Statystyka

(dla informatyków)

Prof. UAM dr hab. Maciej Łuczak
Zakład Statystyki Matematycznej i Analizy Danych
Wydział Matematyki i Informatyki UAM

Literatura

- Biecek P. (2017), Przewodnik po pakiecie R, GiS.
- Crawley M.J. (2012), The R Book, Wiley.
- Ćwik J., Koronacki J. (2009), Statystyczne systemy uczące się. Ćwiczenia w oparciu o pakiet R, Oficyna Wydawnicza Politechniki Warszawskiej.
- Gągolewski, M. (2014), Programowanie w języku R, PWN.
- Górecki T., (2011), Podstawy statystyki z przykładami w R, BTC.
- Krzyśko M., Wołyński W., Górecki T., Skorzybut M. (2008), Systemy uczące się, WNT.

Język R

- Program R jest zaawansowanym pakietem statystycznym i językiem programowania istniejącym na platformy Windows, Linux oraz MacOS. Objęty jest licencją GNU GPL
- Pierwsza wersja R (początek lat 90) została napisana przez Roberta Gentlemana i Ross Ihake pracujących na Wydziale Statystyki Uniwersytetu w Auckland. Obecnie rozwojem R kieruje fundacja „The R Foundation for Statistical Computing”
- Język R był wzorowany na języku S opracowanym w AT&T Bell Laboratories i stosowanym w programie S-PLUS
- Największą siłą R jest kilkadziesiąt tysięcy bibliotek funkcji napisanych przez tysiące osób z całego świata, przeznaczonych do najróżniejszych zastosowań. Każda biblioteka dostarczana jest z pełną dokumentacją

Język R - własności

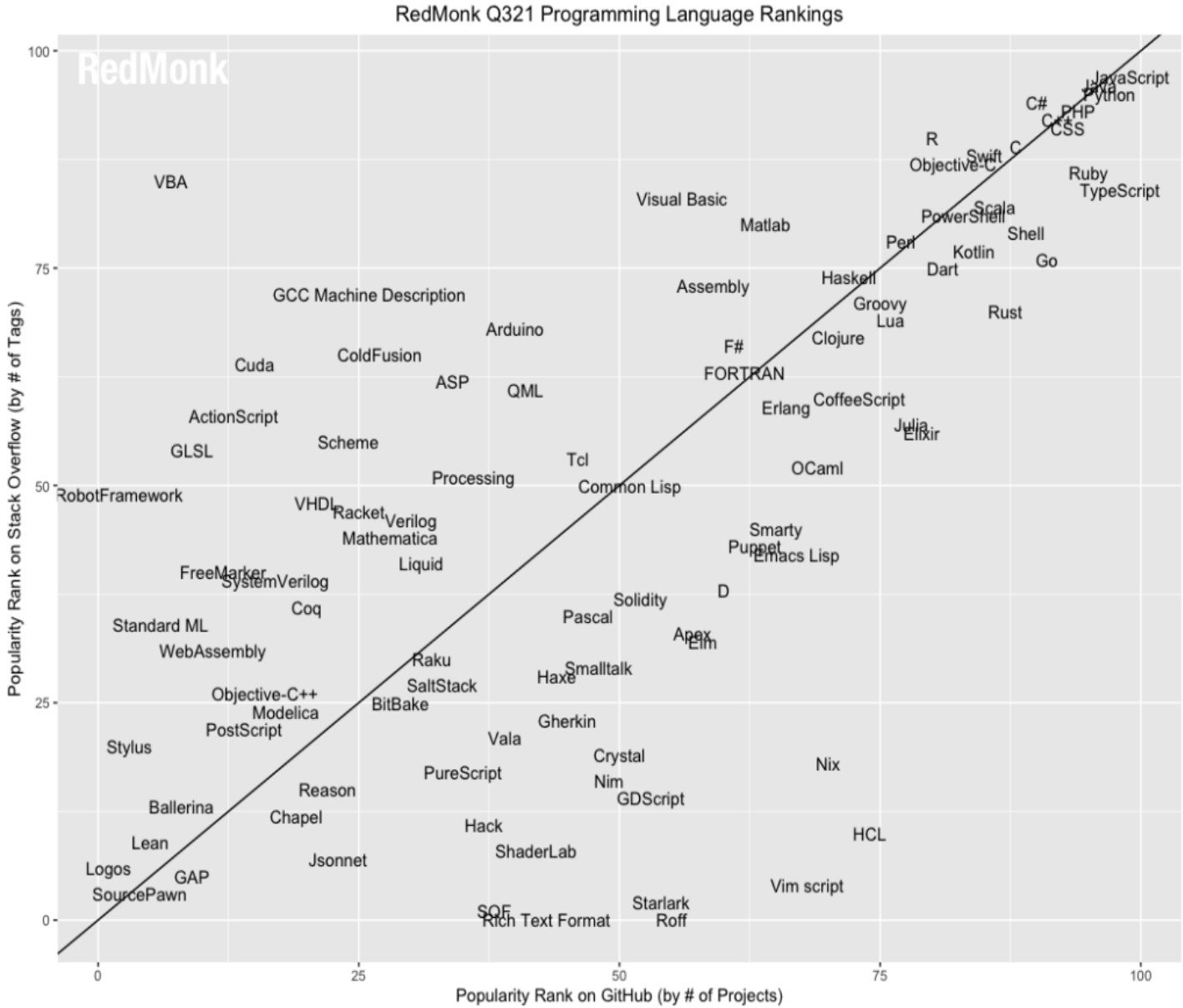
- Otwarta licencja
- Porównywalny, często lepszy niż rozwiązania komercyjne
- Dostępny na Windows, Linux, MacOS
- Uniwersalny język programowania
- Język obiektowy i funkcyjny
- Olbrzymie możliwości wizualizacji danych
- Łatwy do opanowania nie tylko dla informatyków

