole tekstowe, pozostałe
                                                         numeryczne
```

Przykład – wyjście

```
> (x<-1:15) # wziecie w nawiasy powoduje wyswietlenie wyniku
 [1] 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
>
> # mozna w ten sposob wyswietlac praktycznie dowolnie skomplikowane obiekty
> print(x)
 [1] 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
>
> cat(x,"\n")
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
> cat("Napis\n")
Napis
> cat("Napis\n",file="output.txt")
> cat("Wynik: suma =",round(pi,3),"$\n")
Wynik: suma = 3.142
```

6.2 Przykładowe dane tekstowe

Zaczniemy od prezentacji przykładowych danych tekstowych. Najczęściej właśnie z takich danych będziemy korzystać w analizach.
Oczywiście, R może pobierać dane bezpośrednio na przykład z bazy Oracle, ale praktycznie z każdego systemu baz danych można wyeksportować dane w formacie tekstowym. W związku z tym komunikacja z R wykorzystująca pliki tekstowe jest najbardziej uniwersalnym rozwiązaniem.

```
"NumPreg" "Glucose" "BloodPress" "Insulin" "BMI" "DiabPedigree" "Age" "Diagnosis"
"1" 6 148 72 0 33.6 0.627 50 "Chory"
"2" 1 85 66 0 26.6 0.351 31 "Zdrowy"
"3" 8 183 64 0 23.3 0.672 32 "Chory"
"4" 1 89 66 94 28.1 0.167 21 "Zdrowy"
"5" 0 137 40 168 43.1 2.288 33 "Chory"
"6" 5 116 74 0 25.6 0.201 30 "Zdrowy"
"7" 3 78 50 88 31.0 0.248 26 "Chory"
```

Pliki mają pewne bardzo ważne własności, które można kontrolować z pomocą parametrów funkcji wczytujących. Należą do nich:

- precyzja,
- posiadania nagłówka lub jego brak,
- separator pól,
- separator dziesiętny,
- kodowanie znaków,
- kodowanie brakujących wartości,
- cudzysłowy (do łańcuchów tekstowych),
- definiowanie rodzajów zmiennych

6.3 Dane z plików tekstowych

Stwierdziliśmy już, że komunikacja przez pliki tekstowe jest najbardziej uniwersalnym sposobem dostarczania danych dla R. W tym rozdziale przedstawimy, jak wczytywać i zapisywać dane w formatach tekstowych.

Odczyt – funkcja `read.table()`

Podstawową funkcją wczytującą pliki tekstowe jest `read.table()`. Wywołanie tej funkcji ma następującą postać:

```
read.table(file, header = FALSE, sep = "", quote = "\"'",
          dec = ".", row.names, col.names, as.is = FALSE,
          na.strings = "NA", colClasses = NA, nrows = -1,
          skip = 0, check.names = TRUE, fill = !blank.lines.skip,
          strip.white = FALSE, blank.lines.skip = TRUE,
          comment.char = "#", allowEscapes = FALSE, flush = FALSE)
```

gdzie poszczególne parametry oznaczają:

Parametr	Znaczenie
`file`	plik wejściowy
`header`	czy plik ma nagłówek?
`sep`	separator pól
`dec`	separator dziesiętny
`row.names`	nazwy przypadków
`col.names`	nazwy zmiennych
`na.strings`	kodowanie brakujących wartości
`colClasses`	klasy zmiennych dla kolumn

Może się to wydawać przytłaczające, ale mnogość parametrów świadczy o elastyczności funkcji. Wielu z tych parametrów podawać nie musimy, a zazwyczaj wystarczy takie wywołanie:

```
dane <- read.table("dane.txt", header=FALSE, sep=";",dec=".", na.strings = "brak")
```

Możliwe jest także wywołanie z jedynym argumentem:

```
dane <- read.table("dane.txt")
```

Ciekawostkę stanowi fakt, że w razie potrzeby można wczytać dane wprost z adresu URL.

Zapis – funkcja `write.table()`

Zapis danych odbywa się równie łatwo dzięki funkcji `write.table()` o składni bardzo zbliżonej do funkcji `write.table()`. Najprostsze wywołanie tej funkcji to:

```
write.table(dane, file="dane.txt")
```

Wywołanie z dodatkowymi parametrami jest niewiele bardziej złożone:

```
write.table(dane, file="dane.txt", append=TRUE,
                sep=";", row.names=TRUE, col.names=FALSE)
```

6.4 Pliki binarne w R

System R posiada własny standard zapisywania i odczytywania plików binarnych. Korzystanie z plików binarnych R jest wygodne, ponieważ pliki binarne są mniejsze od tekstowych.
Czasami po wczytaniu pliku tekstowego do R trzeba nadać zmiennych odpowiednie typy. Zazwyczaj R prawidłowo odróżnia zmienne kategoryczne od numerycznych, ale nie jest w stanie zgadnąć, czy zmienna kategoryczna ma mieć wartości uporządkowane, czy nie. Wtedy zmiennym nadajemy typ „ręcznie". Po zapisaniu takich danych w formacie tekstowym typ zmiennej nie będzie zapamiętany. Rozwiązaniem tego problemu jest zapisanie danych w postaci binarnej. Dodatkową zaletą jest to, że do plików binarnych możemy zapisywać wszelkie obiekty, nie tylko ramki danych.

Odczyt – funkcja `load()`

