

Підготовка, обробка та ефективне
використання даних для наукових
досліджень

Юрій Клебан

2021-05-19

Contents

Загальна інформація	5
1 Вступ до курсу	7
План	7
1.1 Що таке R?	7
1.2 Історія створення R	8
1.3 Основи роботи з R	9
1.4 Основи роботи з пакетами в R	29
2 Базові конструкції мови R: типи та структури даних. Частина 1	35
План	35
2.1 Введення-виведення даних	36
2.2 Оголошення та ініціалізація змінних	36
2.3 Базові типи даних	37
2.4 Оператори	41
2.5 Корисні математичні функції	46
2.6 Введення-виведення даних	50
3 Базові конструкції мови R: типи та структури даних. Частина 2	51
План	51
3.1 Набори даних	51
3.2 Вектори (vectors)	51

4	4. Збір та читання інформації з різноманітних джерел: файли, веб-сторінки, бази R	55
	План	55
	4.1 Загальний опис структури даних	55
	4.2 Опис набору даних Telecom Users	56
	4.3 Імпорт даних засобами RStudio	58
	4.4 CSV-файли: читання, запис	58
	4.5 Excel (xlsx): читання, запис	62
	4.6 XML: читання, запис	65
	4.7 JSON and API	68
	4.8 Google Services	71
5	5. Прийоми маніпулювання даними (з використанням можливостей бібліотеки dplyr)	73
	План	73
	5.1 Загальний опис структури даних	74
	5.2 Опис набору даних Telecom Users	74
	5.3 Імпорт даних засобами RStudio	76
	5.4 CSV-файли: читання,	76
6	8. Представлення результатів аналізу даних.	77
7	7. Побудова та навчання математичних моделей, метрики оцінки їх якості.	79
8	Приклади задач та їх розв'язки	81
	8.1 Задачі	81
	8.2 Рішення задач	85
	Джерела	91

Загальна інформація

Увага. Курс у процесі розробки. Матеріали додаватимуться по мірі їх написання та рецензування.

R: НАУКОВА РОБОТА

Базовий курс з основ збору, аналізу та підготовки даних для наукових досліджень



Юрій Клебан, 2021



Курс створено у межах проекту “Підготовка, обробка та ефективне використання даних для наукових досліджень (на основі R)”, що підтримує Європейський союз за програмою House of Europe.

Chapter 1

Вступ до курсу

План

- Що таке R?
 - Історія створення R
 - Основи роботи з R
 - R Project
 - * Завантаження та інсталяція R
 - * Перший запуск R GUI
 - * Поняття робочого простору
 - * Поняття робочого каталогу
 - * Допомога (help/?)
 - Робота з R Studio
 - * Завантаження та інсталяція RStudio Desktop
 - * Створення першого проекту в RStudio
 - Робота з Jupyter Notebook
 - Огляд додаткових IDE та сервісів для роботи з R
 - Основи роботи з пакетами в R
 - Команди для роботи з пакетами
 - Робота з пакетами в RStudio
-

1.1 Що таке R?

R є поширеною мовою програмування для роботи з даними (DataScience) та машинного навчання (Machine Learning). Але Ви можете скористатися

засобами R і для простіших задач: обчислення, візуалізація даних.

Синтаксис мови програмування R є досить простим для вивчення та використання, а широкий набір готових пакетів дозволяє використати готові розробки для виіршення широкого спектру задач від статистичних обчислень до навчання нейронних мереж для розпізнавання/класифікації зображень.

Важливо відмітити, що мова програмування R є безкоштовною (*free*) і має відкритий код (*open source*).

R має ряд корисних властивостей, серед яких варто виділити:

- **Візуалізація даних.** Побудова різноманітних видів графіків, робота з мапами, широкий спектр бібліотек та налаштувань до них.
- **Повторне використання коду.** На відміну від електронних таблиць, що мають обмеження на кількість спостережень (наприклад, MS Excel), R дозволяє працювати з великими масивами даних та перезапускати обчислення у потрібний момент не створюючи додаткових копій даних.
- **Машинне навчання.** R дозволяє використати для побудови, навчання та тестування моделей, а також оптимізації гіперпараметрів та відбору факторів дуже велику кількість алгоритмів. Існують також спеціальні пакети, що об'єднують у собі усі описані функції та алгоритми, наприклад, *caret* та *mlr*.
- **Автоматизація.** Написаний код та проекти можна перетворити у готові до публікації та впровадження продукти (*deployment*) або використовувати напрацьовані алгоритми для швидкого вирішення схожих задач (*pipeline*).

Також можна виділити досить корисні фічі **Розробка веб-застосунків** та **Звітність**, адже, використовуючи спеціальні бібліотеки (*shiny*, *shinydashboard*, *flexdashboard*, *rmarkdown*, *knitr* тощо), результати виконаної роботи можна “оживити” або сформувати “на льоту” готові до презентації документи.

1.2 Історія створення R

Мова програмування R виникла як продовження статистичної мови S. Назва мови S була обрана аналогічно до C. Створена S була у

1976 році компанією Bell Labs. Мова S мала кілька версій і широко використовувалася для комерційного програмування. Найпотужнішою була версія S-Plus, що мала реалізацію за досить немалою кількістю функцій під Windows та Unix-платформи, що стримувало її розвиток. Саме в цей момент розпочинається історія R.

Влітку 1993 року двоє молодих новозеландських вчених анонсували свлю нову розробку, яку вони назвали R (є інформація, що буква "R" була обрана тому, що вона стоїть перед "S" у латинському алфавіті, тут є аналогія з мовою "C", якій передувала мова "B") (А.Б. Шипунов, Е.М. Балдин, П.А. Волкова, А.И. Коробейников, С.А. Назарова, С.В. Петров, В.Г. Суфьянов., 2012). За задумом авторів (Robert Gentleman та Ross Ihaka) це повинна була бути нова реалізація мови S, що відрізнялася від S-Plus деякими деталями, наприклад, роботою з локальними та глобальними змінними, пам'яттю тощо. Фактично було створено нову мову, що відгалуджується від S.

Проект з самого початку розвивався досить повільно, але коли у команди розробників R з'явилися ресурси, в тому числі зручна системи створення розширень (пакетів), все більше аналітиків, статистиків, вчених, програмістів почало переходити з S-Plus на R. Коли були усунуті проблеми роботи з пам'яттю перших версій R, на цю мову почали переходити користувачі інших статистичних пакетів (SAS, Stata, SYSSTAT).

Кількість книг та публікацій у мережі Інтернет по роботі з R постійно зростає разом із зацікавленням молодих і вже досвідчених спеціалістів зі сфери ІТ темою науки про дані, машинним навчанням, аналітикою для бізнесу, охорони здоров'я тощо.

1.3 Основи роботи з R

1.3.1 R Project

R є безкоштовним програмним забезпеченням, що розповсюджується за умовами GNU General Public License. Код, написаний на R компілюється та запускається на різних платформах: UNIX, Windows, MacOS (R Core Team, 2020).

1.3.1.1 Завантаження та інсталяція R

Для завантаження актуальної версії R варто перейти на сайт проекту <https://cran.r-project.org/>.

