Materiały dodatkowe do ćwiczeń projektowych

Przedmiot: Data Mining - metody eksploracji danych

Studia Tutorskie

Z. M. Łabęda-Grudziak

Instytut Automatyki i Robotyki

Rok akademicki 2023/2024

Regulamin i zasady zaliczenia

- Systemem obliczeń i językiem programowania obowiązującym na zajęciach projektowych jest R
- W ramach ćwiczeń projektowych możliwe jest otrzymanie 15 pkt. za projekt indywidualny i 25 pkt za projekt grupowy.
- Ocena punktowa z ćwiczeń projektowych jest sumą liczby punktów za zadania ze wszystkich projektów - w sumie maksymalnie 40 pkt.
- Podstawą do zaliczenia ćwiczenia projektowego są: sprawozdanie, program obliczeniowy i prezentacja. Sprawozdanie powinno zawierać następujące punkty:
    1. Imię i nazwisko studenta (studentów)
    2. Numer ćwiczenia i treść zadań
    3. Analiza wyników.
    4. Kody źródłowe wszystkich procedur.
- Jeżeli zachodzi uzasadnione podejrzenie, że rozwiązanie zadania nie odbyło się samodzielnie prowadzący może obniżyć ocenę włącznie z niezaliczeniem zadania.
- Do zaliczenia ćwiczeń projektowych wymagane jest zdobycie łącznie co najmniej 20 pkt.
- Zaliczenie ćwiczeń projektowych jest warunkiem zaliczenia przedmiotu.

SPIS TREŚCI

I. WSTĘP

Na ćwiczeniach projektowych z metod eksploracji danych będziemy posługiwali się pakietem R (*https://www.r-project.org*). Pod tą samą nazwą będziemy również rozumieli język programowania. R jest środowiskiem, w którym są zaimplementowane metody statystyczne oraz metody analizy i wizualizacji danych.

System R jest dostępny na praktycznie wszystkich platformach: Windows, Linux, MacOS. System R posiada graficzny interfejs użytkownika o dosyć ograniczonych możliwościach (tylko na platformie Windows). Można 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/`)

System R posiada 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 ogromna zaleta to dostępność kodów źródłowych wszystkich metod zaimplementowanych w R.

Poniższy wstęp ma charakter informacyjny i nie wyczerpuje dokładnie całości zagadnień. Dlatego zachęcam do eksperymentowania i do korzystania z wielu źródeł i pomocy jak np. dokumentacji dostępnej pod adresem:

- http://cran.r- project.org/doc/contrib/Komsta-Wprowadzenie.pdf
- http://cran.r-project.org/doc/manuals/R-intro.html
- http://cran.r-project.org/doc/manuals/R-lang.html
- http://cran.r-project.org/doc/contrib/R_language.pdf
- http://cran.r-project.org/doc/contrib/refcard.pdf
- https://www.datacamp.com/community/tutorials/machine-learning-in-r

Pomocny może być np. blog: http://www.r-bloggers.com/. Sam R posiada również wbudowaną pomoc on-line i dokumentacje, do której dostęp dają odpowiednio polecenia:

```
> help(nazwa) # pomoc na temat konkretnej funkcji
> help(''nazwa'') # pomoc na temat konkretnej funkcji
> help.start() # pomoc w formacie HTML
> help.search() # pomoc w formacie HTML
> example(nazwa) # przykładowe zastosowanie funkcji
> find(nazwa) # pakiet, w którym dostępna jest dana funkcja
```

II. INSTALACJA R

Wymagania sprzętowe i programowe dla R są minimalne. 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 3.5.1 for Windows"
- instalacja: uruchomienie ściągniętego pliku
- 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... "

Pierwszy skrypt będzie sprawdzał jaki jest katalog roboczy oraz jak ustalić nowy katalog roboczy.

```
getwd () sprawdzanie katalogu roboczego
setwd ("/ Users / michael / Desktop / ksap /") ustawianie katalogu roboczego
```

Każda funkcja jest wywoływana jako nazwa, po której podawana jest lista argumentów rozdzielonych przecinkami.

A. Pakiety

Po zainstalowaniu R mamy do dyspozycji duże możliwości, ponieważ dostarczany jest on z kilkudziesięcioma podstawowymi pakietami. Zdecydowana większość pakietów umieszczona jest na serwerach **CRAN** (*Comprehensive R Archive Network*).

Aby zainstalować dodatkowy pakiet należy wybrać z menu `Packages` opcję `Install package(s)...` . Przy instalacji R zapyta z jakiego serwera chcemy skorzystać (można

wybrać dowolny, np. najbliżej nas zlokalizowany), a następnie wyświetli listę pakietów gotowych do zainstalowania. Automatycznie zostaną zainstalowane pakiety, które są niezbędne do poprawnego działania wybranego przez nas pakietu. Jeżeli chcemy korzystać z pakietu, to po zainstalowaniu trzeba go załadować. Do tego celu służy polecenie:

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

```
> library() # sprawdzenie listy pakietów
> library(pakiet) # wczytanie pakietu do pamięci
> library(help = pakiet) # opis pakietu
> help(package = pakiet)
> detach("package:pkg", unload = TRUE) # usuwanie pakietu z pamieci
> search() # lista wczytanych do pamięci pakietów
```

B. Instalacja Rattle

```
> install.packages("rattle")
```

W trakcie instalacji i po mogą zostać zainstalowane pakiety będące zależnościami Rattle, np. `Gtk2`. Wprowadzamy dwa następujące polecenia w wierszu R. To załaduje pakiet Rattle, a następnie startuje go.

```
> library(rattle)
> rattle()
```

- Rattle to prosty interfejs z kilkoma zakładkami. Poruszać się będziemy od lewej zakładki do prawej, które odzwierciedlają typowy proces eksploracji danych.
- Aby poprawnie używać Rattle musimy dostarczyć wymagane informacje dla aktualnej zakładki (co sugeruje tekst na pasku statusu) i wcisnąć przycisk Execute (lub F2), aby wykonać akcję (zawsze przed przejściem do kolejnego kroku).
- Pasek statusu informuje, czy akcja jest zakończona. Komunikaty języka R (np. błędy) mogą pojawić się w konsoli R, z której wystartowaliśmy R.
- Kod języka R przekazywany przez Rattle do wykonania pojawia się w zakładce Log. Dzięki temu możemy je przeglądać i ewentualnie kopiować jako tekst i uruchamiać w konsoli R. Możemy również zapisać całą sesję do pliku R. W tym celu wybieramy polecenie Export i zapisujemy plik R
- Aby wyjść z Rattle po prostu wciskamy przycisk Quit. Nie spowoduje to na ogół wyjścia z konsoli R , aby ją opuścić musimy wpisać komendę `q()` .

III. PODSTAWY R

A. Podstawowe własności języka R

Język R jest językiem interpretowanym a nie kompilowanym. Korzystanie z R sprowadza się do podania ciągu komend, które maja zostać wykonane. Kolejne komendy mogą być wprowadzane linia po linii z klawiatury lub też mogą być wykonywane jako skrypt (czyli plik tekstowy z zapisaną listą komend do wykonania).

Podstawowe własności R:

- Jest wrażliwy na wielkość znaków, więc symbole X i x, to różne symbole odwołujące się do innych zmiennych.
- Znaki używane w nazwach zmiennych zależą od systemu operacyjnego i jego lokalizacji (locale). Wszystkie znaki alfanumeryczne są dopuszczalne oraz znaki "." i "_".
- Nazwa musi rozpoczynać się od kropki lub litery, a jeśli pierwsza jest kropka to kolejnym znakiem nie może być cyfra. Nazwy nie są ograniczone jeśli chodzi o liczbę znaków.
- Nie wymaga deklarowania obiektów i określania ich typu przed pierwszym użyciem (w przeciwieństwie do większości języków programowania), co z jednej strony jest ułatwieniem, ale z drugiej wymaga precyzji i wzmożonej uwagi.
- Komendy są oddzielane średnikami lub znakiem nowej linii.
- Komentarze tworzymy za pomocą znaku hasz #, wszystko od tego znaku do końca wiersza jest komentarzem. Jeśli komenda jest długa i nie mieści się w wierszu, to po przejściu do nowego wiersza R zmieni znak zachęty na +.
- Podczas sesji R, obiekty tworzymy przypisując im nazwę. Polecenie języka R `objects()` (lub `ls()`), może być użyte do wyświetlenia nazw obiektów, które są przechowywane w aktualnej sesji R . Zbiór obiektów aktualnie przechowywanych w sesji, to przestrzeń robocza (**workspace**).
- Na końcu sesji mamy możliwość zapisania tej przestrzeni roboczej w aktualnym katalogu, obiekty będą zapisane w pliku **RData**, a polecenia w pliku **.Rhistory**.

