Wprowadzenie do języka R

1) Konsola R i środowisko RStudio - pierwsze kroki

1. Zapoznajemy się z zawartością strony https://www.r-project.org
2. Uruchamiamy konsolę R (*R GUI* w syst. Windows) i w linii komend (w dolnej części okna, po znaku `>`) wpisujemy kolejno (i naciskamy `[ENTER]`):

```
 > R.version.string
 > getRversion()
 > help(help)
 > ?help
 > ?"+"
 > library(help = "stats")
 > 2+2
 > 2^5
 > FALSE == FALSE
 > 3 > 2
 > 2 + [naciskamy ENTER]
 + 2   [naciskamy ENTER] # proszę zwrócić uwagę na zmianę "znaku zachęty"
>
 > print("My name is Inigo Montoya. You killed my father. Prepare to die."
)
 > ?print
 > cat("My name is Inigo Montoya. You killed my father. Prepare to die.")
 > ?cat
 > print("My name is Inigo Montoya.", "You killed my father.", "Prepare to
 die.") # !!!
 > ca # naciskamy klawisz [TAB], wybieramy z listy "cat", otwieramy nawias
 i... kontynuujemy wg następnej linii :)
 > cat("My name is Inigo Montoya.", "You killed my father.", "Prepare to d
ie.")
 > q() # po naciśnięciu klawisza [ENTER] w oknie dialogowym wybieramy opcj
ę "nie zapisuj"
```

   Uwagi:

   - znak `>` nie jest częścią polecenia,
   - tekst umieszczony po znaku `#` jest komentarzem

3. Zapoznajemy się z zawartością strony https://www.rstudio.com
4. Uruchamiamy środowisko RStudio (np. menu start w syst. Windows) i analizujemy kolejne części okna głównego
5. W menu głównym (RStudio) wybieramy kolejno `Tools > Global Options...` i analizujemy możliwości konfiguracyjne środowiska (np. `Pane Layout`)
6. Wybieramy (klikamy w) grupę `Apperance` i ustawiamy wg preferencji odpowiednią wartość parametru `Editor theme` (np. *Material* lub *Solarized Dark*)

7. W prawym dolnym panelu wybieramy zakładkę `Help` , klikamy w ikonkę z domkiem :) i analizujemy zawartość zakładki (np. klikamy kolejno w: `An Introduction to R` , `The R Language Definition` , `Learning R Online`)
8. Sprawdzamy zawartość zakładek wszystkich paneli okna głównego
9. W menu `Tools` wybieramy `Install Packages...` , analizujemy zawartość okna; zamykamy okno klikając `Cancel`
10. W menu `Tools` wybieramy `Check for Package Updates...` i analizujemy zawartość okna; zamykamy okno klikając `Cancel`
11. Analizujemy zawartość menu `Session` , zwracamy szczególną uwagę na `Set Working Directory >` i `Clear Workspace...`
12. ***Zadania***:
    1. Powtórz kroki z punktu 1.2, tym razem wpisując polecenia w konsoli RStudio (lewy dolny róg)
    2. Zastanów się nad zaletami i wadami używania konsoli R i RStudio w codziennej pracy analityka. Czy w każdej sytuacji RStudio jest lepszym rozwiązaniem?

2) Pierwszy skrypt w środowisku RStudio

1. [*W RStudio*] w menu `File` wybieramy `New File > R Script`
2. W utworzonym pliku `Untitled1` wpisujemy:

```
print("Nie bądź bezpieczny")
print("poeta pamięta")
print("Możesz go zabić - narodzi się nowy")
print("Spisane będą czyny i rozmowy")

cat(" Nie bądź bezpieczny\n",
    "poeta pamięta\n",
    "Możesz go zabić - narodzi się nowy\n",
    "Spisane będą czyny i rozmowy")
```