На сайті обираємо завантаження R для потрібної операційної системи. У межах курсу ми вкристиовуємо Windows, проте на синтаксис мови програмування та процес написання коду це не впливає:

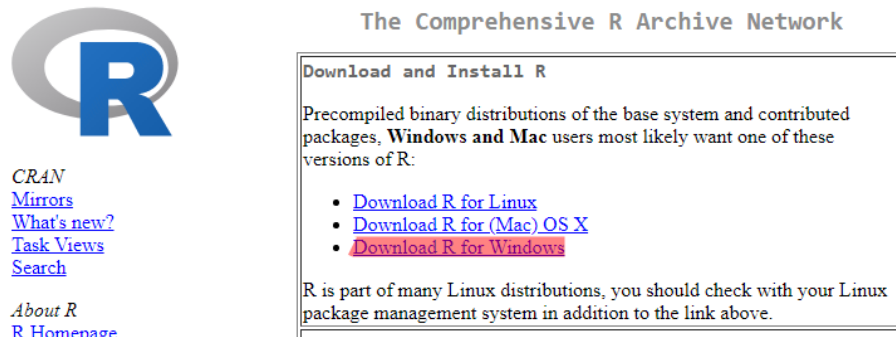


Figure 1.1: Завантаження R. Вибір ОС

У наступному вікні клікаємо на **install R for the first time**:

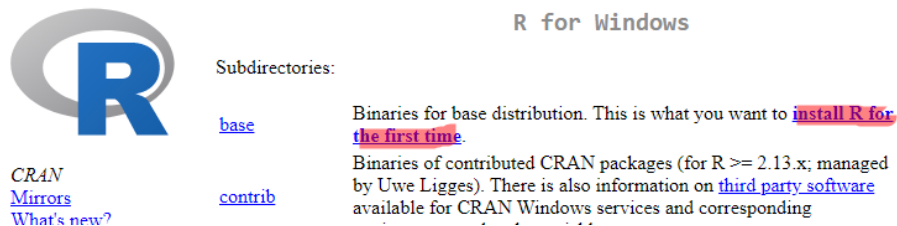


Figure 1.2: Завантаження R. Перша інсталяція

Далі обираємо **Download R 4.X.X for Windows**, де 4.X.X версія R, яка може бути відмінною на момент вивчення курсу:

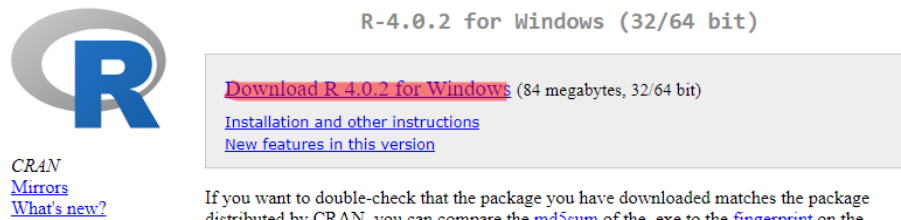


Figure 1.3: Завантаження R. Завантаження версії для ОС

Після завантаження файлу інсталяції потрібно його запустити. Зазвичай завантажений файл можна побачити у лівому нижньому кутку браузера або у розділі "Завантаження" Вашого браузера. Наприклад, у браузері Google Chrome знайти цей пункт меню так:

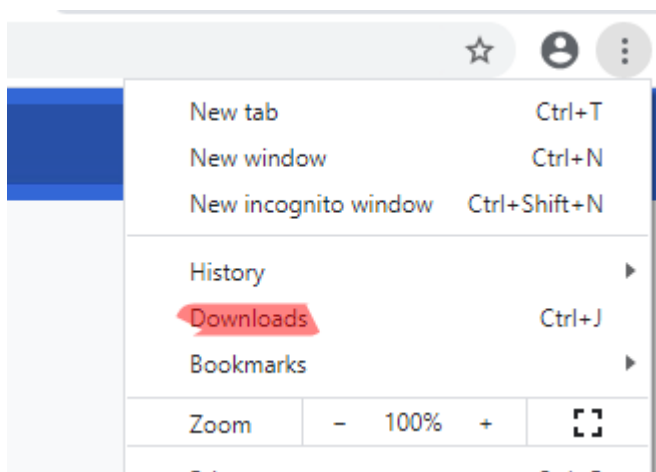


Figure 1.4: Завантаження R. Розділ “Завантаження” у Google Chrome

Процес інсталяції ПЗ не відрізняється від інших програм і детального опису не потребує. Основним тут є запам'ятати шлях встановлення проекту або “відмітити галочками” пункти щодо публікації на *Робочий стіл* чи у меню швидкого доступу ярликів для того, щоб знайти файли запуску.

1.3.1.2 Перший запуск R GUI

За замовчуванням під час інсталяції пропонується шлях `C:\Program Files\R\R-4.X.X.`

Для запуску R GUI (стандартного графічного інтерфейсу для роботи з R) потрібно зайти у папку `bin\x64` (або `i386`, якщо у Вас 32-х розрядна ОС) та запустити файл `Rgui.exe`.

Вигляд вікна R GUI зображено нижче:

GUI (Graphical User Interface) - набір візуальних компонентів для інтерактивної взаємодії користувача з програмним забезпеченням.

У вікні *R Console* можна вводити команди/інструкції R, що будуть виконуватися:

Результати виконання команд зберігаються у пам'яті програми і можуть бути використані у наступних блоках коду:

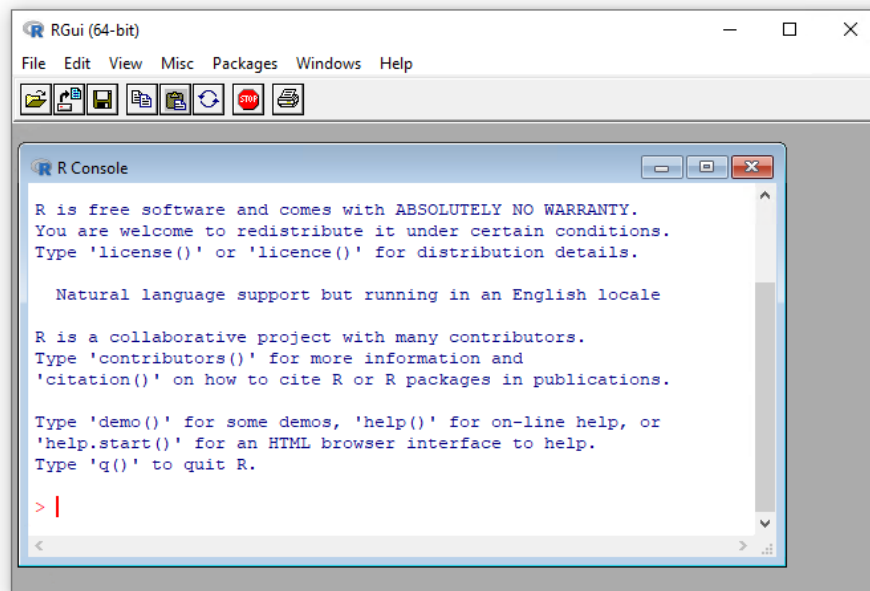
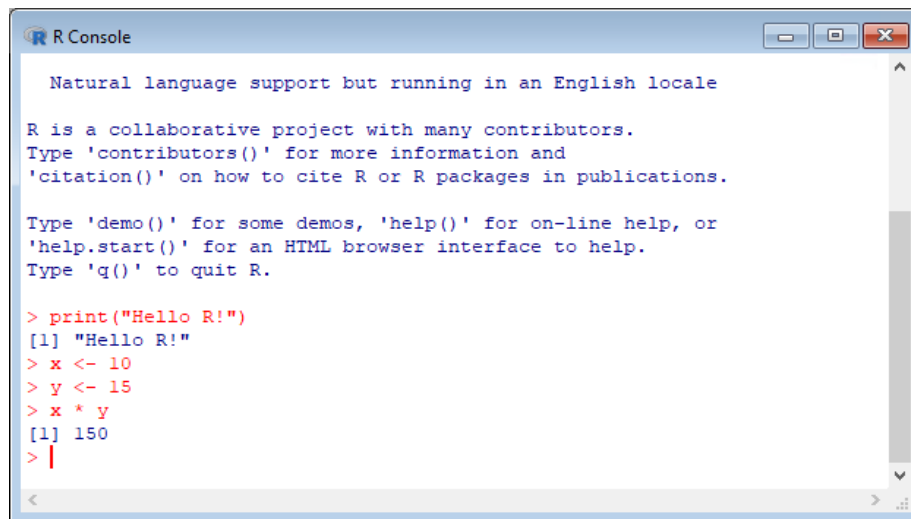


Figure 1.5: Вигляд головного вікна RGui



Середовище R GUI має широкий спектр функцій і дозволяє написати будь-якого рівня складності проекти на R, проте він є лише базовою графічною обгорткою для R. Розглянемо інші зручніші середовища для написання R-коду.