```
load("dane.RData")
```

Wynikiem działania funkcji `load()` jest wczytanie obiektów do przestrzeni roboczej. Nazwy tych obiektów będą takie same, jak nazwy, pod jakimi obiekty zostały wcześniej zapisane.

Zapis – funkcja `save()`

```
save(x,y,z,"dane.RData")
```

Skrócona wersja save, zapisuje do pliku całą przestrzeń roboczą R (czyli wszystkie obiekty w pamięci systemu R):

```
save.image()
```

Uwaga: kolejne wersje R zapisują w inny sposób pliki binarne, kompatybilność wstecz nie jest więc pełna.

Dane z innych systemów statystycznych

Pakiet `foreign` umożliwia komunikację z innymi systemami statystycznymi. Oznacza to możliwość wczytywania, oraz zapisywania danych w specyficznych dla poszczególnych systemów statystycznych formatach. W tabeli poniżej zebrane są częściej przydatne:

Funkcja	**System**
`read.dbf(...)`	format dbf
`read.spss(...)`	SPSS
`library(Hmisc); spss.get(...)`	rozszerzony import SPSS
`read.octave(...)`	Octave
`read.xport(...)`	SAS
`library(R.matlab)`	współpraca z Matlabem

Wbudowane dane

Podstawowe sposoby wczytywania wbudowanych danych, są jak się można było spodziewać jeszcze prostsze. Aby dowiedzieć się jakie dane są dostępne wystarczy posłużyć się poleceniem:

```
data()
```

Dane ze wszystkich zainstalowanych pakietów pojawią się na ekranie jeśli wpiszemy komendę:

```
data(package=.packages(all.available=TRUE))
```

Jeśli wiemy, jakie dane chcemy wczytać:

```
data(Cars93)
```

Uwaga: pakiet `datasets` zawiera dużo interesujących danych.

Wejście i wyjście – podsumowanie

Funkcja	**Działanie**
`scan`	odczyt z ekranu (lub pliku)
`(...)`	wyświetlanie wyniku działania instrukcji
`cat, print`	wyjście na ekran
`format`	formatowanie wyniku do wyjścia na ekran
`sink`	przekierowanie wyjścia do pliku
`pdf, postscript, ...`	urządzenia graficzne
`dev.off`	wyłączenie urządzenia graficznego
`read.table, read.csv`	wczytywanie danych tekstowych
`read.fwf`	wczytanie danych o stałej szerokości pól
`write.table`	zapis danych tekstowych
`write.matrix, write, dump`	inne funkcje do zapisu tekstowego
`load, save, save.image`	odczyt i zapis binarny obiektów R
`data`	dostęp do danych wbudowanych

7 Grafika w R

R ma ogromne możliwości graficzne. Oprócz setek gotowych do użycia wykresów możliwa jest ich nieograniczona modyfikacja, a także tworzenie całkiem nowych typów graficznych prezentacji danych.
Początkujący użytkownik może ograniczyć się do wykorzystywania gotowych funkcji. Przez wywołanie funkcji z odpowiednimi parametrami uzyskuje się możliwość dużej modyfikacji wyglądu rysunków. Taki zachęcający przykład prezentowany jest w tym rozdziale.
Dlaczego R jest doskonały do tworzenia grafiki:

– ogromna liczba gotowych wykresów,

– nieograniczone wręcz możliwości graficzne,

– łatwość generowania gotowych rysunków w wielu formatach bitmapowych i wektorowych,

– automatyzacja tworzenia grafiki dzięki programowaniu,

– dowolna integracja z innym oprogramowaniem i wieloma źródłami danych.

Lista wybranych bibliotek graficznych

System R ma ogromną liczbę bibliotek graficznych. Niektóre z nich są uniwersalne, niektóre związane ze specyficznymi zastosowaniami. Poniżej przedstawiamy listę bibliotek mogących zainteresować początkujących użytkowników R.

Biblioteka	**Przeznaczenie**
`graphics`	standardowa biblioteka graficzna
`grid`	system graficzny bardziej elastyczny niż standardowy
`ggplot`	wygodna uniwersalna biblioteka graficzna
`rgl`	wizualizacja 3d w OpenGL
`scatterplot3d`	wykresy rozrzutu 3d
`lattice`	wizualizacja danych wielowymiarowych
`cat, vcd`	wizualizacja danych kategorycznych
`iplots`	interaktywna grafika

Funkcje graficzne wysokiego i niskiego poziomu

Funkcje graficzne w systemie R można podzielić na funkcje wysokiego poziomu i niskiego poziomu. To rozróżnienie jest ważne i pozwoli na tworzenie bardziej zaawansowanej grafiki.

– Funkcje graficzne **wysokiego poziomu** rysują cały obrazek (np. funkcja `plot()`).

– Funkcje graficzne **niskiego poziomu** uzupełniają tylko istniejące rysunki (np. funkcja `points()`).

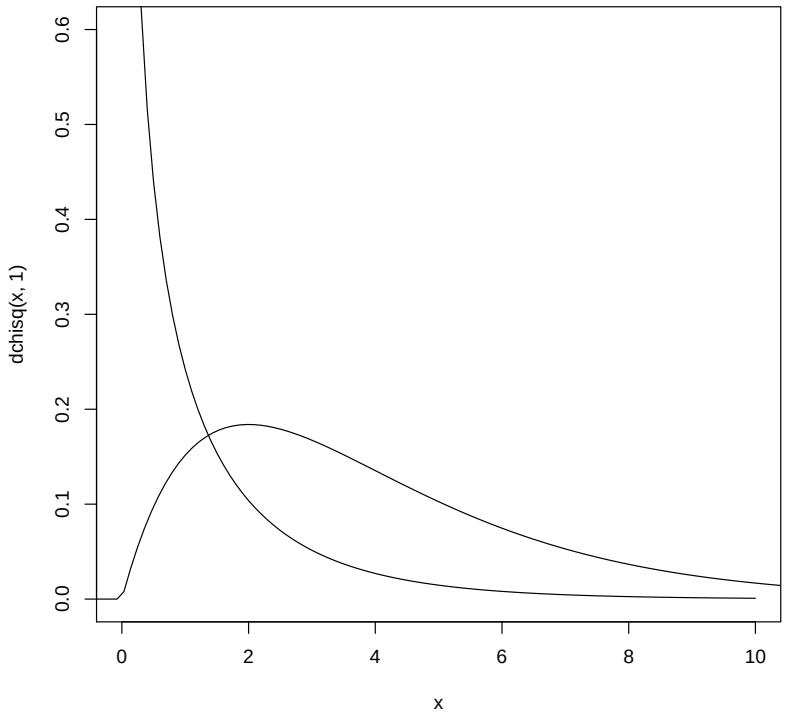
Funkcja `plot()` jest najważniejszą funkcją graficzną. Możemy ją zastosować do wielu typów obiektów. Wywołanie wyłącznie funkcji niskiego poziomu spowoduje błąd.

Przykład 1

Poniższy kod tworzy najprostszy wykres gęstości rozkładów χ^2 dla różnej liczby stopni swobody:

```
x <- seq(from=0,to=10,by=0.01)

curve(dchisq(x,1), xlim=c(0,10), ylim=c(0,.6))
curve(dchisq(x,4), add=T)
```



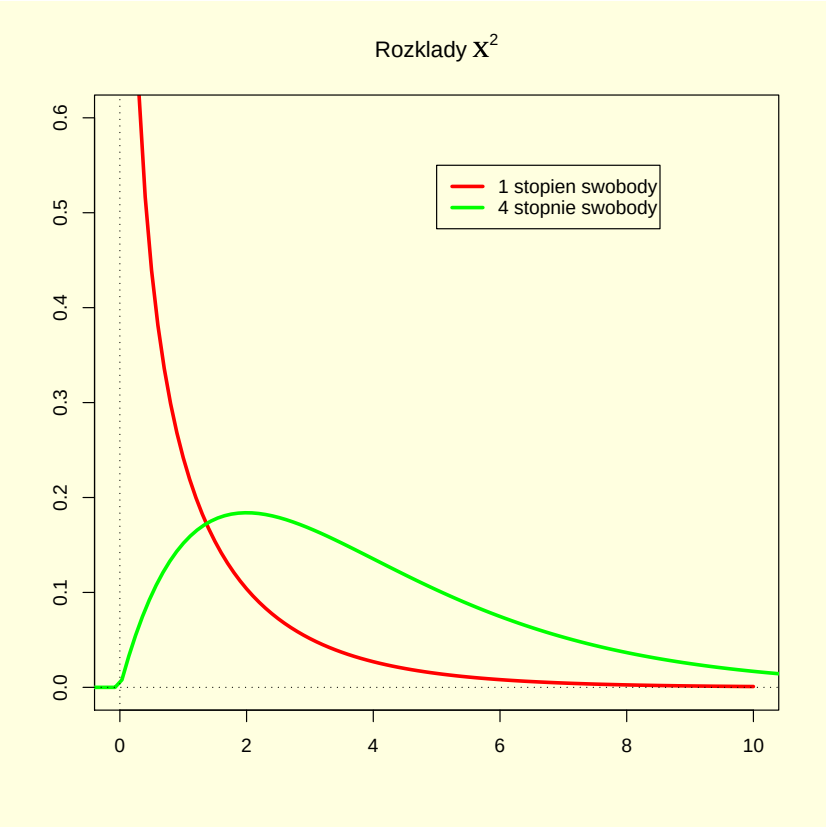
Rysunek 7.1: Najprostszy wykres

Przykład 2

Teraz uatrakcyjnimy ten wykres uzupełniając o kolory i grubości linii oraz opisy osi:

```
curve(dchisq(x,1), xlim=c(0,10), ylim=c(0,.6),
```

```
     col="red", lwd=3, main="Rozklady Chi^2", xlab="",ylab="")
curve(dchisq(x,4), add=T, col="green", lwd=3)
```



Rysunek 7.2: Zmodyfikowany wykres

Przykład 3

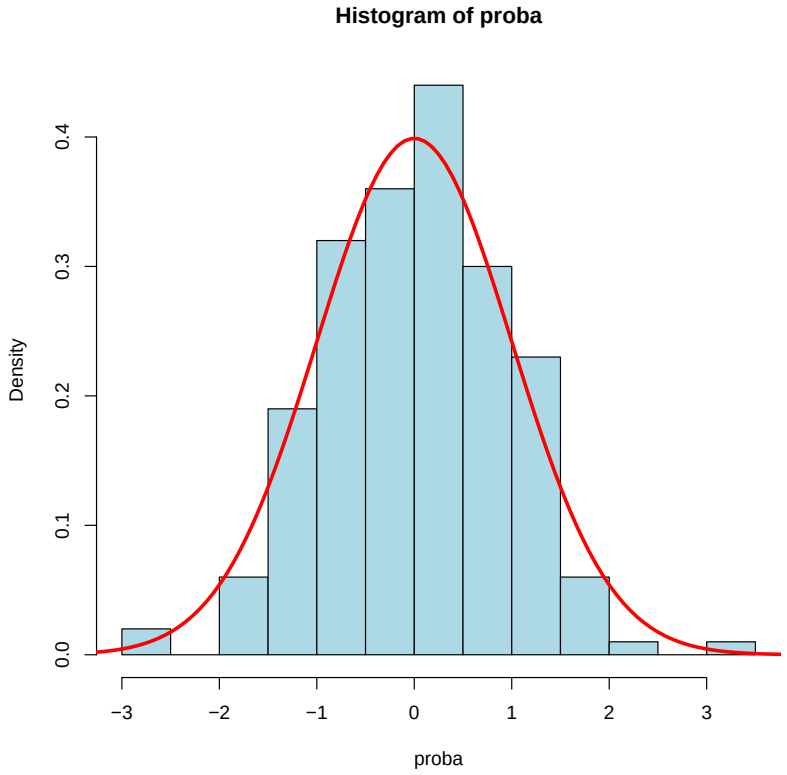
Innym przykładem ilustrującym możliwości graficzne jest histogram. Znów modyfikujemy kolory i grubości linii.

```
proba <- rnorm(200)

hist(proba,20,probability=TRUE,col="light blue")
curve(dnorm(x),lwd=3,col="red",add=TRUE)
```

Wykresy standardowe – wprowadzenie

Wykresy standardowe należą do grupy funkcji wysokiego poziomu. R posiada ogromny wybór gotowych do wykorzystania wykresów standardowych. Tworzenie wykresów standardowych



Rysunek 7.3: Funkcje wysokiego i niskiego poziomu – ilustracja

sprowadza się do wywołania odpowiedniej funkcji (ewentualnie poprzedzonego wczytaniem odpowiedniego pakietu). Wszystkie funkcje graficzne mają wiele parametrów, z pomocą których można modyfikować i udoskonalać prezentację graficzną danych.

Podstawowe wykresy jednej zmiennej

Proponujemy samodzielne zapoznanie się z podstawowymi wykresami dla jednej zmiennej poprzez własne eksperymenty albo skorzystanie z pomocy systemu R.

- `plot()`,
- `barplot()`,
- `hist()`,
- `boxplot()`,
- `pie()`.

Funkcje graficzne wysokiego poziomu – podsumowanie

Funkcja	Działanie
`plot`	podstawowa funkcja graficzna
`hist`	histogram
`pie`	wykres kołowy
`barplot`	wykres słupkowy
`boxplot`	wykres pudełkowy
`pairs`	zestaw wykresów rozrzutu
`stars`	wykres radarowy
`mosaicplot`	wykres mozaikowy

Modyfikacja grafiki

Jedną z mocnych stron R są nieograniczone możliwości modyfikacji wykresów. Możliwe jest uzyskanie praktycznie dowolnej grafiki. Modyfikacje wykresów uzyskuje się dzięki parametrom, które można określić z pomocą funkcji `par()` lub przekazywać do funkcji wysokiego poziomu. Liczba tych parametrów jest ogromna, a jej przedstawienie wykracza poza zakres tego opracowania. Proponujemy samodzielne zapoznanie się z nimi.

Zapisywanie grafiki

Jak już podkreślaliśmy, system R tworzy grafikę o jakości prezentacyjnej. Grafikę można zapisywać w wielu formatach i na wiele sposobów. Istnieją co najmniej trzy sposoby zapisywania grafiki:

– menu okienka graficznego,

– menu kontekstowe pojawiające się po naciśnięciu prawego klawisza myszy, gdy kursor jest ustawiony w okienku graficznym,

– urządzenia graficzne.

Urządzenia graficzne są najbardziej elastycznym sposobem. Korzystając z nich można określić niemal wszystkie parametry tworzonego pliku.

Przykład zapisu do pliku przy pomocy urządzenia graficznego