- Jeśli wystartujemy ponowie R w tym katalogu, to przestrzeń robocza będzie wczytana z tych plików (obiekty i historia poleceń).

- Podczas pracy w trybie interaktywnym wyniki obliczenia pojawiają się natychmiast w oknie programu R. Inaczej jest przy uruchamianiu skryptów zewnętrznych. W takim przypadku w konsoli środowiska R pojawiają się tylko te wartości wyrażeń, które zostaną wprost wskazane przez programistę za pomocą funkcji `print()`.

B. Podstawowe działania arytmetyczne

R możemy używać jako wygodnego kalkulatora:

```
> 4*5 # mnozenie
> 4/5 # dzielenie
> 5 %% 4 # reszta z dzielenia
> 5 %/% 4 # czesc calkowita z dzielenia
> 2^3 # potegowanie
> exp(2) # e^2
> sqrt(3) # pierwiastek kwadratowy
> log(3) # logarytm naturalny z 3
> log(3,10) # logarytm z 3 o podstawie 10
> abs(-3) # wartosc bezwzgledna
```

C. Użyteczne polecenia R

- `help()` - wywołuje pomoc
- `q()` - zamykamy R
- `library(nazwa)` - wczytanie pakietu o wskazanej nazwie
- `data(nazwa)` - wczytanie zbioru danych
- `search()` - ścieżka poszukiwań R
- `ls()` - dostępne obiekty
- `getwd()`, `setwd()` - ustawienie i sprawdzenie katalogu roboczego
- `source(śkrypt.R")` - wczytanie kodu R z pliku
- `history()` - przejrzenie historii wykonanych poleceń

IV. TYPY DANYCH

Wszystkie liczby rzeczywiste w R są przechowywane jako typy `double` (podwójnej precyzji). Można rozróżnić liczby całkowite (`integer`), rzeczywiste (`double`) czy nawet zespolone (`complex`).

```
> x=32
> y = as.integer(x)
> typeof(x)
[1] "double"
> typeof(y)
[1] "integer"
> typeof(y)
[1] "integer"
> is.double(x)
[1] TRUE
> is.integer(x)
[1] FALSE
> is.double(y)
[1] FALSE
> is.integer(y)
[1] TRUE
```

Liczby zespolone oraz wartości specjalne:

```
> sqrt(-1)
[1] NaN
> sqrt(-1 + (0+0i))
[1] 0+1i
```

```
> 10^480
[1] 1e+480
> 10^480 + 10^3480
```

```
[1] Inf
> help(Inf)
> Inf + Inf
[1] Inf
> Inf - Inf
[1] NaN
```

Wartość `NaN` jest specjalną wartością oznaczającą "nie liczbę". Specjalnymi wartościami liczbowymi są nieskończoności (`Inf`, `-Inf`).

Wartości brakujące i puste Pojawiające się w poprzednich przykładach wartość "nie liczba" `NaN` jest szczególnym przypadkiem wartości brakującej `NA`. Dodatkowo, rozróżnia się wartości niezdefiniowaną (pustą) NULL.

```
> x = c(0, NULL, NA, NaN)
> x
[1] 0 NA NaN
> is.na(x)
[1] FALSE TRUE TRUE
> is.nan(x)
[1] FALSE FALSE TRUE
> is.null(x)
[1] FALSE
> as.double(NULL)
numeric(0)
```

Wiele funkcji posiada parametr-flagę `na.rm`, mówiący czy wpierw usunąć brakujące wartości.

```
> sum(NA, 4, 5)
[1] NA
> sum(NA, 4, 5, na.rm = TRUE)
[1] 9
```

Wartości logiczne to odpowiednio wyróżnione wartości `TRUE`, `FALSE`, ale też `NA`. Można je również otrzymać z użyciem operatorów `<`, `<=`, `>`, `>=`, `==`, `!` oraz logicznych `&`, `|` i `!`. Wartość `0` reprezentuje wartość `FALSE` w wyrażeniach logicznych, pozostałe liczby reprezentują wartość `TRUE`.

V. STRUKTURY DANYCH

W języku R jest wiele tzw. podstawowych struktur danych. Najważniejsze dwie struktury danych: wektory i listy, oraz ramka danych.

```
> help(typeof)
```

A. Wektory

Wszystkie możliwe operacje odbywają się wektorowo po elementach. Wektor może zawierać tylko jeden typ danych. W R każda liczba to jest już wektor.

Podsumowanie **operacji na wektorach**:

- `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

```
> 1:30
> 30:1 # sekwencja malejąca
> n = 10
> 1:n-1 # dwukropek ma priorytet, zatem otrzymamy 1:10, pomniejszone o 1
> 1:(n-1) # a teraz sekwencja 1:9
> seq(along=dane) # sekwencja od 1 do długości zmiennej dane
> rep(1:5,5) # powtarzamy 1:5 pięć razy
> rep(1:5,each=5) # powtarzamy każdy z elementów 5 razy
> rep(1:5,length.out=43) # wektor o długości 43,
                         #składający się z powtórzeń 1:5
> a = seq(-1,1,length=10)
> a
 [1] -1.0000000 -0.7777778 -0.5555556 -0.3333333 -0.1111111  0.1111111
 [7]  0.3333333  0.5555556  0.7777778  1.0000000
> a[2] # drugi element
> a[-2] # wszystkie oprócz drugiego
> a[c(1,5)] # pierwszy i piaty
> a[-c(1,5)] # wszystkie oprócz nich
> a > 0 # wektor logiczny - które większe od zera
> a[a>0] # ten sam wektor jako indeks, czyli wypisze te liczby

> mean(a) # średnia z danych
> a/mean(a) # każdą liczbę podziel przez ich średnią
> a*c(1,2) # co drugą liczbę pomnóż przez 2 - ostrzeżenie że wektor dłuższy
           # nie jest wielokrotnością krótszego
> b = a * c(1,2) # co drugi element mnożymy przez 2
> a-b # różnice między elementami
> b[c(1,3,5)] = c(10,11,12) # wstawiamy 10,11,12 na 1,3 oraz 5 pozycje
> a[1:5] = 0 # zerujemy 5 pierwszych elementów
> a[8] = b[8] # 8 element wektora a jest równy 8 elementowi b
```

```
> a = 1/a # wektor a zawiera odwrotności dotychczasowych wartości


> sum(a) # suma elementów a
> sum(a>3) # ile elementów jest większych od 3?
> sum(a[a>3]) # zsumujmy te elementy
> pmin(a,b) # wartości minimalne
> pmax(a,b) # wartości maksymalne
> length(a)
> c=c(a,b) # a teraz łączymy wektory
> sort(c) # sortowanie
> which(c>3) # które elementy są większe niż 3?
> range(c) # jaki jest zakres (min i max?)
> cummin(c) # najmniejsza wartość dotychczasowa
> cummax(c) # największa wartość dotychczasowa
> diff(c) # różnice miedzy kolejnymi wartościami
```

B. Macierze

Podsumowanie **operacji na macierzach**:

- `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
- `diag` - przekątna lub tworzenie macierzy przekątniowej

```
> rbind(1:3, 4:6)

[,1] [,2] [,3]
[1,] 1 2 3
[2,] 4 5 6

> cbind(1:3, 4:6)

[,1] [,2]
[1,] 1 4
```

```
[2,] 2 5
[3,] 3 6

> cbind(rbind(1:3, 4:6), c(7, 8))

[,1] [,2] [,3] [,4]
[1,] 1 2 3 7
[2,] 4 5 6 8

> a = array(1:12, c(3, 4))
> a
     [,1] [,2] [,3] [,4]
[1,]    1    4    7   10
[2,]    2    5    8   11
[3,]    3    6    9   12
> a*a
     [,1] [,2] [,3] [,4]
[1,]    1   16   49  100
[2,]    4   25   64  121
[3,]    9   36   81  144
>  diag(a)
[1] 1 5 9
>  dim(a) = c(2, 3, 2)
> aperm(a, c(2, 1, 3))
, , 1

     [,1] [,2]
[1,]    1    2
[2,]    3    4
[3,]    5    6
```

```
, , 2


     [,1] [,2]
[1,]    7    8
[2,]    9   10
[3,]   11   12


>  solve(diag(1:3), rep.int(1, 3))
[1] 1.0000000 0.5000000 0.3333333
> solve(diag(1:3))
     [,1] [,2]      [,3]
[1,]    1  0.0 0.0000000
[2,]    0  0.5 0.0000000
[3,]    0  0.0 0.3333333
>  b = outer(1:3, 1:3)
> b
     [,1] [,2] [,3]
[1,]    1    2    3
[2,]    2    4    6
[3,]    3    6    9
```

C. Listy

Lista jest obiektem zawierającym obiekty dowolnych typów. Elementy listy mogą posiadać nazwy. Listy często wykorzystywane są przez funkcje do zwracania wyników.

```
> student <-list(imie="Krzysztof", wiek=28, zonaty=F)
> student[[1]]
[1] "Krzysztof"
> student[[2]]
[1] 28
> student[[3]]
[1] FALSE
```

D. Ramki danych

Przypadkiem szczególnym listy jest ramka (`data.frame`). Ramka danych zachowuje się jak macierz. Ramki danych 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. Dostęp do elementów ramki danych można uzyskać tak samo, jak do elementów macierzy.