1.3.1.3 Поняття робочого простору

У процесі виконання коду створені об'єкти/змінні та функції зберігаються у поточній сесії. У R є можливість переглянути список збережених елементів, видалити усі або окремі, зберегти стан поточної сесії на диск та завантажити його пізніше, щоб не проходити усі етапи виконання коду повторно *(інколи дуже складний код може виконувати досить довго і збереження проміжних результатів може бути хорошим рішенням)*.

Для прикладу створимо дві змінні `var1`, `var2` та виведемо на консоль їх значення:

```
var1 <- 10
var2 <- sqrt(15)
var1
```

```
## [1] 10
```

```
var2
```

```
## [1] 3.872983
```

Для того аби переглянути список змінних у поточній сесії варто скористатися `ls()`:

```
ls()
```

```
## [1] "var1" "var2"
```

Якщо виникає потреба очистити робочий простір і звільнити пам'ять використовується команда `rm()`. Так, щоб очистити усі змінні можна скористатися `rm(list = ls())`, якщо ж Ви хочете видалити якусь одну/дві змінних, то просто вкажіть імена:

```
rm(list = c("var1"))
ls()
```

```
## [1] "var2"
```

Таким чином, після виконання коду вище, залишиться лише змінна `var2`.

Зберігання образу (image) робочого простору на диск здійснюється за допомогою команди `save.image("...", "RData")`, а його зчитування за допомогою `load("...", "RData")`:

```
# Clear workspace
rm(list = ls())

# declare data
a <- 10
b <- a + 15

# Save image to file
save.image("tmp.RData")
```

```
# Clear workspace
rm(list = ls())

# load image to file
load("tmp.RData")

print(a)
```

```
## [1] 10
```

```
print(b)
```

```
## [1] 25
```

У прикладі 2 не створюєть жодного параметра, проте вони збережні у файлі сесії.

Для того аби зберегти та зчитати окремиий об'єкт, а не всі елементи сесії у R є спеціальний формат .RDS, який реалізовується методами `saveRDS(' ', file=" _ .rds")` та `readRDS(file=" _ .rds")`.

1.3.1.4 Поняття робочого каталогу

Робота в будь-якому середовищі передбачає зв'язок із поточним каталогом, відносно якого будуються шляхи до файлів. Звичайно можна писати завжди повний шлях до файла, проте такий підхід є досить негнучким і під час перенесення коду між ПК створює чимало проблем розробникам.

Для визначення базового каталогу R в поточній сесії використовують команду `getwd()`. Якщо Ви користуєтеся RStudio та створили проект, то цей каталог буде відповідати повному шляху до папки проекту:

```
getwd()
```

```
## [1] "E:/Repos/YuRa/r-science-dev"
```

Для того аби змінити поточний робочий каталог використовують команду `setwd()`. Після запуску цієї команди функцій `getwd()` буде вказувати уже на нову адресу/шлях.

Варто знати та вміти будувати **абсолютні** та **відносні** шляхи до каталогів та файлів, ці знання корисні для роботи з усіма мовами програмування та більшістю ПЗ для роботи з даними.

Для запису шляху у ОС Windows можна скористатися 2-ма способами:

- `/` - **слеш**, записується як один знак;
- `\\` - **бекслеш**, записується як два знаки.

У прикладі нижче обидва шляхи ведуть до тієї ж папки (drive - буква диска):

```
setwd("drive:/folder1/folder2/")
setwd("drive:\\folder1\\folder2\\")
```

Для перегляду інформації про наявні каталоги та файли у поточній робочій папці можна скористатися командою `dir()` або `list.files()`:

```
dir()
```

```
## [1] "_bookdown.yml"      "_bookdown_files"    "_output.yml"
## [4] "01-chapter1.Rmd"    "01-chapter1_files"  "01-intro_files"
## [7] "02-chapter2.Rmd"    "02-chapter2_files"  "03-chapter3.Rmd"
## [10] "04-chapter4.Rmd"    "05-chapter5.Rmd"    "06-chapter6.Rmd"
## [13] "07-chapter7.Rmd"    "09-chapter9.Rmd"    "10-references.Rmd"
## [16] "book.bib"           "css"                 "data"
## [19] "favicon.ico"        "images"              "inc"
## [22] "index.md"           "index.Rmd"           "packages.bib"
## [25] "preamble.tex"       "r-science.log"      "r-science.rds"
## [28] "README.md"          "render_commands"     "render3424696c2c3d.rds"
## [31] "RScience.Rproj"     "sss.R"               "tmp.R"
## [34] "tmp.RData"
```

```
list.files()
```

```
## [1] "_bookdown.yml"           "_bookdown_files"       "_output.yml"
## [4] "01-chapter1.Rmd"         "01-chapter1_files"     "01-intro_files"
## [7] "02-chapter2.Rmd"         "02-chapter2_files"     "03-chapter3.Rmd"
## [10] "04-chapter4.Rmd"         "05-chapter5.Rmd"       "06-chapter6.Rmd"
## [13] "07-chapter7.Rmd"         "09-chapter9.Rmd"       "10-references.Rmd"
## [16] "book.bib"                "css"                   "data"
## [19] "favicon.ico"             "images"                "inc"
## [22] "index.md"                "index.Rmd"             "packages.bib"
## [25] "preamble.tex"            "r-science.log"        "r-science.rds"
## [28] "README.md"               "render_commands"       "render3424696c2c3d.rds"
## [31] "RScience.Rproj"          "sss.R"                  "tmp.R"
## [34] "tmp.RData"
```

1.3.1.5 Допомога (help/?)

Для отримання швидкої довідки в R варто скористатися функцією `help(_ ' _ _)` або `? _ ' _ _`:

```
# Get help for intersect() function
help(intersect)
```

Якщо є потреба отримати інформацію про пакет скористайтеся:

```
help(package = "stats")
```

1.3.2 Робота з R Studio

1.3.2.1 Завантаження та інсталяція RStudio Desktop

RStudio - це інтегроване середовище розробки для R. Воно включає у себе консоль, підсвічування синтаксису (підказки), прямий запуск коду, інструменти для візуалізації графіків, html-коду, історію виконаних команд, відлагоджування коду, управління робочими просторами, підтримка різних видів розмітки та багато іншого. RStudio має версію з відкритим кодом та комерційну версію для Windows, Linux та Mac, а також веб-версію для серверів на Linux RStudio Server та RStudio Server Pro (R-s, 2021).

IDE (integrated development environment) - комплексне програмне рішення для розробки програмного забезпечення. Зазвичай, складається з редактора початкового коду, інструментів для автоматизації складання та відлагодження програм. Більшість сучасних середовищ розробки мають можливість автодоповнення коду. Wikipedia

Завантажити продукти можна з сайту <https://rstudio.com>. Щоб знайти середовище, яке ми будемо використовувати під час вивчення курсу варто виконати наступні кроки:

1. У головному меню сайту обрати Products > RStudio.
2. Знаходимо на сторінці кнопку для завантаження програми RStudio Desktop версії Open Source та натискаємо **DOWNLOAD RSTUDIO DESKTOP**:

	Open Source Edition	RStudio Desktop Pro
Overview	<ul style="list-style-type: none"> • Access RStudio locally • Syntax highlighting, code completion, and smart indentation • Execute R code directly from the source editor • Quickly jump to function definitions • Easily manage multiple working directories using projects • Integrated R help and documentation • Interactive debugger to diagnose and fix errors quickly • Extensive package development tools 	<p>All of the features of open source; plus:</p> <ul style="list-style-type: none"> • A commercial license for organizations not able to use AGPL software • Access to priority support • RStudio Professional Drivers • Connect directly to your RStudio Server Pro instance remotely
Support	Community forums only	<ul style="list-style-type: none"> • Priority Email Support • 8 hour response during business hours (ET)
License	AGPL v3	RStudio License Agreement
Pricing	Free	\$995/year

[DOWNLOAD RSTUDIO DESKTOP](#)

[DOWNLOAD FREE RSTUDIO DESKTOP PRO TRIAL](#)

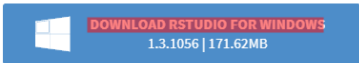
Figure 1.6: Вибір версії RStudio Desktop

3. Далі обираємо завантаження безкоштовної версії RStudio Desktop з наданого переліку:

Після завантаження запускаємо інсталятор RStudio. Особливих кроків у цьому процесі немає.