```
pdf(file="nazwa_pliku.pdf")
hist(rnorm(10^3))
hist(rnorm(10^4))
dev.off()
```

– otwarcie urządzenia graficznego
– operacje graficzne (każdy wykres na osobnej stronie)
– zamknięcie urządzenia graficznego (równocześnie zamyka plik)

Oczywiście istnieje możliwośc wyboru urządzenia graficznego. Najczęściej wybierane to:

`postscript(...)`,

`png(...)`,

`jpeg(...)`,

`bmp(...)`.

Pełna lista urządzeń graficznych pojawi się na ekranie po wpisaniu polecenia `Devices()`.

8 Elementy programowania w R

Jak już wiemy, język programowania jest integralną częścią systemu R, a wykonywanie analiz danych w R ma wiele wspólnego z programowaniem.
R jest zaskakująco zaawansowanym i naprawdę potężnym językiem programowania. Umożliwia pisanie dużych programów, programowanie obiektowe, obsługę wyjątków, debugowanie i optymalizację kodu (profilowanie).
W tym rozdziale chcielibyśmy przedstawić tylko najbardziej podstawowe elementy programowania: pisanie własnych funkcji, pętle i instrukcje warunkowe.

8.1 Instrukcje warunkowe

- `if (warunek)instr1 [else instr2]` pozwala na warunkowe wykonanie fragmentu kodu,
- `ifelse(warunek, instrNie, instrTak)` pozwala na skrócenie zapisu instrukcji warunkowej,
- `switch(wartosc, ...)` pozwala na warunkowe wykonanie jednego z wielu możliwych fragmentów kodu.

Przykład `if(warunek)`

Najczęściej przydatna jest instrukcja `if(warunek)`. Oto prosty przykład sprawdzający, czy liczba jest parzysta:

```
> liczba <- 42
> if (liczba %% 2 == 0){
+ cat("liczba jest parzysta\n")
+ }else{    # wazne zeby "else" bylo zaraz po "}"
+ cat("liczba jest nieparzysta\n")
+ }
liczba jest parzysta
>
```

Przykład 1 `ifelse(warunek, instrNie, instrTak)`

Jeżeli kod wykonywany warunkowo jest krótki, to można uprościć zapis wykorzystując funkcję `ifelse(warunek, instrNie, instrTak)`.

```
> jeden <- "jeden"
> dwa <- "dwa"
> ifelse(jeden == dwa, "inne", "to samo")
[1] "to samo"
```

Przykład 2 `ifelse(warunek, instrNie, instrTak)`

Trzeba zachować ostrożność przy porównywaniu liczb rzeczywistych. Nawet, zdawałoby się prosta, operacja dodawania może być źródłem błędów numerycznych, których się nie spodziewamy:

```
> jeden <- 0.4
> dwa <- 0.1
> ifelse(0.5 == jeden + dwa, "inne", "to samo")
[1] "inne"
```

Przykład 3 `ifelse(warunek, instrNie, instrTak)`

Instrukcja warunkowa `ifelse(warunek, instrNie, instrTak)` świetnie nadaje się do przetwarzania wektorów. To jest właśnie jej podstawowe zastosowanie.

```
> wektor <- rnorm(5)
> wektor
[1] -0.65426992 0.71763580 0.08559266 1.04562332 1.49848355

> ifelse(wektor<0,-1,1)
[1] -1 1 1 1 1
```

Przykład `switch(wartosc)`

Jeśli chcemy uzależnić dalsze działanie programu na przykład od kilku możliwych wersji otrzymanego wyniku, to możemy użyć rozbudowanej instrukcji warunkowej `switch(wartosc)`. Oczywiście, w tej sytuacji wszystkie możliwe wyniki powinny być nam z góry znane.

```
> wartosc <- c(1,2,3)
> switch(length(wartosc),
+ "1" = cat("jeden\n"),
+ "2" = ,
+ "3" = cat("trzy\n"),
+ cat("nie wiem ile\n")
+ )
trzy
```

8.2 Pętle

Pętle w R tylko nieznacznie różnią się od pętli spotykanych w innych językach programowania. Mamy do dyspozycji typowe trzy ich rodzaje:

- `for (iterator)instr` stosuje się, gdy liczba powtórzeń pętli jest z góry znana,
- `while (warunek)instr` stosuje się, gdy powtórzenia mają być wykonywane tak długo, jak długo prawdziwy jest pewien warunek,
- `repeat instr` stosuje się, gdy żadna z powyższych pętli nie może być łatwo zastosowana.

Przykład `for (iterator)instr`

Wewnątrz pętli `for` można umieścić dowolne instrukcje. Można w pętli umieścić również kolejne pętle wewnętrzne, ale zwykle nie jest to potrzebne.