3. Ustawiamy kursor w 1. linii (klikamy w obszarze 1. linii)
4. Naciskamy kilka razy przycisk `Run` (prawa strona "belki narzędziowej" edytora skryptów)
5. Naciskamy kilka razy przycisk znajdujący się po prawej stronie `Run`
6. Naciskamy przycisk `Source`
7. W konsoli RStudio wpisujemy

```
 > ?getwd  # analizujemy opis (w prawym panelu)
 > getwd() # sprawdzamy ustawiony katalog roboczy (w tym katalogu będą zap
isywane domyślnie wszystkie pliki)
```

8. W systemie plików tworzymy nowy katalog, np. o nazwie `stat-lab1` (`mkdir stat-lab1` w systemach uniksowych, a w Windows - wiadomo :)
9. Ustawiamy nowo utworzony katalog jako roboczy (menu `Session > Set Working Directory > Choose Directory` i wskazujemy nowo utworzony `stat-lab1`)
10. Sprawdzamy zawartość zakładki `Files` w prawym dolnym panelu (okna głównego)
11. Zapisujemy utworzony skrypt jako `s1.R` (menu `File > Save/Save As` lub przycisk z dyskietką w belce narzędziowej okna głównego lub edytora skryptów)
12. Zamykamy plik ze skryptem (menu `File > Close` lub klikamy w `x` w tytule zakładki)
13. ***Zadania***:
    1. Otwórz i ponownie uruchom skrypt `s1.R`
    2. Dopisz kilka linii w skrypcie `s1.R` , a następnie sprawdź jego działanie
    3. Dodaj kilka komentarzy w skrypcie `s1.R` (składnia: `#` tekst komentarza)
    4. Ustal, czy w języku R jest specjalna składnia dla komentarzy wieloliniowych

3) Tworzenie obiektów nazwanych w języku R (zmienne i przypisania)

1. W zakładce `help` (prawy dolny panel) w polu wyszukiwania wpisujemy `assignOps` i analizujemy opisy operatorów
2. W konsoli RStudio (lewy dolny panel) wpisujemy `assign` , naciskamy `[ENTER]` i analizujemy opis funkcji `assign`
3. Tworzymy nowy skrypt - np. `s2.R` , wpisujemy w nim:

```
 l1 <- "Nie bądź bezpieczny" # zmienna l1 (od line 1)
 l2 = "poeta pamięta"        # zmienna l2 (wariant z operatorem "=" jest n
iezalecany)
 "Możesz go zabić - narodzi się nowy" -> l3 # przypisanie z "lewej na praw
o"!
 assign("l4", "Spisane będą czyny i rozmowy") # uwaga na "" w definicji zm
iennej "l4"

 print(l1)
 print(l2)
 print(l3)
 print(l4)
```

   i testujemy działanie (np. wykonanie linia po linii - `Run` , całości - `Source` ,...)
4. Sprawdzamy zawartość zakładki `Environment`
5. W konsoli RStudio dodajemy dwie zmienne

```
> x1 <- 10
```

```
> .x2 <- 3.5
> x.3 <- "abc"
```

i sprawdzamy zawartość zakładki `Environment` (czy jest tam widoczna zmienna `.x2` ?)

6. W konsoli RStudio wpisujemy kolejno:

```
> ls() # czy .x2 jest na liście?
> ?ls
> ls(all.names = TRUE)
> rm(x.3) # sprawdzamy zawartość zakładki "Environment"
> ?rm
> ls()
> rm(list = ls()) # sprawdzamy zawartość zakładki "Environment"
> ls()
```

7. ***Zadania***:

    1. Na podstawie skryptu `s1.R` dopisz w `s2.R` część wykorzystującą funkcję `cat` , ale tym razem użyj zmiennych `l1` ... `l4`

    2. Ustal reguły nazewnictwa zmiennych w języku R (przykłady prawidłowych nazw to: `x1` , `X2` , `.x` , `x.y` , a nieprawidłowych: `1x` , `.2y`)

    3. [*opcjonalne*] Jeśli nie wiesz, z jakiego wiersza pochodzi wykorzystywany w ćwiczeniu fragment, ustal co to za wiersz i wpisz cały jego tekst do skryptu :)

    4. [*opcjonalne*] Zapoznaj się z działaniem funkcji `paste` (np. `> ?paste`); czy ta funkcja może być pomocna przy formatowaniu naszego wiersza?