RStudio Desktop 1.3.1056 - [Release Notes](#)

- 1. Install R. RStudio requires R 3.0.1+.
- 2. Download RStudio Desktop. Recommended for your system:



Requires Windows 10/8/7 (64-bit)



All Installers

Linux users may need to [import RStudio's public code-signing key](#) prior to installation, depending on the operating system's security policy.

RStudio requires a 64-bit operating system. If you are on a 32 bit system, you can use an [older version of RStudio](#).




OS	Download	Size	SHA-256
Windows 10/8/7	 RStudio-1.3.1056.exe	171.62 MB	a8f1fee5
macOS 10.13+	 RStudio-1.3.1056.dmg	148.64 MB	f343c77d
Ubuntu 16	 rstudio-1.3.1056-amd64.deb	124.56 MB	cbd5e5e5

Figure 1.7: Завантаження RStudio Desktop

Після запуску IDE RStudio зазвичай складається з 3-х або 4-х блоків: * Файл, з яким працювали останнім (зліва зверху). * Консоль для введення коду та виведення результатів (зліва знизу). * Змінні середовища (Environment) (справа зверху) + Історія команд (History), З'єднання з зовнішніми ресурсами даних, наприклад, бази даних (Connections), навчальна інструкція (Tutorial). * Файли каталогу або проекту (Files), Інсталювані пакети (Packages), Допомога (Help), Візуалізація результатів (Plots, Viewer).

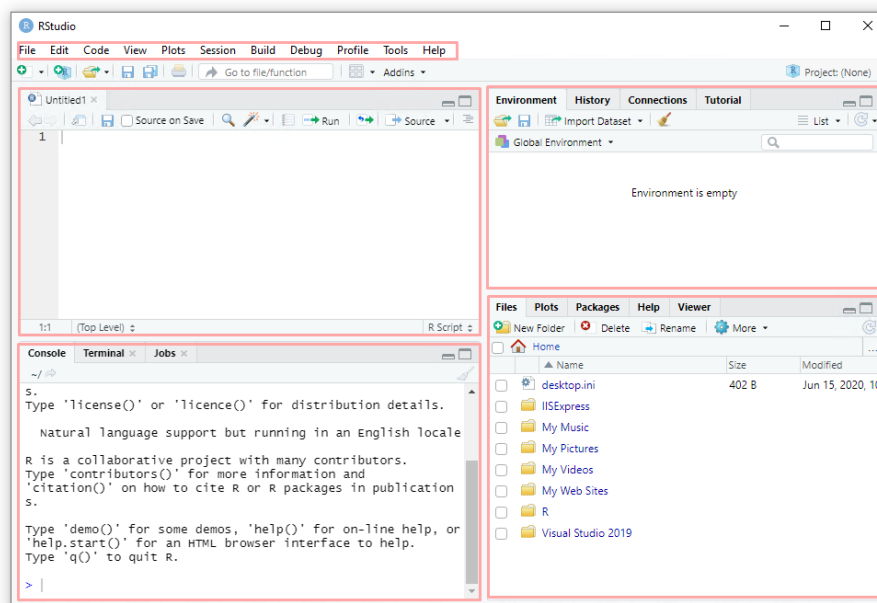
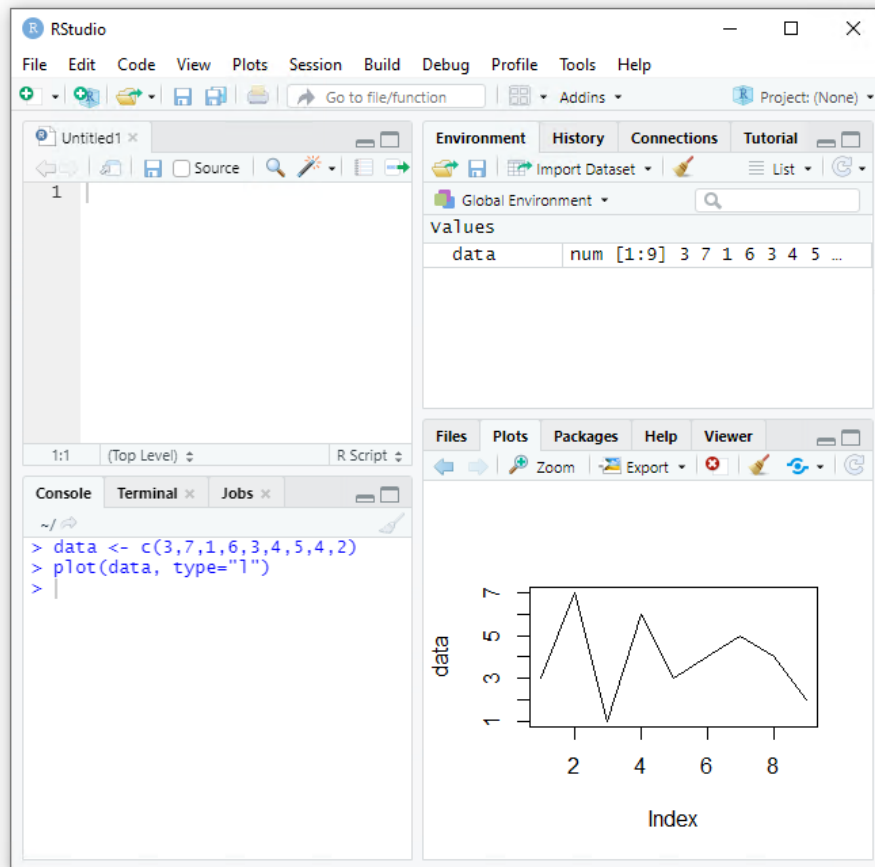


Figure 1.8: Головне вікно RStudio Desktop

Для першої демонстрації роботи виконаємо у консолі 2 рядки коду:

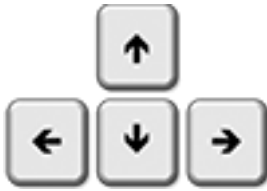


Перший рядок з кодом `data <- c(3,7,1,6,3,4,5,4,2)` створює у пам'яті колекцію чисел. Зверніть увагу, що у блоці **Environments** відображаються усі змінні, що уснують у поточному робочому просторі (про це буде далі).

Другий рядок `plot(data, type="l")` дозволяє побудувати простий лінійний графік (`type="l"` – linear, `"p"` – point, `help(plot)` для деталей). Графіки, що “промальовуються” як картинки відображаються у блоці **Plots**. Якщо ж графік має більш складну візуалізацію з інтерактивними елементами, що використовують уже засоби `html/css/js`, то він буде відображений у блоці **View**.

Якщо переключитися на вкладку **History**, то ми побачимо перелік раніше виконаних команд.

Для швидкого “гортання” уже виконаних раніше команд на консолі (*Console*) можна скористатися клавішами Up/Down на клавіатурі:



1.3.2.2 Створення першого проекту в RStudio {chapter1322}

На відміну від R Gui в RStudio реалізовано концепцію проектів, що дозволяє організувати код та поєднати різні його частини у межах певного рішення.

Створимо наш перший проект.

Для початку оберемо з верхнього меню пункт `File > New Project`. У вікні вибору способу створення проекту клікаємо `New Directory`. Такий спосіб передбачає, що жодного файлу проекту поки не існує або ми пізніше туди скопіюємо уже готовий код.