```
for(i in 1:5){
       cat(paste("krok numer: "), paste(i, "\n"))
}
```

Warto zapamiętać, że R pozwala na użycie innych iteratorów niż liczby:

```
> iteratory <- c("jeden", "dwa", "trzy")
> for(i in iteratory){
+ cat(paste(i, "\n"))
+ }
jeden
dwa
trzy
```

Przykład `while (warunek)instr`

Jeśli chcemy wykonywać pewną operację tak długo, aż pewnien warunek przestanie być prawdziwy, ale nie wiemy dokładnie ilu powtórzeń potrzeba, to z pomocą przychodzi nam pętla `while`.
Można przerwać wykonywanie danej iteracji wpisując wewnątrz pętli `next` i wówczas zostanie rozpoczęte kolejne powtórzenie pętli. Stosowanie polecenia `next` nie znamionuje dobrego stylu programowania.

```
> liczba<-7
> while(liczba>0){
+ cat(paste("liczba = ", liczba, "\n"))
+ liczba <- liczba - 2
+ }
liczba = 7
liczba = 5
liczba = 3
liczba = 1
```

8.3 Pisanie własnych funkcji

Już po krótkim czasie pracy z R zauważymy, że pewne, specyficzne dla analizy danych, z którymi pracujemy, czynności powtarzają się. Okaże się, że trzeba kopiować do kolejnych skryptów te same fragmenty kodu. Takie powtarzające się fragmenty kodu warto zastąpić wywołaniami napisanych przez siebie funkcji. To spowoduje, że nasze skrypty staną się bardziej przejrzyste i trudniej będzie popełnić błąd.
Z czasem napisane własnoręcznie funkcje utworzą bibliotekę znacznie zwiększającą wydajność pracy z danymi. Pisanie funkcji wcale nie jest trudne!
Oto przykładowa funkcja, która znajduje największy element wektora i zwraca podwojoną jego wartość. Nie jest to przykład zbyt praktyczny, ale pokazuje wszystkie ważne aspekty pisania własnych funkcji:

```
max.razy.dwa <- function(wektor)
{
  maksimum <- max(wektor) # znajdujemy maksimum
  res <- 2 * maksimum
  res
}
```

Funkcja ta ma jeden argument: przekazujemy jej wektor liczb. Zwraca liczbę, która jest podwojoną wartością największego elementu przekazanego wektora. Wartością funkcji jest wartość ostatniego wyrażenia. W naszym przypadku jest to wartość zmiennej „res". Kod funkcji należy wpisać (lub wkleić lu wczytać z pliku) do R. Funkcję wykorzystujemy jak każdą inną:

```
> max.razy.dwa(c(1,2,7))
[1] 14
```

Naszą funkcję można zapisać także w bardziej zwięzłej postaci:

```
max.razy.dwa <- function(wektor)
{
  2 * max(wektor)
}
```

Istnieje wiele ciekawych zagadnień związanych z pisaniem funkcji w R (argumenty domyślne, funkcje anonimowe, polimorfizm), ale przekraczają one zakres tego opracowania. Aby zachęcić czytelników do nauki spróbujmy wykonać zdefiniować prostszą funkcję:

```
max.razy.dwa <- function(wektor=1) 2 * max(wektor)
```

oraz wywołać ją bez argumentów, czyli `max.razy.dwa()`. Wtedy przyjmie ona tak zwaną domyślną wartość argumentu. Wartości domyślne często wykorzystywane są w funkcjach graficznych.

9 Elementy statystyki

Zaprezentujemy krótko tylko najważniejsze funkcje zakładając, że czytelnik zna pojęcia z zakresu statystyki, którymi się posługujemy.
Posłużymy się dostępnymi w pakiecie `MASS` danymi `Cars93`. Dane dotyczą samochodów sprzedawanych na rynku amerykańskim w roku 1993. Uzupełnienie tego rozdziału można znaleźć w przykładach na końcu „Wprowadzenia do R".
Na początek wczytujemy dane:

```
library(MASS) # wczytanie pakietu
data(Cars93) # wczytanie danych
```

Teraz elementy statystyki opisowej:

```
# maksymalna i minimalna cena dla aut w wersji podstawowej
max(Cars93$Min.Price)
min(Cars93$Min.Price)