Podsumowanie **operacji na ramkach danych**:

- `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

```
> x = cbind(data.frame(1:3, c(TRUE, TRUE, NA)), letters[1:3])
> x
  X1.3 c.TRUE..TRUE..NA. letters[1:3]
1    1              TRUE            a
2    2              TRUE            b
3    3                NA            c
> dimnames(x) = list(LETTERS[1:3], c("int", "logi", "char"))
> x[["int"]]
[1] 1 2 3
> x[1]
  int
A   1
B   2
C   3
> x[, 1]
[1] 1 2 3
> x[, 1, drop = FALSE]
  int
A   1
B   2
C   3
> x[2:3]
  logi char
A TRUE    a
```

```
B TRUE    b
C   NA    c
```

Ramki mogą służyć do wczytywania danych z zewnętrznych plików, jako format wejściowy obiektów. Używa się do tego funkcji `read.table`. Wczytywane pliki mogą posiadać nagłówki kolumn i wartości rozdzielane wybranymi znakami np. przecinkami (pliki .csv) lub tabulatorami (pliki .txt).

```
> help(read.table)
> 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)
```

- `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

Możliwe jest także wywołanie z jedynym argumentem:

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

```
> 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
```

Do zmiennych można uzyskać bezpośredni dostęp. Przyspiesza to pracę z poziomu linii komend i skraca zapis.

```
> attach()
> detach()
```

VI. KONSTRUKCIE PROGRAMISTYCZNE

R oferuje język skryptowy zawierający typowe instrukcje jak **if**, **while**, **for**. Instrukcja warunkowa **if** przyjmuje postać:

```
> if(warunek){
> instrukcje
> } else {
>  instrukcje
> }
```

```
> x <- -2
> if(x > 0){
>    print ("Mozna obliczyć pierwiastek kwadratowy")
> } else {
> print ("Nie mozna obliczyć pierwiastka kwadratowego")
> }
```

Wektorowe wersja `if()` to funkcja `ifelse()`, która jest znacznie szybsza. Zgodnie z wektorem logicznym podanym jako pierwszy argument wybiera elementy z odpowiednich pozycji wektorów podanych jako dwa kolejne argumenty, odpowiadającym wartościom `TRUE` i `FALSE`.

```
> x <- rnorm (20)
> y <- rep (0, length (x))
> ifelse (x > 0, x, y)
```

W instrukcji w opisie warunku stosujemy następujące operatory relacji: `==`, `<`, `>`, `<=`, `>=`. Operatory logiczne są następujące: `&& (and)`, `|| (or)`. Jeżeli ich argumenty nie są jednoelementowe to brane są pod uwagę tylko pierwsze elementy.

Do dyspozycji mamy standardowy zestaw pętli znanych z choćby języka C. A mianowicie pętlę **for**:

```
> for(iterator){
>   instrukcje
> }
> end
```

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"))}
krok numer:  1
krok numer:  2
krok numer:  3
krok numer:  4
krok numer:  5
```

R pozwala na użycie innych iteratorów niż liczby:

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

Formuła pętli **while** jest następująca:

```
> while(warunek){
>     instrukcje
> }
```

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

VII. WŁASNE FUNKCJE

Wszystkie zadania wykonywane przez nas w środowisku R to wywołania funkcji. R pozwala na definiowanie własnych funkcji przy pomocy składni

```
> nazwa.funkcji = function(parametr1, parametr2, ...){
+ ...
+ }
```

Nazwa funkcji to po prostu nazwa zmiennej pod którą zapisany zostanie nagłówek oraz zestaw instrukcji do wykonania, tzn. funkcje są po prostu kolejnymi obiektami środowiska R.

VIII. PREZENTACJA WYNIKÓW

R oferuje bardzo szerokie możliwości wizualizacji danych. Oprócz setek gotowych do użycia wykresów możliwa jest ich nieograniczona modyfikacja, a także tworzenie całkiem nowych typów graficznych prezentacji danych.

Lista wybranych bibliotek graficznych

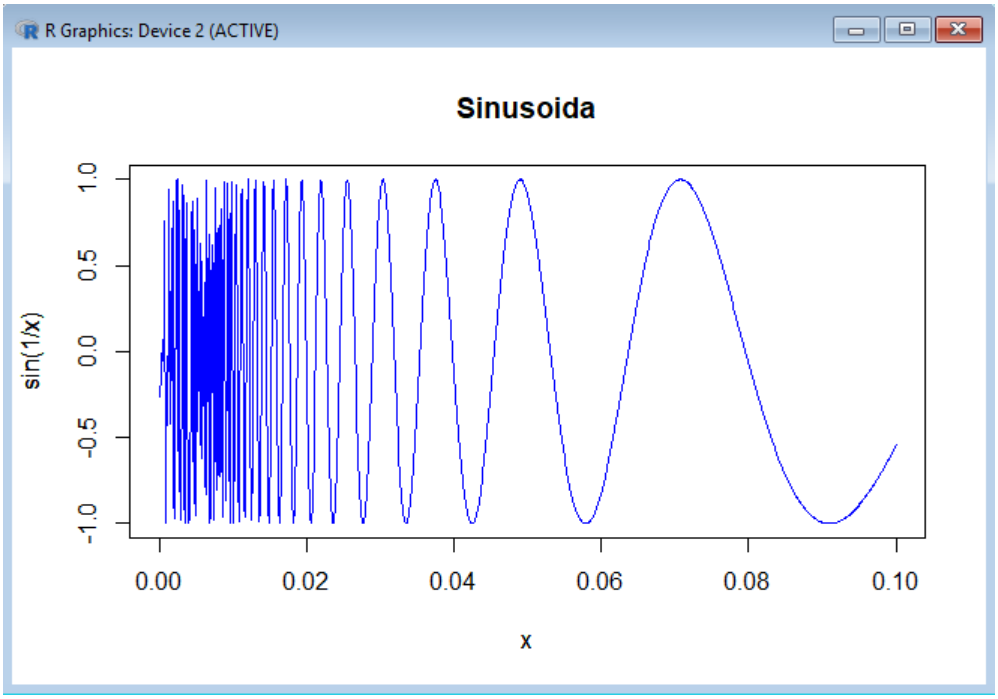
- `graphics` - standardowa biblioteka graficzna
- `grid` - system graficzny bardziej elastyczny niż standardowy
- `ggplot` - wygodna uniwersalna biblioteka graficzna
- `scatterplot3d` -wykresy rozrzutu 3D
- `lattice` - wizualizacja danych wielowymiarowych
- `cat, vcd` - wizualizacja danych kategorycznych

Lista wybranych funkcji graficznych

- `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

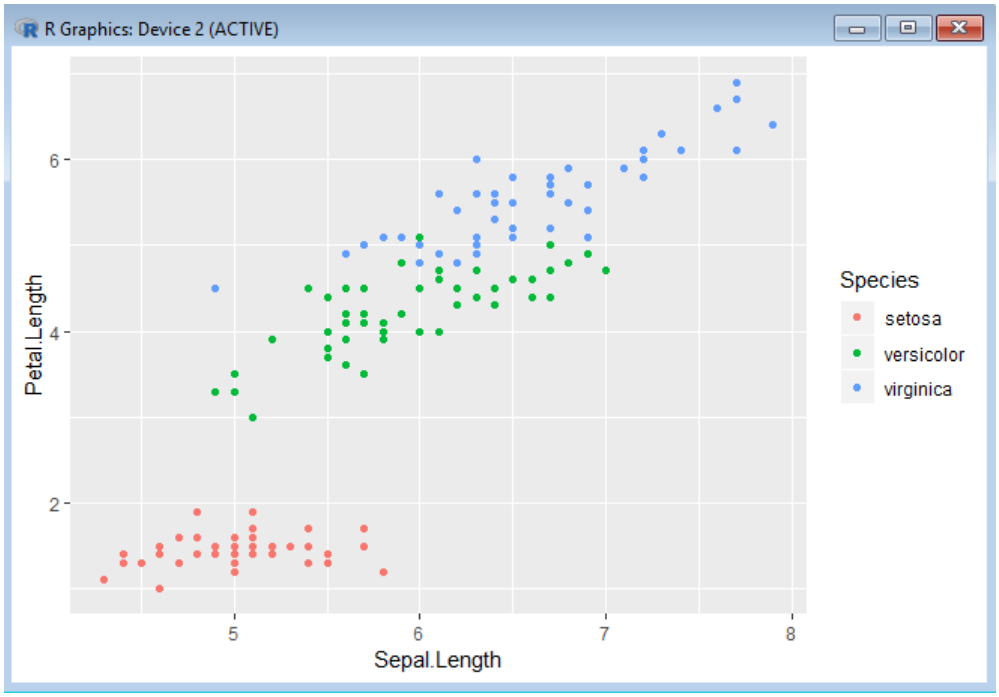
Za tworzenie **wykresów 2D** odpowiedzialna jest funkcja `plot`. Polecenie to przyjmuje szereg parametrów. Opis tych parametrów uzyskamy po wykonaniu skorzystaniu z pomocy: `help(plot)`.