4) 'Atomowe' typy danych w języku R - krótki przegląd

1. W menu `Session` wybieramy `Clear Workspace...` "; w okienku dialogowym wybieramy `Yes` . Sprawdzamy zawartość zakładki `Environment`

2. W konsoli RStudio wpisujemy kolejno:

```
> x1 <- TRUE
> class(x1)
> ?logical
> is.logical(x1)
>
> x2 <- 1.5
> class(x2)
> ?numeric
> is.numeric(x2)
>
> x3 <- 42
> class(x3)
> is.numeric(x3)
>
> x4 <- 42L
```

```
> class(x4)
> ?integer
> is.numeric(x4)
> is.integer(x4)
> is.integer(x3)
> is.integer(as.integer(x3))
>
> x5 <- 1 + 2i
> class(x5)
> ?complex
> is.complex(x5)
> as.complex(x3)
> as.complex(x1)
>
> x6 <- "Rękopisy nie płoną"
> class(x6)
> is.character(x6)
> x7 <- 'Rękopisy nie płoną'
> class(x7)
> is.character(x7)
> x8 <- "He said 'Hello, my name is Inigo Montoya...' and then..."
> x9 <- 'He said "Hello, my name is Inigo Montoya..." and then...'
> is.character(x8)
> is.character(x9)
```

3. W konsoli RStudio wpisujemy:

```
> is.vector(x1)
```

a następnie powtarzamy dla kolejnych zmiennych: `x2` ... `x9`

4. ***Zadania***:

    1. Wyjaśnij wyniki wywołania `is.vector(xi)` dla $xi = x1, \ldots, x9$

    2. W konsoli RStudio wpisz kolejno poniższe linie:

```
> 0.1 + 0.1 == 0.2
> 0.1 + 0.1 + 0.1 == 0.3 # ???
>
> maxN <- 2^.Machine$double.digits
> maxN + 1 == maxN # ???
>
> maxD <- .Machine$double.xmax
> maxD + 1 == maxD # ???
>
> xEps <- .Machine$double.eps / 2
> xEps + 1 == 1 # ???
```

    i wyjaśnij uzyskane wyniki.

3. Podaj kilka przykładów problemów/błędów, które mogą wynikać z nieznajomości powyżej poznanych "osobliwości" (tak naprawdę chodzi tu o znajomość/nieznajomość reprezentacji zmiennoprzecinkowej liczb rzeczywistych)

4. [*opcjonalne*] Poeksperymentuj z różnymi wariantami konwersji typu z wykorzystaniem `as.xxx()`

5) Operatory w języku R - krótki przegląd

1. W konsoli RStudio wpisujemy kolejno:

```
> 2 + 2
> 3 - 2
> 2 * 7
> 9 / 4
> 2 ^ 5
>
> 10 %% 3
> 10 %/% 3
>
> 2 == 2
> 2 != 2
> 3 > 2
> 3 >= 2
> 3 < 2
> 3 <= 2
>
> TRUE | FALSE
> (2 > 3) | (4 > 2)
> TRUE & FALSE
> (2 > 3) & (4 > 2)
> !TRUE == FALSE
> !!FALSE == !TRUE
```

2. [*opcjonalne*] W konsoli RStudio wpisujemy kolejno:

```
> ?c
> c(TRUE, FALSE) & c(TRUE, FALSE)
> c(TRUE, FALSE) && c(TRUE, FALSE)
> c(TRUE, FALSE) && c(FALSE, FALSE)
> c(FALSE, TRUE) | c(FALSE, TRUE)
> c(FALSE, TRUE) || c(TRUE, FALSE)
> c(FALSE, TRUE) || c(FALSE, TRUE)
```

   Uwagi:

   - funkcja `c()` tworzy wektor, czyli c(TRUE, FALSE) to dwuelementowy wektor
   - wektory będą umówione w jednym z kolejnych ćwiczeń