# ktory samochod jest najdrozszy?
ind.max <- which.max(Cars93$Price)
najdrozszy <- Cars93$Make[ind.max]
print(paste("Najdroszy samochod: ", najdrozszy))

mean(Cars93$Price)  # srednia cena
median(Cars93$Price) # mediana ceny
sd(Cars93$Price)  # odchylenie standardowe ceny
var(Cars93$Price)  # wariancja ceny
quantile(Cars93$Price, probs=.25) # pierwszy kwartyl

summary(Cars93$Price) # statystyki opisowe zebrane w jednym zestawieniu
```

Liczności i tablice kontyngencji:

```
# ile jest samochodow okreslonych typow?
table(Cars93$Type)

# ile samochodow okreslonego typu produkuja poszczegolne koncerny?
table(Cars93$Manufacturer, Cars93$Type)
```

Korelacja liniowa:

```
# zależność pomiedzy pojemnoscia silnika, a liczba przejechanych mil
#na jednym galonie paliwa (jazda miejska)
cor(Cars93$EngineSize, Cars93$MPG.city)
```

Podstawowe wykresy statystyczne

Podstawowe wykresy przedstawimy jedynie na przykładach, bez szczegółowego ich omawiania. Po zobaczeniu wykresów łatwo domyśleć się, co one przedstawiają.

```
# histogram ceny
hist(Cars93$Price, 30)

# wykres pudełkowy ceny
boxplot(Cars93$Price)

# wykres pudełkowy ceny z podziałem na typy samochodów
boxplot(Price ~ Type, data=Cars93)

# wykres liczebności poszczególnych typów
barplot(table(Cars93$Type))

# wykres kołowy udziałów poszczególnych typów
pie(table(Cars93$Type))

# wykres zależności między pojemnością skokową silnika a ceną samochodu
#(wykres rozrzutu)
plot(Cars93$EngineSize, Cars93$Price)
```

Statystyka – podsumowanie funkcji

Funkcja	**Działanie**
`max`	największy element
`min`	najmniejszy element
`mean`	średnia
`median`	mediana
`sd`	odchylenie standardowe
`var`	wariancja
`quantile`	kwantyle empiryczne
`summary`	statystyki opisowe
`table`	liczności, tablica kontyngencji
`cor`	współczynnik korelacji

Podstawowe wykresy statystyczne – podsumowanie

`hist`	histogram
`boxplot`	wykres pudełkowy
`barplot`	wykres słupkowy
`pie`	wykres kołowy
`plot`	uniwersalna funkcja do robienia wykresów

10 Losowanie, zmienne losowe i symulacje

Najważniejsza w tym rozdziale jest funkcja `sample`. Używamy jej często przy losowaniu elementów ze zbioru danych (na przykład przy podziale na zbiór uczący i testowy). Wtedy stosuje się ją tak:

```
data(iris)

# tutaj pojawiaja sie numery wybranych 10 obiektow
indeksy <- sample(1:dim(iris)[1], 10)

# wybieramy tylko wylosowane przypadki
iris[indeksy,]
```

Funkcja `sample` ma wiele innych zastosowań. Jako przykład pokażemy symulację rzutu monetą:

```
# losujemy 20 razy ze zbioru {"O", "R"} ze zwracaniem
wyniki <- sample(c("O","R"), replace=TRUE, size=20)
print(wyniki)
table(wyniki) # ile razy wypadl orzel, a ile reszka
```

Do ilustrowania niektórych zagadnień wykorzystujemy dane symulowane. Przy okazji opisania funkcji służących do symulacji napiszemy o rozkładach prawdopodobieństwa dodatkowe dwa zdania.
R udostępnia znakomitą większość (częściej i rzadziej wykorzystywanych w praktyce) rozkładów. Stosowana jest uniwersalna konwencja:

dnazwa – gęstość rozkładu „nazwa”,

pnazwa – dystrybuanta,

qnazwa – kwantyle / dystrybuanta odwrotna,

rnazwa – generator liczb pseudolosowych.

W przypadku rozkładu normalnego oznacza to odpowiednio:

– dnorm(.),

– pnorm(.),

– qnorm(.),

– rnorm(.).

Przykłady

Zastosowanie funkcji związanych z rozkładami prawdopodobieństwa najlepiej jest prześledzić na prostych przykładach.

```
# gestosc rozkladu normalnego w 0
dnorm(0)