```
> x = seq(0, 0.1, length = 10^3)
> plot(x, sin(1/x), type = "l")
> title(main = "Sinusoida", xlab = "x", ylab = "sin(1/x)")
```



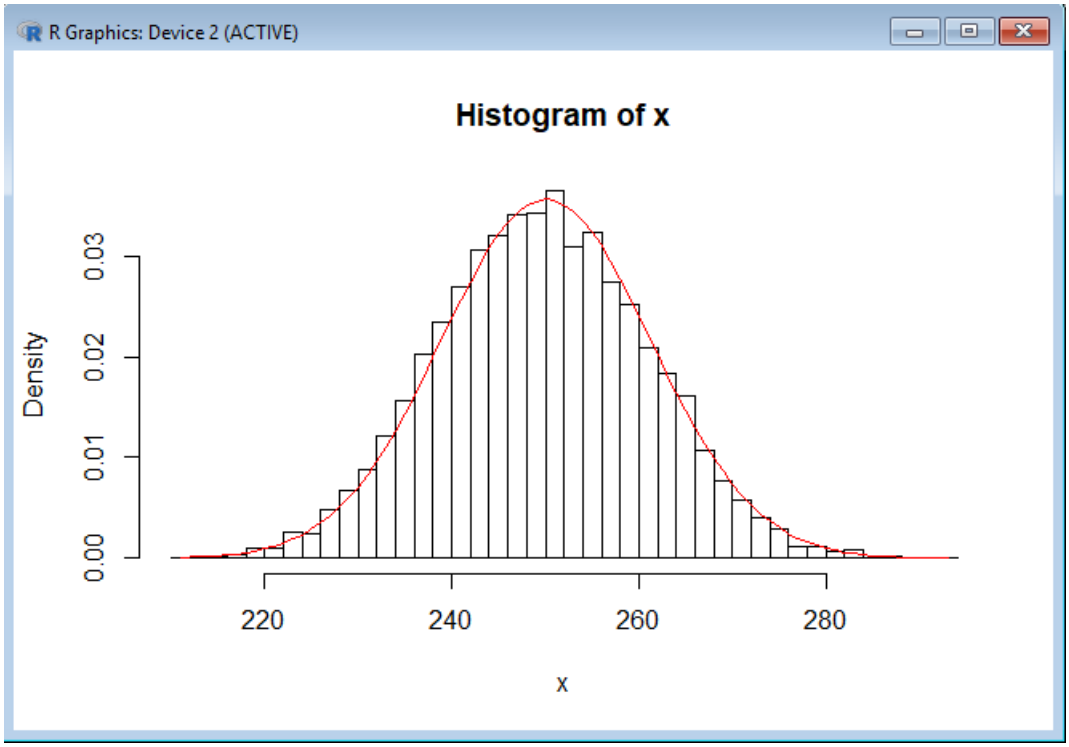
Pakiet `ggplot2` jest jednym z najbardziej zaawansowanych narzędzi do tworzenia wykresów statystycznych. Zaawansowanie nie oznacza, że można szybko zrobić w nim wykres, ani też, że dostępnych jest wiele szablonów wykresów. Oznacza, że konstrukcja pakietu jest na tyle elastyczna, że można z nim wykonać praktycznie każdą grafikę statystyczną.

```
> library(ggplot2)
> ggplot(iris, aes(x = Sepal.Length, y = Petal.Length)) +
  geom_point(aes(color = Species))
```



Do rysowania **histogramów** służy funkcja `hist`. Histogram umożliwia nam zaprezentowanie rozkładu zmiennej liczbowej. Rezultatem jest możliwość odczytania częstości występowania poszczególnych wartości w serii danych.

```
> x=rbinom(10000, 500, 1/2)
> hist(x)
> hist(x, breaks=40, probability=T) # narzucamy liczbę „przedziałów” hist
> x=sort(x)
> lines(x, dnorm(x,mean(x), sd(x)))
```

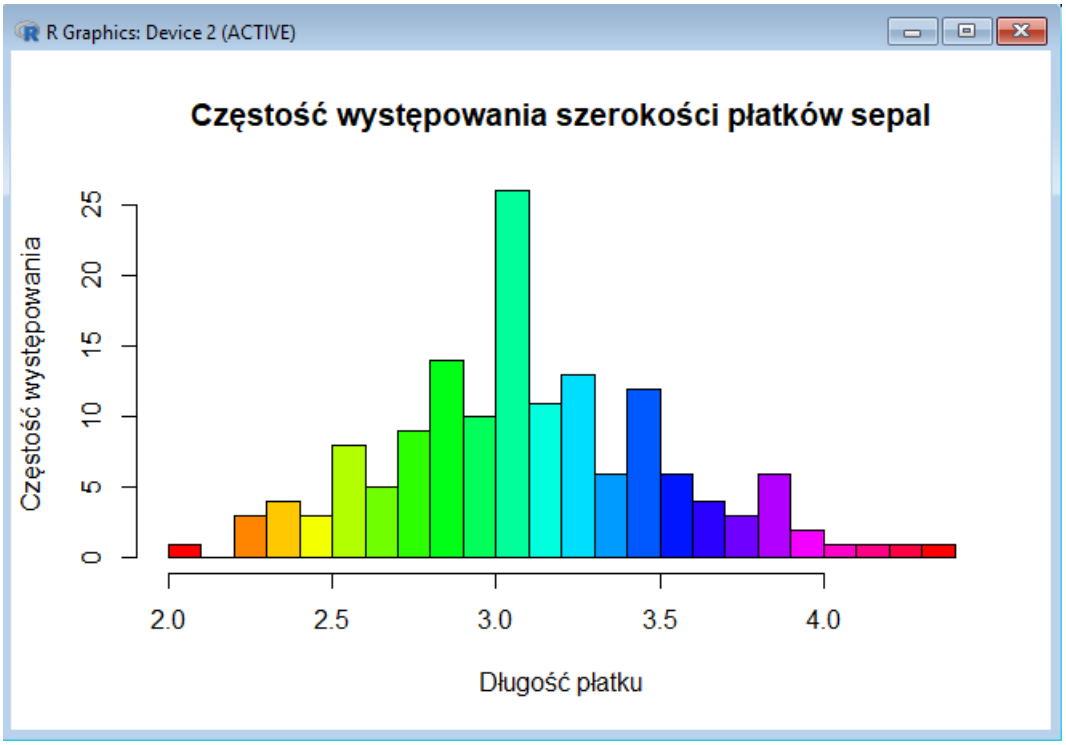



```
> liczba <- length(unique(iris$Sepal.Width))
> hist(iris$Sepal.Width,
     breaks = liczba, # liczba słupków to liczba unikalnych wartości
     right = FALSE,
     main = "Częstość występowania szerokości płatków sepal",
     xlab = "Długość płatku",
     ylab = "Częstość występowania",
     col = rainbow(liczba_unikalnych))
```

Z poniższego histogramu można odczytać szerokość płatku `sepal` wynosząca 3.0 występuje w 25 instancjach zbioru danych `iris`.

Wykorzystanie pakietu `ggplot2`:

```
> library(ggplot2)
> histogram <- ggplot(data=iris, aes(x=Sepal.Width))
> histogram + geom_histogram(binwidth=0.2, color="black",
  aes(fill=Species)) + xlab("Sepal Width") +
ylab("Frequency") + ggtitle("Histogram of Sepal Width")
```

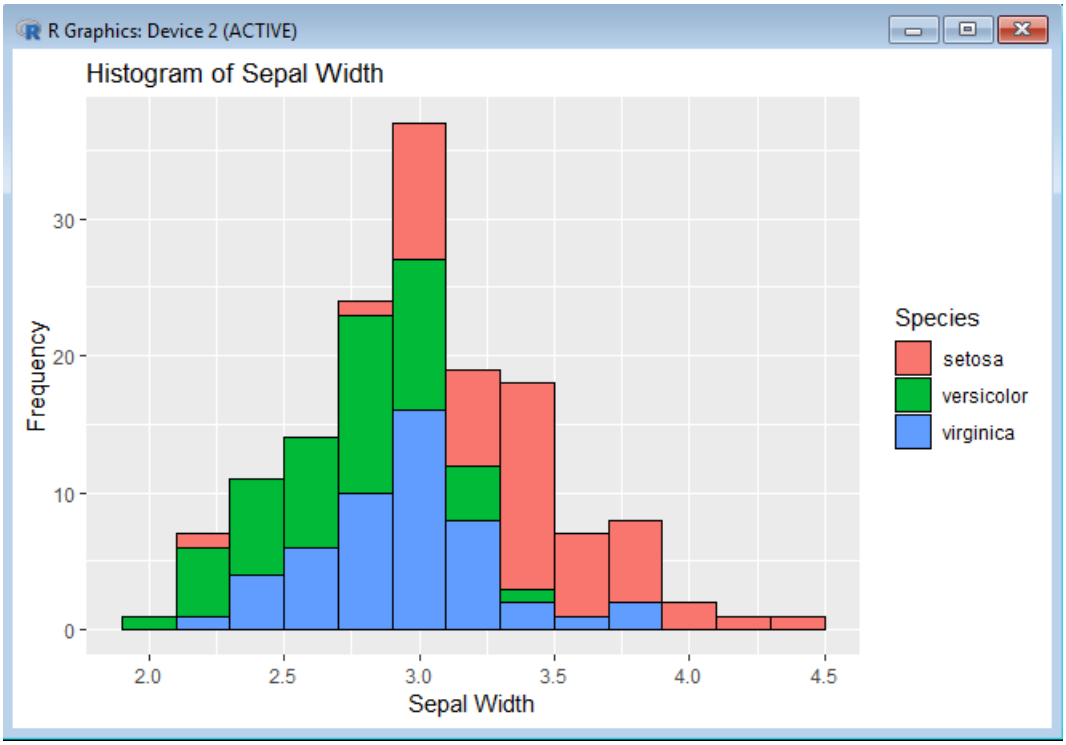


Do rysowania **wykresu kołowego** służy funkcja `pie`.

```
> percentage <- table(iris&Species) / length(iris&Species)
>             # procentowy udział każdego z gatunków w zbiorze danych
> lbls <- c("setosa","versicolor","virginica")  #  etykiety
> pie(percentage,labels = lbls, col=rainbow(length(lbls)),
  main="Pie Chart of Species")
> legend("topright", c("setosa","versicolor","virginica"),
  cex = 0.8, fill = rainbow(length(lbls)))
```

Największym problemem pokazanego wykresu kołowego jest próba rozróżnienia liczebności gatunków `setosa` i `versicolor`. Różnica procentowa pomiędzy nimi jest ciężka do zaobserwowania. Problem można ominąć używając funkcji `ggplot`.

```
> quan <- as.vector(table(iris$Species))
> pos <- cumsum(quan) - quan/2
> quantity <- data.frame(Species=c("setosa", "versicolor", "virginica"),
```

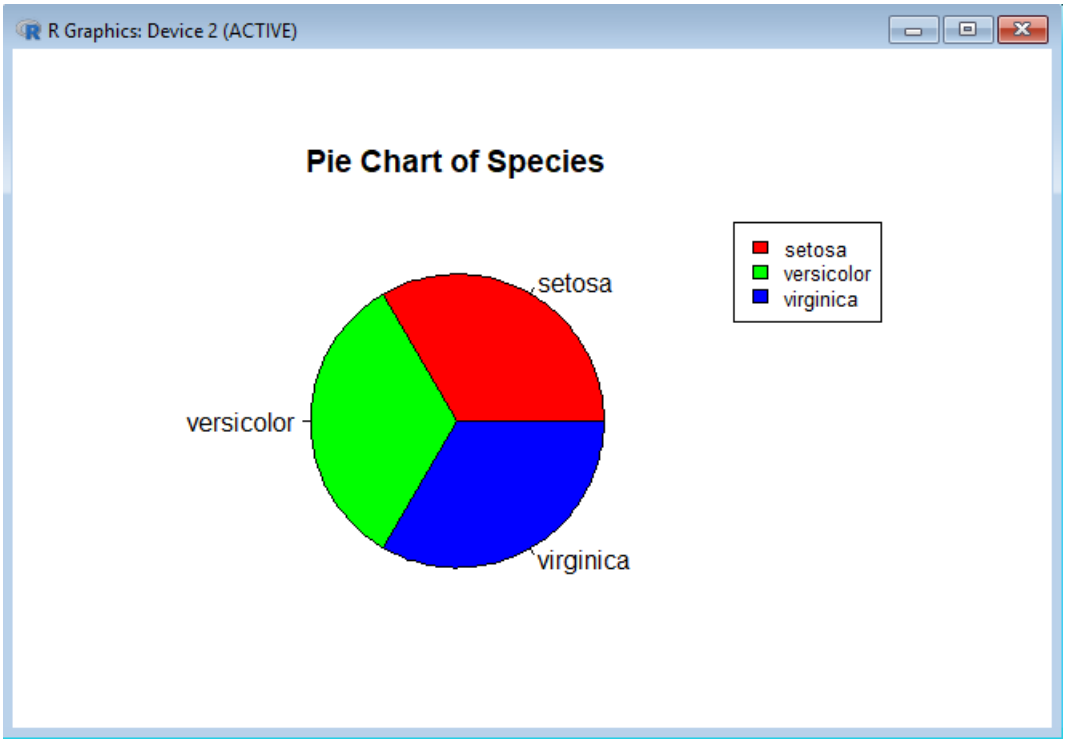


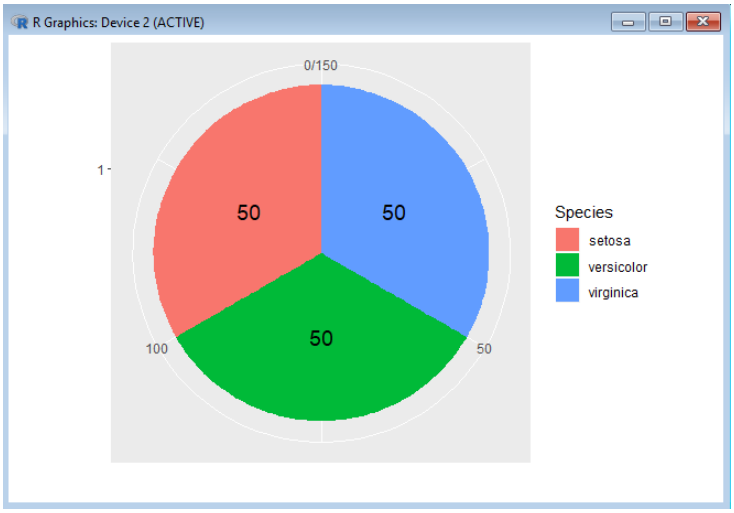