3. [*opcjonalne*] W konsoli RStudio wpisujemy kolejno:

```
> ?":"
> 1:5
> -5:5
> -5:-1
> class(1:5)
> is.vector(1:5)
>
> ?"%in%"
> 1 %in% -5:5
> 1 %in% 2:10
>
> ?matrix
> ?t
> M = matrix( c(1,2,3,4,5,6), nrow = 2, ncol = 3, byrow = TRUE)
> class(M)
> M %*% t(M)
```

Uwaga: macierze będą umówione w jednym z kolejnych ćwiczeń

4. ***Zadania***:

    1. Wyjaśnij wyniki poniższych działań:

    ```
    > TRUE + TRUE
    > TRUE - TRUE
    > TRUE * FALSE
    > TRUE ^ TRUE
    > TRUE ^ FALSE
    > FALSE ^ FALSE
    >
    > 1 * FALSE + 3 * TRUE
    > TRUE / FALSE
    > FALSE / FALSE
    > 2L + (FALSE + 1) / TRUE
    ```

    Uwaga: przydatne może być wykorzystanie funkcji `class()`, np. `class(1L + FALSE)`

    2. [*opcjonalne*] Wyjaśnij wyniki poniższych działań

    ```
    > (to.be <- FALSE == FALSE) | !to.be
    > rm(to.be)
    >
    > (to.be = FALSE == FALSE) | !to.be
    > rm(to.be)
    >
    > (FALSE == FALSE -> to.be) | !to.be
    ```

    3. [*opcjonalne*] Poeksperymentuj z operatorami w zakresie łączenia różnych typów danych w jednym wyrażniu (np. `"one" < 2`, `1 == "1"`, `"x" + 1`, `"abc" / 3`)

6) Wybrane typy danych języka R: `wektor`

1. W konsoli RStudio wpisujemy kolejno:

```
> v1 <- 1:5
> v2 <- c(10,20,30,40,50)
> v3 <- seq(100, 500, by = 100)
>
> is.atomic(v1)
> ?is.atomic
>
> v1
> v2
> v3
>
> v1 + v2
> > v2 - v1
> v1 * v2
> v2 / v1
> v2 ^ v1
>
> 2 * v2
> v3 + 1
>
> v1 + 2 * v2 + 3 * v3 / (v1 + v2)
> c(1,10,100,50) <= c(0, 12, 102, 7)
```

2. W konsoli RStudio wpisujemy kolejno:

```
> v4 <- c(4:1, 9:12)
> v5 <- c(100,1000)
> v4 + v5
> v5 + v4
>
> v6 <- 10:12
> v4 + v6
```

3. W konsoli RStudio wpisujemy kolejno:

```
> v7 <- 1:10
> v7[1]
> v7[10]
> v7[3] + 2 * v7[5] - v7[1]
```

4. W konsoli RStudio wpisujemy kolejno:

```
> v8 <- c(TRUE, FALSE, TRUE)
> v8
> class(v8)
>
```

```
> v9 <- c(TRUE, 1L, FALSE)
> v9
> class(v9)
>
> v10 <- c(TRUE, 1L, 2.5)
> v10
> class(v10)
>
> v11 <- c(TRUE, 1L, 2.5, "abc")
> v11
> clas(v11)
```

5. W konsoli RStudio wpisujemy kolejno:

```
> v12 <- c(1, 2, NA, 4)
> ?NA
> is.na(v12)
> class(NA)
> 2 ^ v12
> is.finite(v12)
>
> v13 <- c(100, 10, 0)
> v14 <- rep(1000, 3)
> v14 / v13
> ?rep
>
> v15 <- c(sqrt(3), sqrt(7), sqrt(-2))
> v15
> ?sqrt
> is.finite(v15)
> length(v15)
> ?length
>
> v16 <- c(0, 2) / 0
```

6. W konsoli RStudio wpisujemy kolejno:

```
> x12 <- 1:10
> x12
> x12[2:3] <- c(11,12)
> x12
> x12[-(1:4)] <- 20
> x12[] <- 0
```