# dystrybuanta w 0
pnorm(0)

# ile masy standardowego rozkladu normalnego N(0,1) miesci sie miedzy -3 a 3?
pnorm(3) - pnorm(-3)

# 100 obserwacji z rozkladu N(2,1)
rnorm(100,mean=2,sd=1)

# pierwszy kwartyl
qnorm(0.25)

# zobaczmy, czy przypomina rozklad normalny
hist(rnorm(1000))

# proba z rozkladu jednostajnego i histogram dla takiej proby
runif(1000)
hist(runif(1000), prob=TRUE)
```

Zmienne losowe i symulacje – podsumowanie

Funkcja	Działanie
`sample`	losowanie indeksów obiektów, inne rodzaje losowań
`dnorm`	gęstość rozkładu normalnego
`pnorm`	dystrybuanta rozkładu normalnego
`qnorm`	kwantyl rozkładu normalnego
`rnorm`	liczby pseudolosowe z rozkładu normalnego
`runif`	liczby pseudolosowe z rozkładu jednostajnego

11 Przykładowe sesje z R

W tym rozdziale przedstawiamy kilka przykładowych sesji w R. Pokazują one jak przebiega analiza danych rzeczywistych. Zachęcamy do sprawdzenia zaprezentowanych kodów w R fragment po fragmencie oraz obserwowania rezultatów. W każdym kroku można zobaczyć, jakie obiekty pojawiły się w pamięci R (nazwa.obiektu + ENTER lub print(nazwa.obiektu)).
Sesje takie można zapisać do plików. Wtedy nazywamy je skryptami R i dzięki nim bez problemu można powtórzyć zapisaną analizę.
Po przeczytaniu całego „Wprowadzenia do R" jego czytelnicy będą mogli już samodzielnie modyfikować przedstawione w tym rozdziale skrypty.

Prosty model liniowy

Ten przykład ilustruje prostotę, z jaką buduje się w systemie R modele statystyczne. Ilustruje także działanie dwóch ważnych funkcji: `plot` oraz `summary`. Są one używane dla różnych typów obiektów.

```
library(MASS)                                          – wczytanie biblioteki
data(iris)                                             – wczytanie danych
model.reg <-lm(Petal.Length ~Sepal.Length, data=iris)  – budowa modelu regresyjnego
summary(model.reg)                                     – tak wygląda model
plot(model.reg)                                        – zestaw wykresów diagnostycznych
```

Można operować na wynikach zawartych w zmiennej `model.reg` i wykorzystać je do dalszych obliczeń.

Przykład praktycznej analizy danych (trochę bardziej zaawansowanej)

W tym przykładzie pokazujemy, jak wykryć zależność między zmiennymi ciągłymi oraz jak zbudować opisujący ją model regresyjny. Uwaga! W tym momencie nie zajmujemy się w ogóle aspektami statystycznymi, na przykład takimi jak jakość dopasowania modelu.

```
library(MASS) # wczytujemy biblioteke MASS, w ktorej sa dane
data(Cars93) # wczytujemy dane

# przyrzyjmy się zmiennym: cena samochodu, pojemnosc skokowa silnika
hist(Cars93$Price)
hist(Cars93$EngineSize)

# zobaczymy, czy one od siebie zaleza (widac zaleznosc, ale niezbyt liniowa)
plot(Cars93$EngineSize, Cars93$Price)
```

```
# korelacja jednak jest dosyc wysoka
cor(Cars93$EngineSize, Cars93$Price)

# spróbujmy zbudować model regresji liniowej
# model da nam odpowiedz, ile musimy zaplacic,
# jesli chcemy miec samochod o okreslonej pojemnosci silnika
library(MASS)
model <- lm(Cars93$Price ~ Cars93$EngineSize, data=Cars93)

# zobaczmy, jak model wyglada
print(model)

# spojrzmy, czy model jest dobrze dopasowany
plot(model)

# prognozowane ceny na podstawie pojemnosci skokowej
fitted(model)

# spojrzmy na wykres
plot(Cars93$EngineSize, Cars93$Price)
abline(model)
points(Cars93$EngineSize, fitted(model), col="red", pch=20, cex=1.5)
```

Przykład nieco bardziej zaawansowanej grafiki

Oto przykład wykorzystania funkcji graficznych wysokiego i niskiego poziomu. Proponujemy po wpisaniu kodu do R powtórzyć kilkukrotnie polecenie rozpoczynające się od `points`. Zrozumienie działania przykładu pozostawiamy jako ćwiczenie.

```
# nic nie rysuje (type="n"), tylko definiuje zakresy osi
plot(0,0,xlim=c(-3,3),ylim=c(-3,3),type="n")

# warto uruchomic kilkukrotnie!
points(rnorm(20),rnorm(20),pch=1:20,cex=5*rnorm(20),col=rainbow(20))
```

Bibliografia

[1] Books related to R. Internet, http://www.r-project.org/doc/bib/R-books.html.

[2] M.J. Crawley. Statistical Computing. An Introduction to Data Analysis Using S-Plus. Wiley, 2006.

[3] P. Dalgaard. Introductory Statistics with R. Springer, 2004.

[4] J.J. Faraway. Linear Models with R. Chapman & Hall/CRC, 2004.

[5] M. Lavine. Introduction to Statistical Thought, http://www.stat.duke.edu/~michael/book.html. 2006.

[6] P. Murrell. R Graphics. Chapman & Hall/CRC, 2005.

[7] F. Romain. R Graph Gallery, http://gallery.r-enthusiasts.com/. Internet.

[8] W. N. Venables and B. D. Ripley. Modern Applied Statistics with S. Springer, 2002.

[9] V. Zoonekynd. Statistics with R, http://zoonek2.free.fr/UNIX/48_R/all.html. Internet.