```
  quantity=quan, position = pos)
> pie <- ggplot(iris, aes(x=factor(1), fill=Species)) + geom_bar(width=1) +
  geom_text(data=quantity, aes(x=factor(1), y=position, label=quantity),
size=5) + labs(x="", y="")
> pie + coord_polar(theta="y")
```

Wykres pudełkowy jest realizowany za pomocą funkcji `boxplot` i wizualizuje 5 podstawowych statystyk dla rozkładu zmiennej liczbowej: wartość minimalną, pierwszy kwartyl (kwantyl rzędu 0.25), medianę, drugi kwartyl (kwantyl rzędu 0.75) oraz wartość maksymalną.

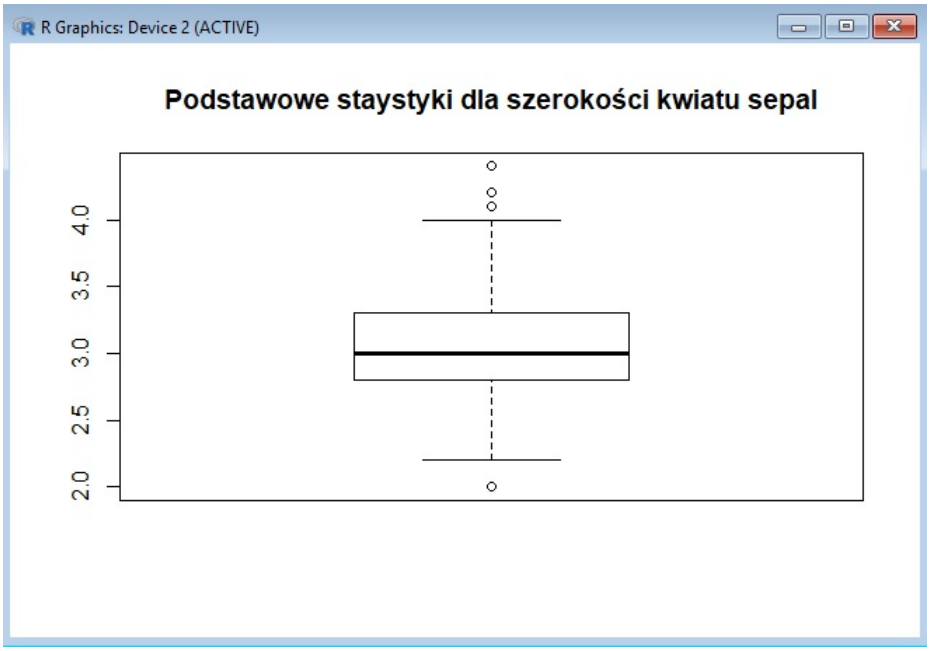
```
> boxplot(iris$Sepal.Width, main = "Boxplot dla szerokości kwiatu sepal")
```

Wykres słupkowy umożliwia wizualizacje wartości przy pomocy wysokości słupka. Posłuży nam do zaprezentowania dokładnej liczebności poszczególnych gatunków irysów. Do wyliczenia częstości występowania poszczególnych cech grupujących posłuży nam funkcja table().

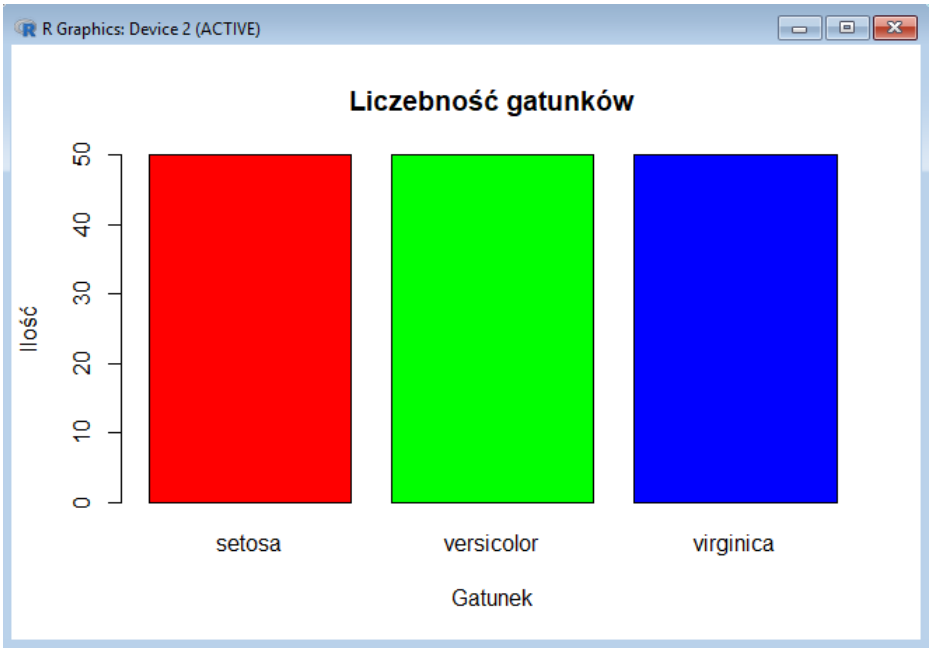





```
# liczebność każdego z gatunków
count <- table(iris$Species)
# generujemy trzy kolory
colors <- rainbow(3)
# rysujemy wykres
barplot(count,
```



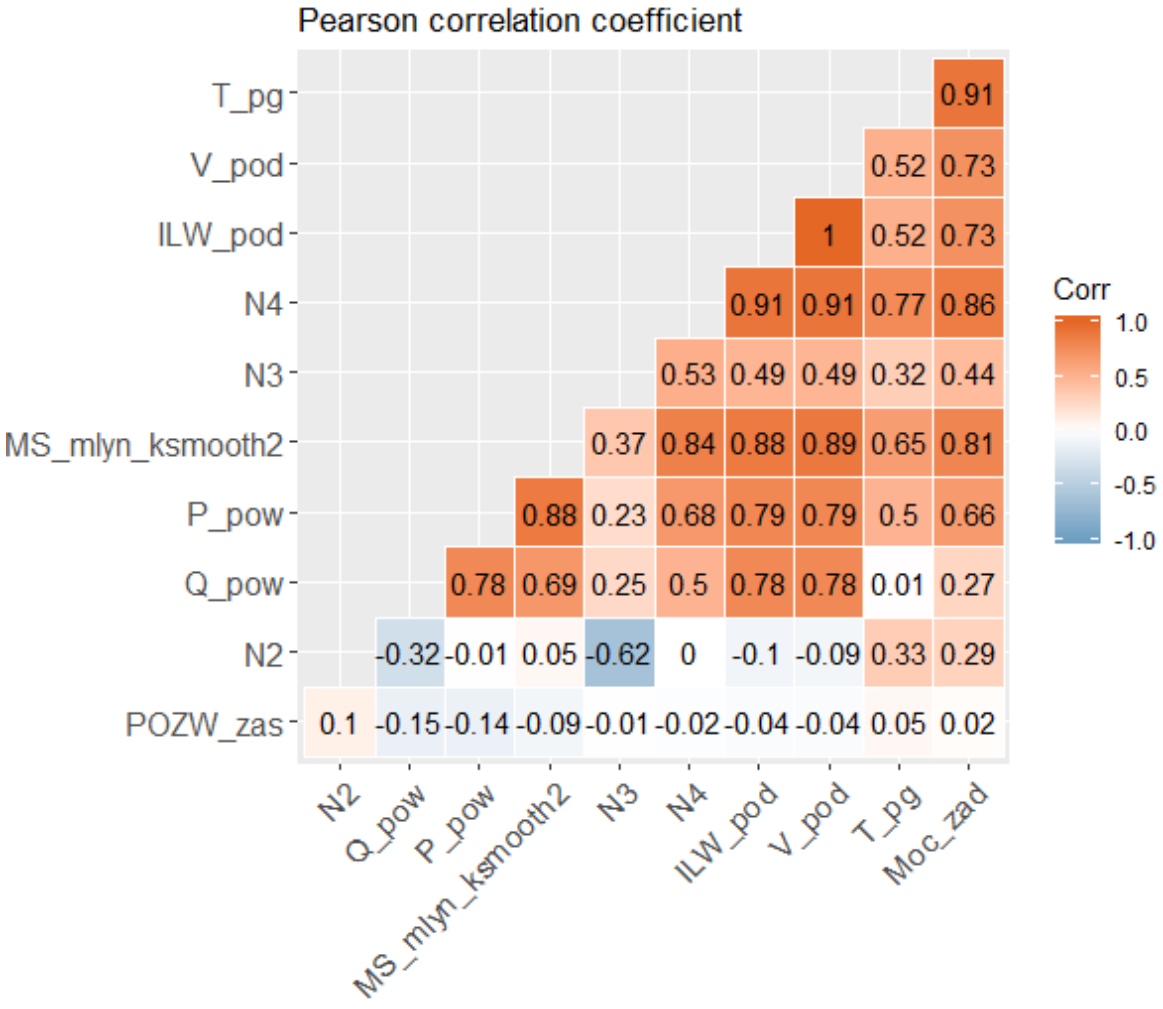

```
        main = "Liczebność gatunków",
        ylim = c(0, 50),
        xlab = "Gatunek",
        ylab = "Ilość",
        col = colors)
```



Można zapisać wygenerowany wykres:

```
> ggsave(filename = "plik.png", width = 10, height = 10)
```

Celem analizy korelacji jest stwierdzenie, czy między badanymi zmiennymi zachodzą jakieś zależności, jaka jest ich siła, jaka jest ich postać i kierunek.



IX. R DO REALIZACJI PROJEKTÓW

A. Analiza danych

```
> install.packages(''pakiet'')  #  instalowanie wybranego pakietu
> library(pakiet)  # ladowanie wybranego pakietu
> moje.dane<-read.table("zbior.txt", header= TRUE, sep="\t") #  wczytanie
> attach(moje.dane) # uzyskanie bezposredniego dostępu do
                    # zmiennych w tym zbiorze
```

- Struktura zbioru