Język R - zalety

- Język skryptowy, dynamicznie typowany
- Otwarte źródła
- Multiplatformowość
- Zaawansowany język statystyczny
- Olbrzymie możliwości wizualizacji danych
- Podobny do innych języków
- Olbrzymia społeczność użytkowników
- Olbrzymia ilość pakietów i bibliotek
- Łatwość opanowania
- Zwarty i przejrzysty kod

Język R - wady

- Język skryptowy, dynamicznie typowany
- Wolny (bardzo wolny), niska prędkość wykonywania kodu
- Udziwnienia składni
- Utrudniona współpraca z kodem produkcyjnym
- Czasami niska jakość pakietów


RedMonk Q321 Programming Language Rankings
RedMonk
Popularity Rank on Stack Overflow (by # of Tags)
Popularity Rank on GitHub (by # of Projects)
0
25
50
75
100
JavaScript
Python
C#
PHP
CSS
R
Swift
C
Objective-C
Ruby
TypeScript
Scala
PowerShell
Shell
Perl
Kotlin
Go
Dart
Haskell
Groovy
Lua
Rust
Clojure
F#
FORTRAN
CoffeeScript
Erlang
Julia
Elixir
OCaml
Tcl
Common Lisp
Smarty
Puppet
Emacs Lisp
D
Solidity
Pascal
Apex
Elm
Smalltalk
Haxe
Raku
SaltStack
BitBake
Gherkin
Vala
Crystal
Nix
Nim
GDScript
PureScript
Reason
Chapel
Hack
ShaderLab
HCL
Vim script
Starlark
Roff
Rich Text Format
SQF
Jsonnet
Lean
Logos
GAP
SourcePawn
Ballerina
Stylus
PostScript
Modelica
Objective-C++
Liquid
Racket
Verilog
Mathematica
VHDL
RobotFramework
Processing
Scheme
GLSL
ActionScript
Cuda
ColdFusion
ASP
QML
Arduino
GCC Machine Description
Assembly
Visual Basic
Matlab
VBA
FreeMarker
SystemVerilog
Coq
Standard ML
WebAssembly