7. ***Zadania***:

    1. Rozpisz dokładnie (na poziomie elementów wektorów) działania: `v4 + v5` i `v4 + v6` oraz `v1 + 2 * v2 + 3 * v3 / (v1 + v2)` ("reguła zawijania")

    2. Ustal sposób działania "reguły zawijania" - mechanizmu pozwalającego wykonywać operację wektorowe na wektorach różnej długości

3. Ustal sposób działania mechanizmu konwersji typów na przykładach wektorów `v9` , `v10` , `v11`

4. Porównaj znaczenia: `NA` , `NaN` , `Inf` i `NULL` i podaj kilka przykładów, w których te wartości specjalne się pojawiają

7) Wybrane typy danych języka R: `macierz`

1. W konsoli RStudio wpisujemy kolejno:

```
> ?matrix
> M1 <- matrix(c(1:12), nrow = 3)
> M1
> class(M1)
> is.vector(M1)
> dim(M1)
> nrow(M1)
> ncol(M1)
> ?dim
> ?nrow
> ?ncol
>
> M2 <- matrix(c(1:12), nrow = 3, byrow = TRUE)
>
> rownames <- c("r1", "r2", "r3")
> colnames = c("c1", "c2", "c3", "c4")
> M3 <- matrix(c(1:12), nrow = 3, byrow = TRUE, dimnames = list(rownames,
colnames))
>
> dimnames(M2) <- list(rownames, colnames)
> M2
```

2. W konsoli RStudio wpisujemy kolejno:

```
> M1[1,2]
> M2["r1","c2"]
> M2
> M2[1,]
> M2[c(1,3),]
> M2
> M2[c(1,3), c(2,4)]
```

3. W konsoli RStudio wpisujemy kolejno:

```
> M1[1,2] <- 12
> M1
> M2["r2","c2"] <- 22
> M2
> M1[c(1,2), c(1,2)] = matrix(rep(0,4))
```

4. W konsoli RStudio wpisujemy kolejno:

```
> M1 <- matrix(c(1:12), nrow = 3)
> M1 - 1
> M1^2
> M1 * (M1 - 1)
> M1 ^ (M1 - 7)
> (M1 - 1) / M1
```

5. W konsoli RStudio wpisujemy kolejno:

```
> M1 <- matrix(c(1:12), nrow = 3)
> M1
> cbind(M1, c(13,14,15))
> ?cbind
> M1
> M1 <- cbind(M1, c(13,14,15))
> M1
> ncol(M1)
> rbind(M1, rep(100, ncol(M1)))
```

6. ***Zadania***:

    1. Z macierzy `M3 <- matrix(c(1:12), nrow = 3)` , wybierz podmacierz składającą się 1. i 3. wiersza
    2. Z macierzy `M3` wybierz 3. kolumnę
    3. W macierzy `M4 <- matrix(c(1:16), nrow = 4)` przy pomocy jednej instrukcji ustaw na diagonali wartości 100 (*wskazówka*: pomocna może okazać się funkcja `diag()`)

8) Wybrane typy danych języka R: `tablica`

1. W konsoli RStudio wpisujemy kolejno:

```
> ?array
> A1 <- array(c(1:9, 101:109, 1001:1009), dim = c(3,3,3))
> A1
> A1[1,2,3]
> dimnames(A1) = list(c("a1", "a2", "a3"), c("b1", "b2", "b3"), c("c1", "
c2", "c3"))
> A1
```

2. W konsoli RStudio wpisujemy kolejno:

```
> A1[1,1,]
> A1[1,,]
> A1[,1,]
> A1[,,1]
> A1
```

3. W konsoli RStudio wpisujemy kolejno:

```
> A1[1,1,1] <- 1000
> A1[,1,1] <- rep(0, 3)
> A1[1,,] <- matrix(c(-9:-1), nrow = 3)
```

4. W konsoli RStudio wpisujemy kolejno:

```
> A1 <- array(c(1:9, 101:109, 1001:1009), dim = c(3,3,3))
> A1
> 2 * A1
> A1 / A1
> (A1 + 1) / A1^2
```