```
> print(moje.dane) # wyswietlenie zbioru
> names(moje.dane) # zmienne w zbiorze moje.dane
> str(moje.dane) # struktura zbioru moje.dane
> levels(moje.dane[[1]])  # przedzialy pierwszej zmiennej
> head(moje.dane, n = 10) # pierwsze 10 wierszy
> tail(moje.dane, n = 5) # ostatnie 5 wierszy
> dim(moje.dane)  # wymiar ramki danych
```

- Statystyki opisowe

```
> cor(x, use =, method =) # korelacja; x to ramka danych
                 # use okresla metode traktowania danych brakujacych:
# all.obs zaklada brak danych niekompletnych
                 # complete.obs pomija dane brakujace,
 # method okresla typ korelacji: pearson, spearman lub kendall
> cor(x,y) # korelacja x i y
> mean(x)  # srednia
> median(x)  # mediana
> quantile(x)  # kwantyle
```

```
> sd(x) # ddchylenie standardowe dla 1 zmiennej:
> var(x) # wariancja dla 1 zmiennej
```

- Brakujące dane

```
> is.na(x) #  zwraca TRUE gdy brak jest wartosci x
> mean(x, na.rm = TRUE)  # srednia liczona po pominieciu
> nowe.dane <- na.omit(moje.dane) #  zwraca kompletne dane
```

- Miary kształtu:

```
> skewness(wektor)
> kurtosis(wektor)
```

- Graficzna reprezentacja danych ilościowych:

```
> boxplot(wektor,range=1.5,horizontal=FALSE) # wykres skrzynkowy
> hist(wektor,freq=TRUE) # histogram licznosci
> hist(wektor,freq=FALSE) # histogram czestosci
```

- Graficzna reprezentacja danych jakościowych:

```
> barplot(licznosci,col=c(''green'',...,''blue'') # wykres slupkowy
> pie(licznosci,col=c(''blue'',...,''red'') # wykres kolowy
```

B. Regresja liniowa

- Dopasowanie modelu:

```
> model=lm(y~ x1+x2+...+xN,data=zbior) # dopasowanie modelu


> summary(model) # statystyki opisowe
> abline(model)  # linia regresji
```

```
> model.nowy=lm(y~ x1+x2+...+xN,data=zbior,
+ subset=(x!=" ")) # dopasowanie modelu po usunięciu wybranego atrybutu
> coef(model)[1]  # wyraz wolny
> coef(model)[2]  # wspolczynnik kierunkowy dla
                          # modelu z 2 zmiennymi objasniajacymi
> summary(model)$sigma  # nieobciazony estymator odchylenia std.
> fitted(model)  # wartosci prognozowane
```

- Wskaźniki dopasowania modelu, residua, obserwacje nietypowe:

```
> residuals(model)  #  wartosci resztowe
> summary(model)$r.squared  # wspolczynnik determinacji
> summary(model)$adj.r.squared # skorygowany
                                      # wspolczynnik determinacji
> hatvalues(model)   # wartosci wplywowe
> rstandard(model)   # residua standaryzowane
>rstudent(model) # residua studentyzowane
> cooks.distance(model)  # odleglosci Cooke'a
> vif(model)       # wspolczynnik podbicia wartiancji
> dffits(model) #  wartosci DFFITS
> dfbeta(model)  # wartosci DFBETAS
> inf<- influence.measures(model); inf; summary(inf)
     # wskazanie wszystkich obserwacji wpływowych
> outlier.test(model) # test do wykrywania obserwacji odstajacych
                      # wymaga biblioteki \texttt{car}
```

- Badanie istotności niektórych zmiennych w modelu:

```
> model.mniejszy=lm(y~ x1+x2+...+xP, data = ramka danych)
> model.wiekszy=lm(y~ x1+x2+...+xQ, data = ramka danych)
> anova(model.mniejszy, model.wiekszy)
```

- Diagnostyka dopasowania modelu:

  Funkcja `plot` wykonuje 4 wykresy diagnostyczne: wykres residuów w funkcji prognoz, wykres kwantylowy dla residuów standardyzowanych r_i, wykres $\sqrt{r_i}$ w funkcji prognoz oraz wykres standaryzowanych residuów r_i w zależności od wpływów h_{ii}.

```
par(mfrow= c(2, 2))  #   4 wykresy w jednym oknie
> plot(model)
> qqnorm(model$res); qqline(model$res)  # wykres normalny dla reszt
> cutoff <- 4/((nrow(model)-length(model$coefficients)-2))
# jezeli bysmy chcieli aby punktem granicznym byl poziom, gdy
# odleglosc jest wieksza niz wartosc 4/(n-k-1);
# przypisujemy do zmiennej cutoff poziom
> plot(model, which=4, cook.levels=cutoff) # wykres odleglosci Cook'a
> influencePlot(model, main="Influence Plot",sub="Circle size is
  proportial to Cook's Distance") # wykres dla samych danych wplywowych
```

  Funkcja `predict` służy do wyestymowania średniej oczekiwanej wartości zmiennej objaśnianej dla konkretnych wartości zmiennych objaśniających `x1` i `x2` wraz z 95-procentowym przedziałem ufności:

```
> predict(model,data.frame(x1=,x2=),interval="confidence",level=0.95)
                 # predykcja z 95-procentowym przedziałem ufności
```

C. Regresja logistyczna

- Wyznaczanie współczynników modelu regresji logistycznej:
  - dla danych niepogrupowanych (y to zmienna typu `factor`, pierwszy (zgodnie z porządkiem alfabetycznym) poziom uznawany jest za porażkę, wszystkie pozostałe - za sukces)

```
> model=glm(y~x1+x2,family=binomial(link="logit"))
```

  - dla danych pogrupowanych

```
> y=cbind(lp.sukcesow,lp.porazek)
> model=glm(y~x1+x2,family=binomial(link="logit"))
```

- Diagnostyka modelu:

```
> residuals.glm(nazwa.modelu,type="response")  # residua odpowiedzi
> residuals.glm(nazwa.modelu,type="pearson")  # residua Pearsonowskie
> residuals.glm(nazwa.modelu,type="deviance")  # residua odchyleniowe
```

D. Analiza skupień

Wykorzystanie metod analizy skupień w środowisku `R` jest możliwe dzięki użyciu następujących pakietów: standardowego pakietu `stats`, pakietu `cluster` oraz dodatkowo pakietów `flexclust` i `mclust02`.

Wywołanie algorytmu k-średnich gdy znamy optymalną liczbę skupień jest dość proste. Realizacja grupowania przy użyciu funkcji `kmeans` może być następująca:

```
> klaster = kmeans(dane,3)
> plot(dane, pch=klaster$cluster)
  # lub
>klaster1 = kmeans(dist(dane),3,20)
> plot(dane,pch=19,col=cl$cluster,main="k-means")
```

Ogólna formuła tej funkcji ma postać:

```
> kmeans(x, centers, iter.max = 10, nstart = 1,
      algorithm = c("Hartigan-Wong", "Lloyd", "Forgy",
      "MacQueen"))
```

- `x` to macierz z danymi podlegającymi grupowaniu
- `centers` to albo liczba skupień, które chcemy utworzyć, albo podane początkowe centra skupień; jeśli jest to liczba skupień, wówczas procedura wyboru potencjalnych centrów skupień odbywa się w sposób losowy
- `iter.max` to maksymalna liczba iteracji
- `nstart` to liczba losowych zbiorów branych pod uwagę w grupowaniu (jeśli w `center` podano liczbę grup)
- `algorithm` określa, który algorytm będzie wykonany spośród dostępnych: Hartigan and Wong (domyślny), MacQueen, Lloyd czy Forgy.

Odpowiednio manipulując tymi parametrami można optymalizować budowane skupienia obiektów w danym zbiorze. Zauważmy, że dane `x` nie mogę posiadać obserwacji z brakującymi danymi, w przeciwnym razie funkcja może nie zadziałać, ponieważ używamy miary odległości.

Grupowanie realizowane jest także poprzez funkcję `mclust` z pakietu o tej samej nazwie. W podstawowej wersji wywołania metody nie podaje się liczby skupień, a jedynie zbiór danych, które chcemy pogrupować:

```
> klast<-Mclust(dane)
> skupienia<-klast$classification
> plot(dane,pch=19,col=skupienia,main="Mclust")
```

Utworzony w wyniku wykres będzie bardzo podobny do tego utworzonego przez metodę `kmeans`. W pakiecie `stats` dostępna jest funkcja `cutree`, która pozwala klasyfikować obiekty do jednej z utworzonych grup, pozwalając jednocześnie na sterowanie nie tylko liczbą tworzonych skupień, ale i poziomem odcięcia w tworzonym drzewie.