Język R - podstawy

- Klasycznym operatorem przypisania jest <−, można również wykorzystywać znak = oraz −>
```
a <- 10
a = 10
10 -> a
```
- Jeżeli chcemy, aby wynik przypisania został wyświetlony na ekranie, należy przypisanie zamknąć w nawiasy ()
```
(a = 10)
```
- Jeśli chcemy, aby kilka wyrażeń było zapisanych w jednej linii, to musimy oddzielić je średnikiem
```
a = 10; a = a + 1
```

R - podstawy

- Komentarz poprzedzamy znakiem hash `#`, wszystko do końca linii jest już komentarzem
```
a = 10 # przypisanie
```
- R odróżnia wielkie i małe litery.
`a` i `A` to dwie różne nazwy
- Znak `.` (kropka) nie jest znakiem zastrzeżonym, można go używać w nazwach (nie może być pierwszym znakiem nazwy)
```
poprawna.nazwa
.niepoprawna
```
- W celu określenia kolejności działań używamy nawiasów okrągłych `()`
```
b = (a + 2) * (a - 2)
```

R - podstawy

- Do grupowania wyrażeń używamy nawiasów klamrowych { }

```
{a = 10; a = a + 1}

{a = 10
 a = a + 1}

{
 a = 10
 a = a + 1
}
```

R – proste typy danych

- Logiczny o wartościach `TRUE`, `FALSE` lub `T`, `F`
- Numeryczny (double, integer)
```
a = 10.5
b = 3; c = 3L
```
- Zespolony
```
z = 2 + 3i
```
- Znakowy – napisy, łańcuchy znaków. Powinny być zawarte między znakami ’ lub ”. Można używać znaków sterujących (`\n`, `\t`)
- Funkcje odczytujące typ i strukturę danych to: mode i str
```
mode(z) == "complex"
```

R – operatory arytmetyczne i logiczne

- Standardowe operatory arytmetyczne

```
+ - * /
2 + 2 * 2
```

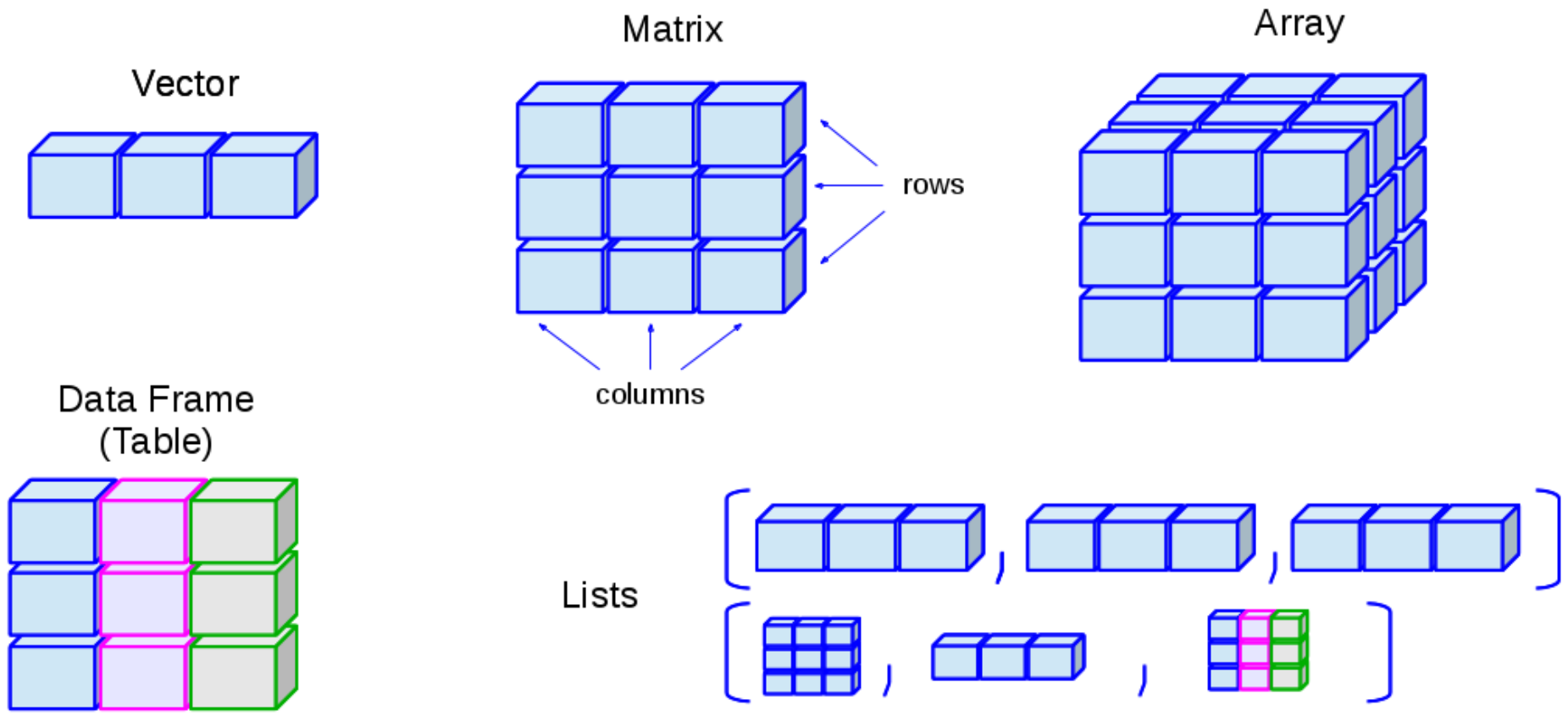
- Standardowe operatory logiczne

```
== > < >= <=
&& || ! & |
```