5. ***Zadania***:

    1. Sprawdź, czy funkcja `dim()` działa także na tablicach (array)

    2. [*opcjonalne*] Podaj kilka przykładowych problemów, w których tablica może być odpowiednią strukturą danych

9) Wybrane typy danych języka R: `czynnik (ang. factor)`

1. W konsoli RStudio wpisujemy kolejno:

```
> gen <- c("male","male","female","female","male","female","male")
> class(gen)
> is.factor(gen)
>
> genFact <- factor(gen)
> genFact
> genFact[1]
>
> str(gen)
> str(genFact)
> ?str
>
> levels(genFact)
> ?levels
>
> genFact[1] <- "female"
> genFact[1] <- "Female"
```

2. [*opcjonalne*] W konsoli RStudio wpisujemy kolejno:

```
> ?gl
> gl(3,2, labels = c("L1", "L2", "L3"))
> gl(2,3, labels = c("L10", "L20"))
```

3. ***Zadania***:

1. Podaj kilka przykładów zastosowania czynników (factors) w modelowaniu danych
2. [*opcjonalne*] Podaj przykład zastosowania funkcji `gl()`

10) Wybrane typy danych języka R: `lista`

1. W konsoli RStudio wpisujemy kolejno:

```
> ?list
> l1 <- list(1L, c(TRUE, FALSE), matrix(c(1:4), nrow = 2), list(1,2))
> l1[1]
> l1[2]
>
> l1[[2]]
> ?"[["
```

2. W konsoli RStudio wpisujemy kolejno:

```
> l1[2]
> l1Orig <- l1
> l1[2] = c(FALSE, FALSE)
> l1[2]
> l1[[2]]
>
> l1 <- l1Orig
> l1[[2]] <- c(FALSE, FALSE)
> l1[2]
> l1[[2]]
```

3. W konsoli RStudio wpisujemy kolejno:

```
> l_123 <- list(1,2,3)
> l_ABC <- list("A","B","C")
> l_123_ABC <- c(l_123, l_ABC)
>
> l_1To5 <- list(1:5)
> l_11To15 <- list(11:15)
> class(l_1To5)
> class(l_11To15)
> is.vector(l_1To5)
>
> v_l_1To5 <- unlist(l_1To5)
> v_11To15 <- unlist(l_11To15)
> class(v_l_1To5)
> class(v_11To15)
```

4. ***Zadania***:
    1. Podaj kilka przykładów zastosowania list w modelowaniu danych
    2. Porównaj listę z wektorem, macierzą, tablicą

11) Wybrane typy danych języka R: `ramka danych` - pierwsza wzmianka

1. W konsoli RStudio wpisujemy kolejno:

```
> emp.data <- data.frame(
    emp_id = c(1:3),
    emp_fname = c("Jan", "Marek", "Michał"),
    emp_sname = c("Kwiatkowski", "Nowak", "Kowalski"),
    salary = c(10000, 12000, 15000)
  )
> emp.data
> str(emp.data)
> ?str
>
> summary(emp.data)
> ?summary
```

2. W konsoli RStudio wpisujemy kolejno:

```
> ?"$"
> emp.data$emp_id
> emp.data$emp_fname
> emp.data$emp_sname
> emp.data$salary
>
> rows1And2 <- emp.data[1:2,]
> rows1And3 <- emp.data[c(1,3),]
>
> cols1And3 <- emp.data[,c(1,3)]
> data.frame(emp.data$emp_id, emp.data$emp_sname)
```

3. ***Zadania***:

    1. Przeanalizuj definicję ramki danych `emp.data` ; jaka struktura przechowuje kolumny, a jaka wiersze?
    2. Porównaj funkcje do pobierania danych z ramki z językiem SQL
    3. [*opcjonlane*] Sprawdź, czy istnieją funkcje dla ramek danych, które pozwalają uzyskać podobną do języka SQL "siłę wyrazu" (np. `SELECT ... FROM ... WHERE` , złączenia,...)