```
cutree(tree, k = NULL, h = NULL)
```

gdzie odpowiednio:

- `tree` jest rezultatem wywołania funkcji `hclust`
- `k` to liczba skupień
- `h` to poziom odcięcia drzewa (`tree`)

```
> hklast <- hclust(dist(dane))
> cutree(hklast, k=3)
```

Algorytm k-średnich cechuje się licznymi wadami, sprawia, że chętniej używanym jest algorytm np. k-medoidów. W środowisku `R`, w ramach pakietu `cluster`, dostępna jest funkcja `pam` realizująca algorytm o nazwie PAM (ang. *Partitioning Around Medoid*). Przykładem jej wywołania jest następująca komenda:

```
> klast <- pam(dane,3)
> sil <- silhouette(kluster)
> summary(sil)
> plot(sil, col = c("red", "blue", "green"))
```

E. Drzewa decyzyjne

Podstawowe pakiety służące do budowy modeli drzew klasyfikacyjnych i regresyjnych dostępnych w programie R to `rpart`, `party`, `maptree`, `randomForest`.

Podstawowe funkcje, które uruchamiają procedury budowy modeli drzew klasyfikacyjnych i regresyjnych:

```
> tree(formula, data, subset, na.action=na.pass,
       control=tree.control(nobs), method, split=c("gini"))
> rpart(formula, data, subset, na.action=na.pass, method,
       split=c("information","gini"), control)
> ctree(formula, data, subset, controls)
        tree.control(nobs, mincuit=5, minsize=10)
> rpart.control(minsplit=20, minbucket=round(minsplit/3),
                cp=0.01, maxdepth=30)
> ctree_control(mincriterion = 0.95, minsplit = 20,
                minbucket = 7, maxdepth = 30)
```

gdzie

- `formula` - symboliczny opis modelu dyskryminacyjnego i regresyjnego
- `data` - macierz danych, która uwzględnia zmienne modelu
- `subset` - funkcja wskazująca podzbiór obserwacji do klasyfikacji
- `na.action` - wyrażenie mające na celu wskazywanie sposobu postępowania w przypadku braków obserwacji
- `control` - parametry sterujące procedurą budowy drzewa
- `method` - metoda budowy drzewa, jedyna domyślna wartość to podział rekurencyjny

- `split` - określenie miary różnorodności (domyślnie `"Gini"` - indeks różnorodności Giniego)
- `nobs` - liczba obserwacji w zbiorze uczącym
- `mincriterion` - wartości statystyki lub 1 - p-wartość
- `mincut` - minimalna liczba obserwacji w węźle, który powstał w wyniku podziału (domyślnie 5)
- `minsize` - minimalna liczba obserwacji w węźle, który ulega podziałowi (domyślnie 10)
- `minsplit` - minimalna liczba obserwacji w węźle, który ulega podziałowi (domyślnie 20)
- `minbucket` - minimalna liczba obserwacji w liściu drzewa (domyślnie minsplit/3)
- `cp` - zmiana wartości poprawy modelu, jeżeli podział nie poprawia o więcej niż `cp` to nie jest wykonywany (domyślnie 0.01; wartość 0 zbuduje kompletne drzewo o maksymalnej głębokości, uwzględniając wartości parametrów `minsplit` i `minbucket`)
- `maxdepth` - maksymalna głębokość drzewa (domyślnie 30)

Kolejno w wierszu odczytujemy: numer węzła, nazwę atrybutu oraz wartość w której wykonany został podział, liczbę obserwacji w węźle, liczbę obserwacji o wartości błędnie zaklasyfikowanych, klasę do jakiej został zaklasyfikowany węzeł oraz prawdopodobieństwa a posteriori dla klas. Symbol `*` informuje o tym czy dany węzeł jest liściem.

Kolejnym elementem jest wyświetlanie drzewa. Aby to uczynić możemy skorzystać najpierw z funkcji `X11()`. W ten sposób zostało przygotowane nowe okno do wyświetlenia drzewa. Polecenie `plot` tworzy sam rysunek drzewa (dodatkowo budując drzewo za pomocą funkcji `rpart` otrzymamy drzewo z krawędziami o długości proporcjonalnej do stopnia zróżnicowania obserwacji w węzłach), natomiast polecenie `text` dodaje napisy w węzłach drzewa.

```
Summary of the Decision Tree model for Classification (built using 'rpart'):

n= 256

node), split, n, loss, yval, (yprob)
      * denotes terminal node

1) root 256 41 No (0.83984375 0.16015625)
  2) Pressure3pm>=1011.9 204 16 No (0.92156863 0.07843137)
    4) Cloud3pm< 7.5 195 10 No (0.94871795 0.05128205) *
    5) Cloud3pm>=7.5 9  3 Yes (0.33333333 0.66666667) *
  3) Pressure3pm< 1011.9 52 25 No (0.51923077 0.48076923)
    6) Sunshine>=8.85 25  5 No (0.80000000 0.20000000) *
    7) Sunshine< 8.85 27  7 Yes (0.25925926 0.74074074) *

Classification tree:
rpart(formula = RainTomorrow ~ ., data = crs$dataset[crs$train,
    c(crs$input, crs$target)], method = "class", parms = list(split = "information"),
    control = rpart.control(usesurrogate = 0, maxsurrogate = 0))

Variables actually used in tree construction:
[1] Cloud3pm    Pressure3pm Sunshine

Root node error: 41/256 = 0.16016

n= 256

        CP nsplit rel error  xerror    xstd
1 0.158537      0   1.00000 1.00000 0.14312
2 0.073171      2   0.68293 0.80488 0.13077
3 0.010000      3   0.60976 0.97561 0.14169

Time taken: 0.11 secs
```

```
> X11()
> plot(drzewo)
> text(drzewo,use.n=TRUE,all=FALSE,cex=0.7)
```

gdzie

- `use.n=TRUE` podaje liczebności klas w liściach
- `all=TRUE` podaje klasę w węzłach
- `cex` rozmiar czcionki

Inny wykres uzyskam korzystając z funkcji `draw.tree()` z pakietu `maptree`. Można również użyć funkcji `ctree()` w celu lepszego graficznego przedstawienia wyników .

```
> drzewoA <- draw.tree(drzewo,cex=0.7,nodeinfo=TRUE)
> drzewoB <- ctree(y~x1+x2+x3+ x4+x5+x5, data=dane,
  controls=ctree_control(minsplit=2,minbucket = 1,
  maxdepth = 10, mincriterion = 0.50))
> plot(drzewoB)
```

Parametr `nodeinfo` odpowiada za opis węzła. Dzięki niemu można uzyskać dodatkowe informacje na temat ilości elementów poprawnie zaklasyfikowanych, podanej w procentach, czyli także prawdopodobieństwa a posteriori dla klas.

Uzyskane modele klasyfikacyjne oraz regresyjne można zastosować do predykcji danych:

```
> predict(tree, newdata, type=c("vector","prob","class","matrix"),
  na.action=na.pass)
```

gdzie

- `tree` - obiekt w postaci drzewa
- `newdata` - zbiór rozpoznawany tzn. zbiór, który będzie podlegać klasyfikacji
- `type` - forma w jakiej mają być podane wyniki, do wyboru mamy:
  - `"vector"` - do każdej z obserwacji przyporządkowany jest numer klasy
  - `"prob"` - prawdopodobieństwo a posteriori dla każdej z klas
  - `ćlass"` - nazwa klasy w której znalazła się dana obserwacja
  - `"matrix"` - wynikiem jest macierz w której kolejnych kolemnach uzyskujemy numer klasy, liczebność klasy oraz prawdopodobieństwo a posteriori
  - `na.action` - wyrażenie mające na celu wskazywanie sposobu postępowanie w przypadku braków wartości zmiennych w zbiorze `newdata`

```
> predict(drzewo,dane.testowe,type="vector")
```

Korzystając z parametru `class` zamiast `vector` w dolnym wierszu otrzymałabym nazwę klasy - np. ”tak” lub ”nie”. Wyniki można przedstawić w postaci tabelki.

```
> tablica <- table(predict(drzewo,dane.testowe,type="class"),
  dane.testowe\$y)
> tablica
    nie tak
nie  11  2
tak   6  1
```

Wynikiem jest tabelka. Wiersz u góry oznacza, w której klasie była obserwacja, natomiast kolumna po lewej oznacza, do której klasy przydzielił ją klasyfikator. Można również zliczyć ilość błędów predykcji za pomocą

```
> sum(tablica)-sum(diag(tablica))
```

W wyniku otrzymuję 8. Oznacza to, że na 20 elementów, które należało przyporządkować, źle przyporządkowanych zostało 8.

X. 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.