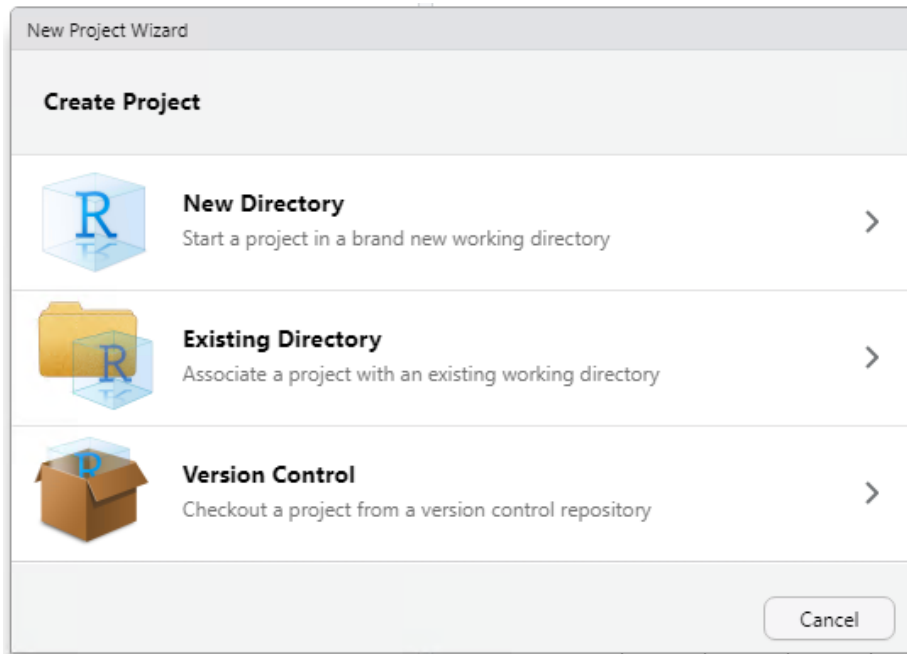


Figure 1.9: RStudio Desktop. Новий проект

На наступному кроці обираємо `New Project`:

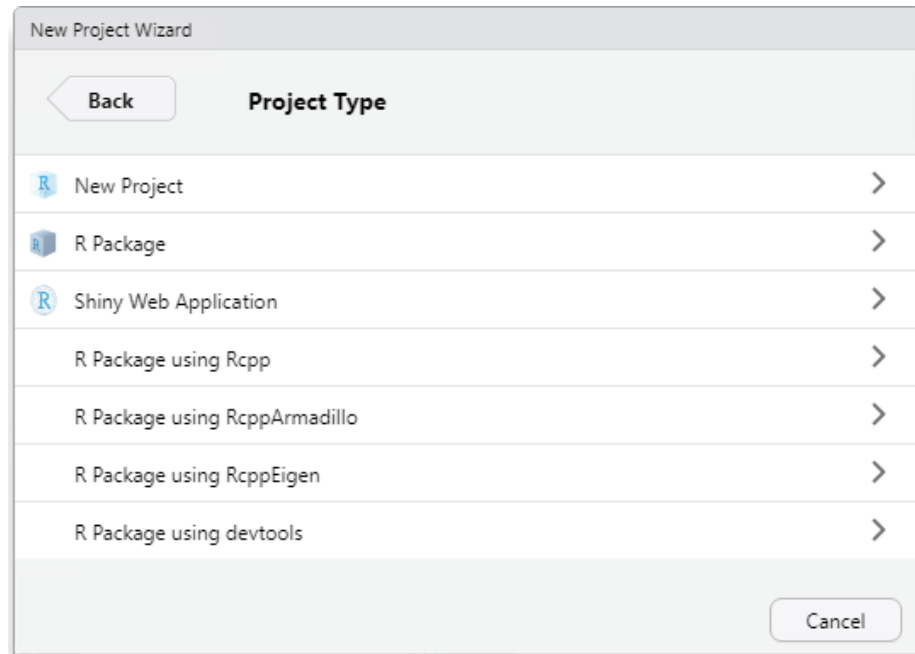


Figure 1.10: RStudio Desktop. Новий проект. Тип проекту

Після кліку на **Create Project** буде створено папку за попередньо обраним шляхом. Для запуску проекту або швидкого перемикання між проектами можна скористатися як пунктами головного меню, так і підменю проектів справа. Також відкрити проект можна запуском файлу `*.Rproj` у провіднику Windows.

Щоб додати новий файл з кодом R потрібно обрати з головного меню **File > New file > R Script** або скористатися командою `Ctrl+Shift+N`. Новий файл буде створено з назвою `Untitled[X]`, тому рекомендую одразу його зберегти, наприклад, як `TestCode.R`.

Для першого проекту розв'яжемо наступну задачу:

Написати програму, що генерує вектор з 20-ти випадкових чисел у межах `[1;5]`, обчислює середнє та суму чисел, а також виводить гістограму частоти кожного значення (скільки разів дане число повторюється у векторі).

Код для генерації 20-ти випадкових чисел у діапазоні `[1;5]` матиме наступний вигляд:

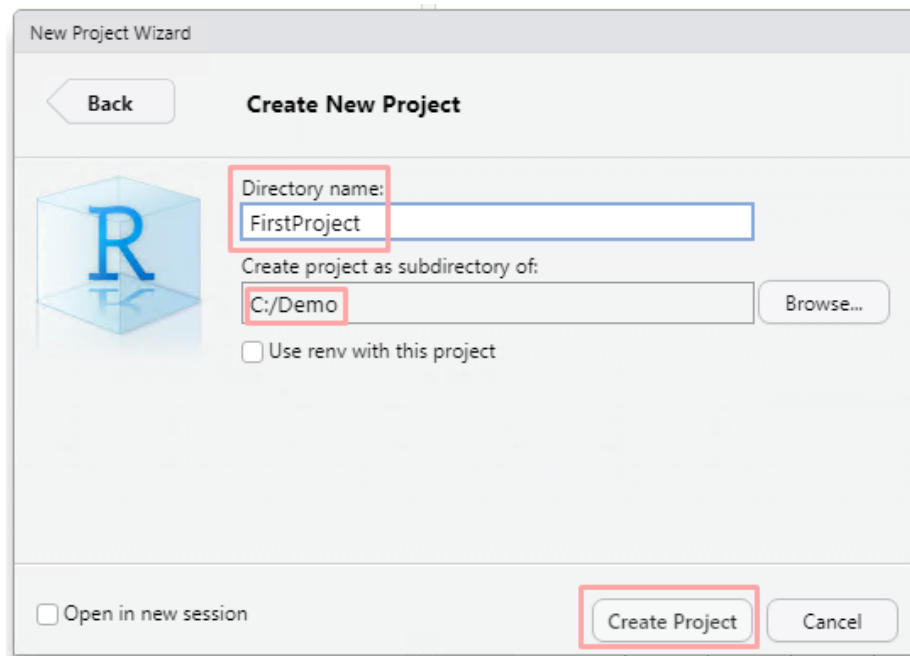


Figure 1.11: RStudio Desktop. Новий проект

```
vtr <- sample(1:5, 20, replace=TRUE)
vtr
```

```
## [1] 5 1 1 1 1 5 4 5 4 4 3 5 5 5 1 4 4 2 2 2
```

Результати виконання на Вашому ПК будуть іншими, адже **псевдогенератор** випадкових чисел буде брати іншу “точку відліку” для генерування чисел. Рекомендую перегляду функцію `set.seed()`.

P.S. Також зустрічав у мережі інформацію, що робота `set.seed()` для R у 4+ версії може відрізнятися від 3+. Після перевірки інформації оновлю даний текст.