- Operatory arytmetyczne mają wyższy priorytet niż logiczne

```
a + 1 < 2 * b
a < 1 && b < 1
```

R – złożone typy danych



R – złożone typy danych - wektor

Wektor jest to (skończony) ciąg elementów tego samego typu.

- Tworzenie wektora
  ```
  c(1, 2, 3)
  c(1.2, 3.14, 1, -2.5)
  ```
- Operator `:` (dwukropek) generuje liczby z zakresu
  ```
  2:8 == c(2, 3, 4, 5, 6, 7, 8)
  ```
- Funkcja `seq` generuje liczby z podanego przedziału, przy czym można podać krok (by) i zakres (from, to)
  ```
  seq(from = 2, to = 11, by = 3) == c(2, 5, 8, 11)
  ```
- `rep` – generuje ciąg składający się z powtórzeń innego ciągu
  ```
  rep(10, 3) == c(10, 10, 10)
  ```

R – złożone typy danych - wektor

- Indeksowanie wektora (operator `[]`). Indeksy zaczynają się od `1`

```
W = c(1, 3, 5, 7)
W[2] == 3
W[c(2, 3)] == c(3, 5)
W[2:4] == c(3, 5, 7)

V = W[-c(2, 3)]
V == c(1, 7)

V = W[W < 4]
V == c(1, 3)
```

R – złożone typy danych - czynnik

- Czynnik (typ wyliczeniowy) jest to właściwie wektor indeksów (liczb całkowitych) wskazujących na dany poziom czynnika
- Tworzenie czynnika – funkcja `factor`

```
wektor = c("1", "2", "2", "3", "1", "1", "1")
czynnik = factor(wektor)
```

- Funkcja `levels` zwraca poziomy czynnika, a funkcja `nlevels` liczbę tych poziomów

```
levels(czynnik) == c("1", "2", "3")
nlevels(czynnik) == 3
```

R – złożone typy danych - macierz

- ```
  M = matrix(1:9, 3, 3)
  M == 1 4 7
       2 5 8
       3 6 9

  M[3, 2] == 6
  M[1:2, 2:3] == 4 7
                 5 8
  M[c(1, 3),] == 1 4 7
                 3 6 9
  M[, 1:2] == 1 4
              2 5
              3 6
  ```
- ```
  A = array(1:12, dim = c(2, 3, 2))
  ```