Обчислення та виведення на консоль інформації про суму та середнє значення:

```
vtr_sum <- sum(vtr)
vtr_mean <- mean(vtr)

print(paste0("Sum: ", vtr_sum))
```

```
## [1] "Sum: 64"
```

```
print(paste0("Mean: ", vtr_mean))
```

```
## [1] "Mean: 3.2"
```

Виведемо гістограму:

```
hist(vtr, breaks = 5)
```

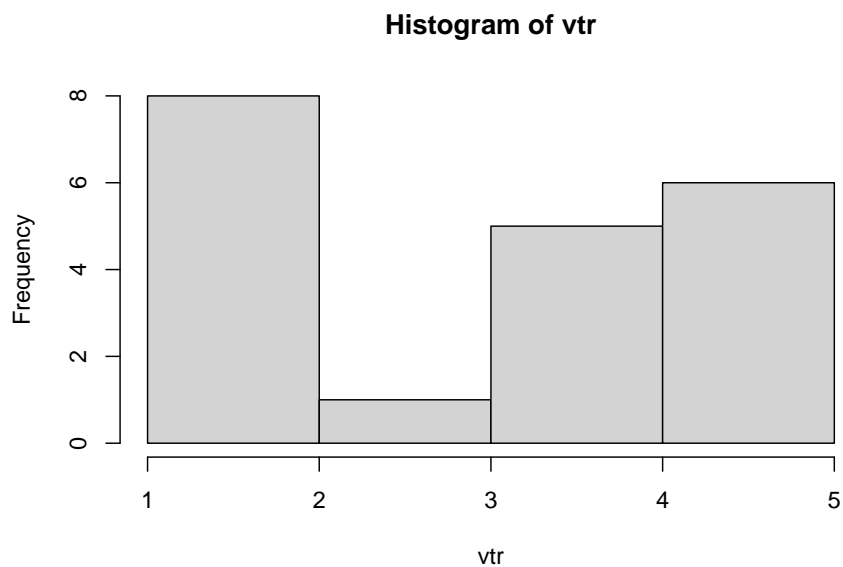


Figure 1.12: Приклад візуалізації гістограми в R

Примітка. Детальніше про параметри функції `hist()` можна почитати тут: <https://www.rdocumentation.org/packages/graphics/versions/3.6.2/topics/hist>.

Орієнтовний вигляд вікна RStudio після виконання усіх описаних вище операцій матиме наступний вигляд:

Варто звернути увагу на виділений блок `Environment`, де можна переглянути усі доступні змінні, що є на даний момент у `'`. До цих параметрів можна звертатися у коді чи з консолі у будь, який момент. Детальну інформацію про робоче середовище розглянуто нижче.

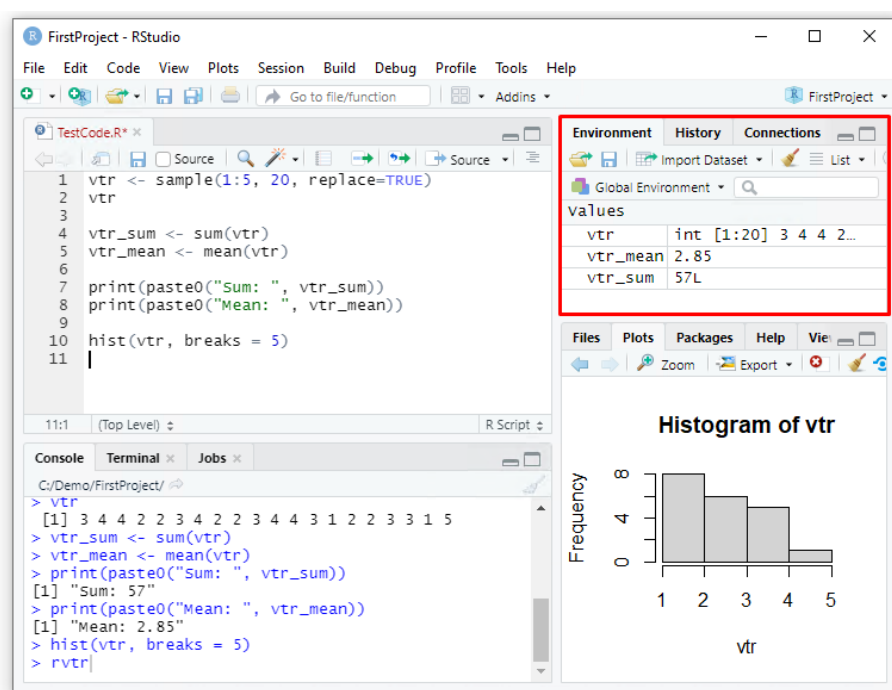


Figure 1.13: RStudio Desktop. Перегляд змінних

1.3.3 Робота з Jupyter Notebook

Ноутбуки стали зручним та поширеним інструментом для аналізу даних, а також послідовного викладення матеріалів чи результатів дослідження. Перевагою такого інструменту є перемішування коду, результатів його виконання та іншого текстового наповнення, що дозволяє сформувати “на льоту” готові до читання документи.

*Примітка. Лекційні матеріали даного курсу виконані саме у ноутбуках.

Використання ноутбуків у навчальному процесі дозволяє описати не лише теоретичний матеріал, але приклади коду, що будуть виконувати безпосередньо під час ознайомлення з лекцією. Також слухач курсу може відредагувати наявний код та перевірити результати його виконання.

Розглянемо процес інсталяції та запуску Anaconda (середовище з відкритим кодом для вирішення задач Data Science) та Jupyter Notebook на ПК.

Для встановлення середовища Anaconda потрібно перейти на сайт проекту та завантажити індивідуальну версію продукту: <https://www.anaconda.com/products/individual> (Ana, 2021).

*Примітка. Усі операції у даному курсі виконуються під операційну систему Windows 10 Education.

Процес інсталяції середовища Anaconda не відрізняється від стандартного покрокового встановлення програм у Windows.

Після запуску Anaconda Navigator для початку потрібно створити нове середовище та налаштувати роботу R:

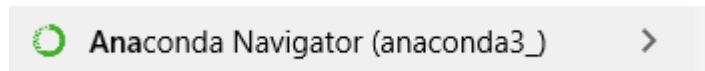


Figure 1.14: Anaconda

Для початку потрібно перейти на вкладку Environments та натиснути Create:

У вікні, що відкрилося потрібно відмітити [x] встановлення інструментів для роботи з R:

Після встановлення R-інструментів потрібно переключитися на вкладку Home та робочий простір:

Після завантаження робочого простору оберіть Launch для запуску Jupyter Notebook з переліку встановлених засобів. Jupyter Notebook буде запущено у браузері за замовчанням Вашого ПК. Відкрити ноутбук можна обравши потрібний файл, а створити новий у меню справа New > Notebook > R:

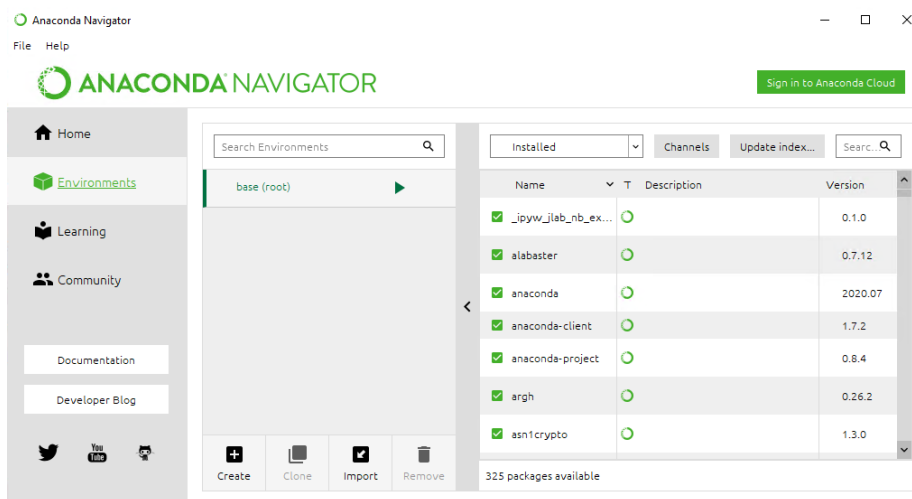


Figure 1.15: Anaconda. Головне вікно

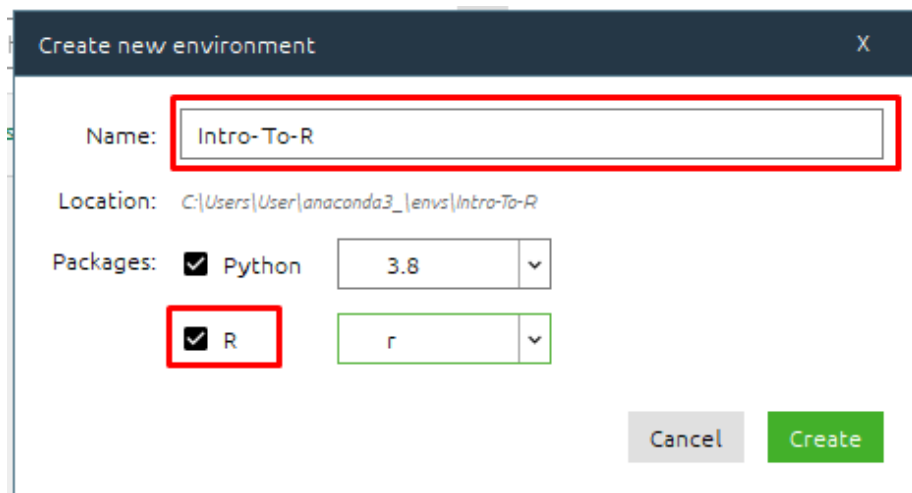


Figure 1.16: Anaconda. Створення нового середовища на основі R

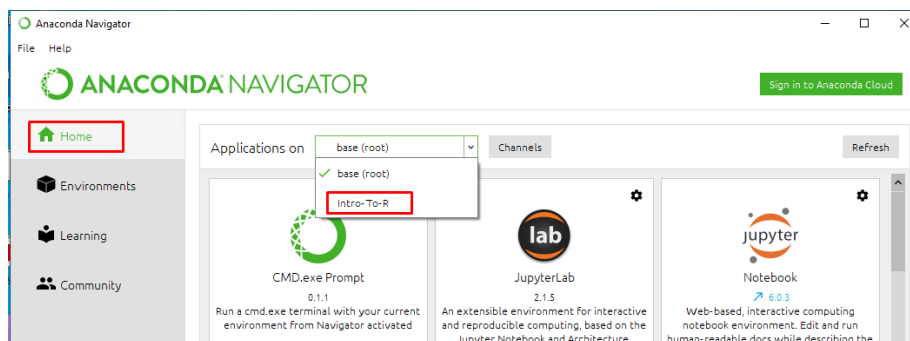


Figure 1.17: Anaconda. Зміна середовища

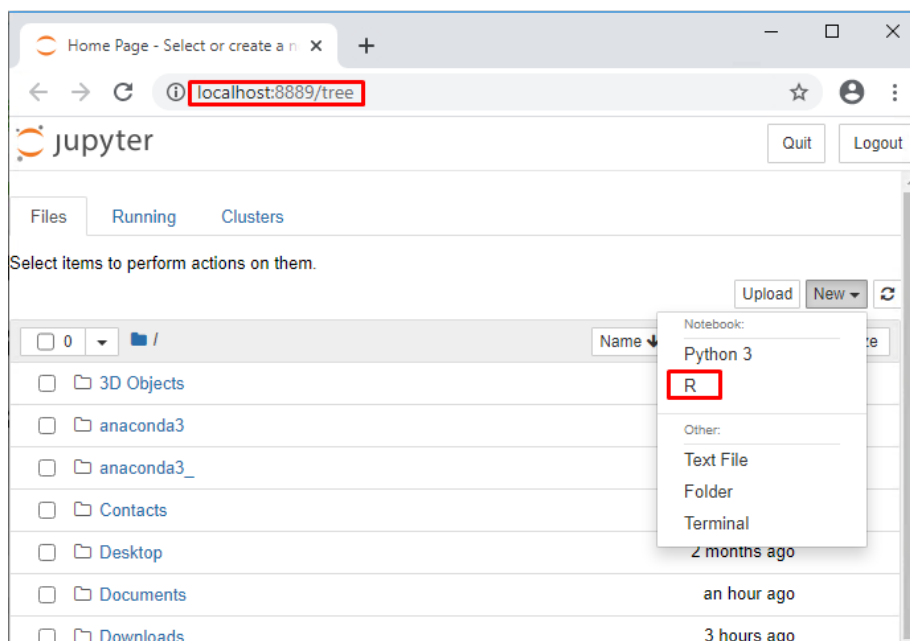


Figure 1.18: Jupyter Notebook. Створення нового ноутбука

1.3.4 Огляд додаткових IDE та сервісів для роботи з R

Окрім середовищ описаних вище існує ряд досить цікавих інструментів, що роблять досить зручною роботу з R-кодом. Розглянемо ці інструменти.

Visual Studio Code - безкоштовний редактор коду від Microsoft, орієнтований на велику кількість мов програмування та фреймворків (vs-, 2021). Серед інших інструментів у VS Code доступні також розширення для роботи з R:

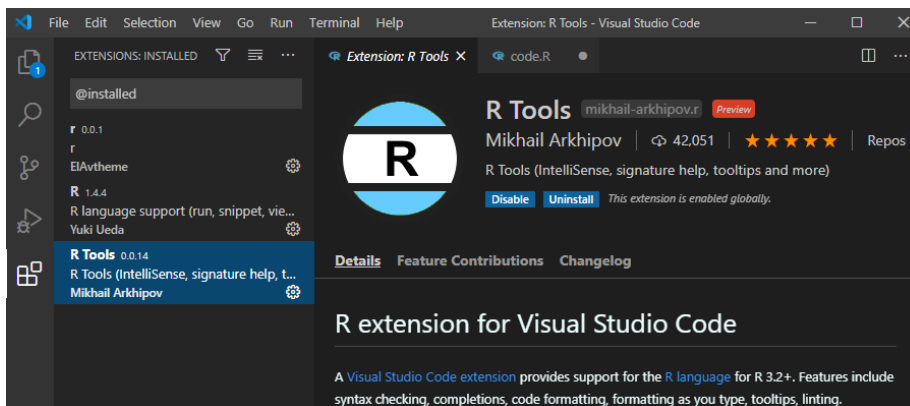


Figure 1.19: Visual Studio Code. Інсталяція RTools

Visual Studio Community Edition - безкоштовне середовище розробки від компанії Microsoft. VS створено з самого початку для розробки під платформу .NET та мови програмування C#, VB.NET, F# тощо, але з часом отримало багато розширень, що дозволяють у тому числі, працювати і з проектами в R (vis, 2021).

Google Collab - онлайн сервіс для роботи з ноутбуками для Data Science від компанії Google (goo, 2021):

Примітка. Код у прикладі вище написаний на Python.

kaggle.com - сервіс для змагань з Data Science та Machine Learning. Окрім переліку змагань, наборів даних сервіс має досить зручні ноутбуки.

Загалом сервісів та середовищ для розробки в R існує досить багато і їх кількість зростає, але це не впливає на принципи написання коду та роботу з даними.