R – złożone typy danych - lista

- Lista jest to ciąg elementów o dowolnych typach
```
L = list(1, "abc", c(1,2,3))
```
- Elementy listy mogą być nazwane
```
L – list(imie = "Adam", nazwisko = "Nowak",
wiek = 25)
```
- Do elementów listy można odwoływać się poprzez ich nazwy korzystając z operatora `$` (dolar) lub indeksowania operatorem `[[]]`
```
L$nazwisko == "Nowak"
L[[3]] == 25
```

R – złożone typy danych – ramka danych

- Ramka danych jest to typ danych zawierający dane tabelaryczne
- Technicznie jest to po prostu lista wektorów, gdzie każdy wektor przedstawia jedną z cech danych
- Do ramki danych możemy odwoływać się przez nazwy wektorów cech (kolumn) lub jak do macierzy (operator `[,]`)

```
D = data.frame(cecha1 = c(1,2,3,4),
               cecha2 = c(0,3,6,9))
D$cecha2[3] == 6
D[2, 3] == 6
```

R – instrukcje warunkowe

- Instrukcja warunkowa `if`

```
if (wartość_logiczna)
{
  instrukcje
}
```

- Instrukcja warunkowa `if else`

```
if (wartość_logiczna)
{
  instrukcje_1
}
else
{
  instrukcje_2
}
```

R – instrukcje warunkowe

- Funkcja `ifelse`

```
ifelse(wart_logiczna, wart_1, wart_2)
ifelse(x < 0, "a", "b")
```

- Funkcja `switch`

```
switch(klucz, wart_1 = akcja_1,
              wart_2 = akcja_2, ...)
```

R - pętle

- Pętla `for`

```
for (iterator in kolekcja)
{
  blok_instrukcji
}
```

- Pętla `while`

```
while (wartość_logiczna)
{
  blok_instrukcji
}
```

- Pętla `repeat`

```
repeat
{
  blok_instrukcji
}
```

- Instrukcja `break` przerywa wykonywanie dowolnej pętli
- Instrukcja `next` powoduje przejście do następnej iteracji dowolnej pętli

R - funkcje

- Definicja funkcji (funkcja anonimowa)

```
function(lista_argumentow)
{
  blok_instrukcji
}
```

- Przypisanie funkcji do zmiennej

```
f = function(lista_argumentow)
{
  blok_instrukcji
}
```

R - funkcje

- Za wynik funkcji przyjmowana jest wartość wyznaczona w ostatniej linii ciała funkcji lub jako argument funkcji `return`

```
f = function(x, y)
{
  x = 2 * x
  y = 2 * y
  x + y
}
f = function(x, y)
{
  x = 2 * x
  y = 2 * y
  return(x + y)
}
```

R - funkcje

- Argumenty nazwane i domyślne

```
f = funkcion(x, y = 0)
{
  x + y
}
f(2, 3) == 6
f(2) == 2
f(2, y = 10) == 12
f(x = 10) == 10
f(y = 100, x = 200) == 300
```

R - funkcje

- Funkcje mogą być argumentami innych funkcji oraz wartościami zwracanymi z funkcji. W szczególności jako argument funkcji oraz wartość zwracaną można przekazać funkcję anonimową

```
zlozenie = function(f, g)
{
  function(x) f(g(x))
}

h = zlozenie(function(x) x^2, function(x) x+1)
h(10) == (10 + 1)^2
```

R

- RStudio – otoczenie programistyczne (IDE, RAD)
- Wczytywanie i używanie pakietów/bibliotek
  `install.packages, library`

```
install.packages("nazwa_pakietu", dependencies = TRUE)
library(pakiet1)
library(pakiet2)
```

- Wyświetlanie wartości wyrażeń `print, cat`

```
cat("a =", a)
```

- Możliwości graficzne, wizualizacja danych
  `plot, hist, boxplot, barplot, curve`
  `ggplot2`
- Pomoc
  `? ?? help args example`