1.4 Основи роботи з пакетами в R

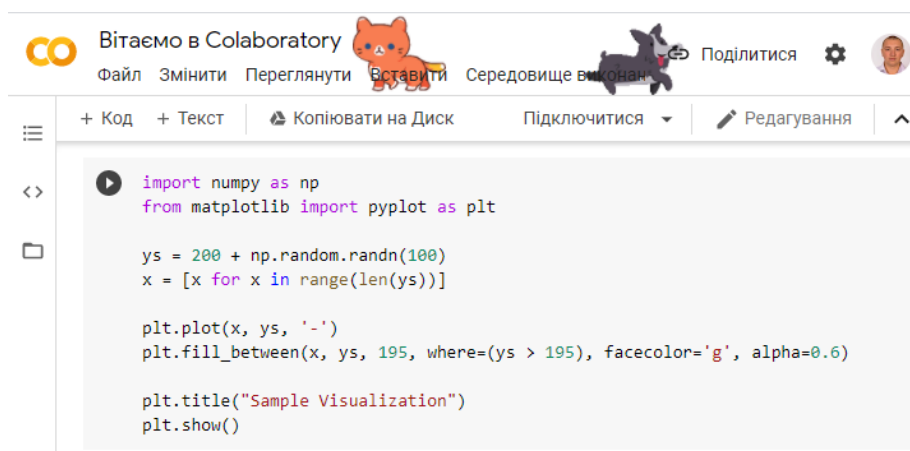


Figure 1.20: Google Collab

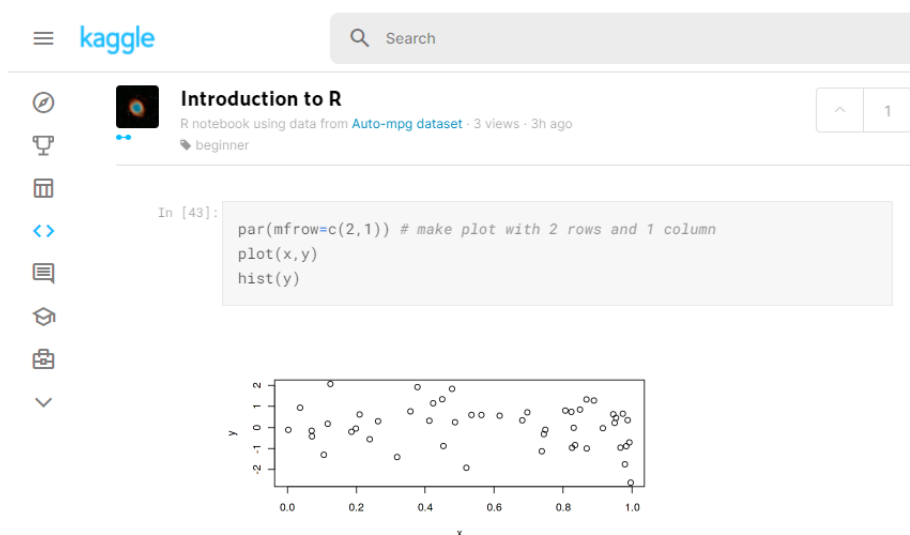


Figure 1.21: Kaggle.com

1.4.1 Команди для роботи з пакетами

Своєю популярністю R завдячує, у тому числі, і можливості швидко реалізувати досить складні дослідження за допомогою наборів уже готових функцій. Такі функції об'єднуються у пакети та публікуються вченими, дослідниками та розробниками зі всього світу.

Пакети в R - організовані набори методів та класів для виконання вузького набору задач під час програмування на R. Вони містять як функції так і опис способів їх використання, а чтакож дані для відтворення прикладів коду.

Пакети можуть бути завантажені з офіційного сайту проекту cran.r-project.org / (R Core Team, 2020) або інших джерел (dev-версії є доступні на [github](https://github.com)). Завантаження пакетів у R можна здійснювати як з локального диска, так і з серверів у мережі інтернет.

Для встановлення пакету використовується команда `install.packages()`:

```
install.packages("fun")
```

Для підключення пакету та його використання варто скористатися `library()`:

```
packageDescription("fun")  
help(package = "fun")
```

Дуже рекомендую почитати детальніше про пакети у статті на DataCamp: R Packages: A Beginner's Guide.

1.4.2 Робота з пакетами в RStudio

Робота з пакетами в RStudio організована досить зручно і дозволяє швидко переглянути інформацію про пакет та функції, які він дозволяє використати.

Для інсталяції та оновлення пакетів можна скористатися меню **Tools**:

Після вибору **Install Packages...** відкриється вікно, де можна обрати як джерело інсталяції пакету так і сам пакет з переліку, ввівши кілька перших букв його назви:

RStudio дозволяє також переглянути інстальовані пакети/бібліотеки, розроблені іншими користувачами та завантажені у пам'ять ("галочка" навпроти назви пакету):

Доступ до функцій та інших елементів пакету можна здійснювати використавши запис `_ :: _ ()` без підключення бібліотеки за допомогою `library()`:

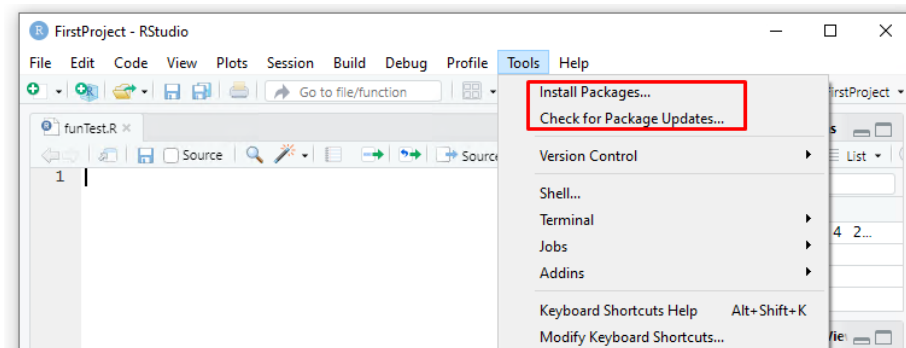


Figure 1.22: RStudio Desktop. Меню інсталяції пакетів

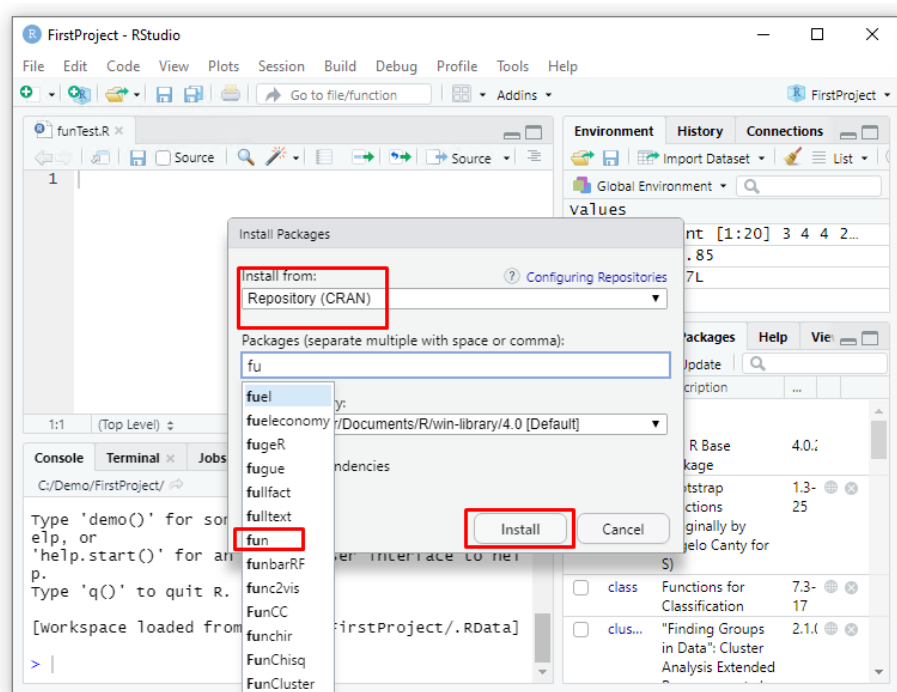


Figure 1.23: RStudio Desktop. Вибір пакету для інсталяції

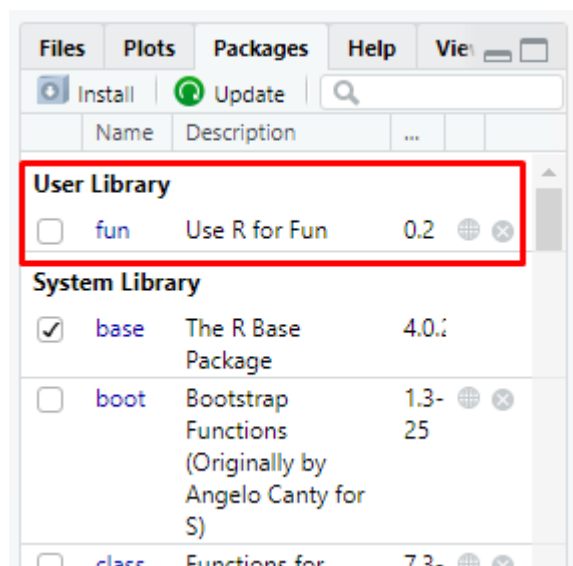


Figure 1.24: RStudio Desktop. Перегляд інстальованих пакетів

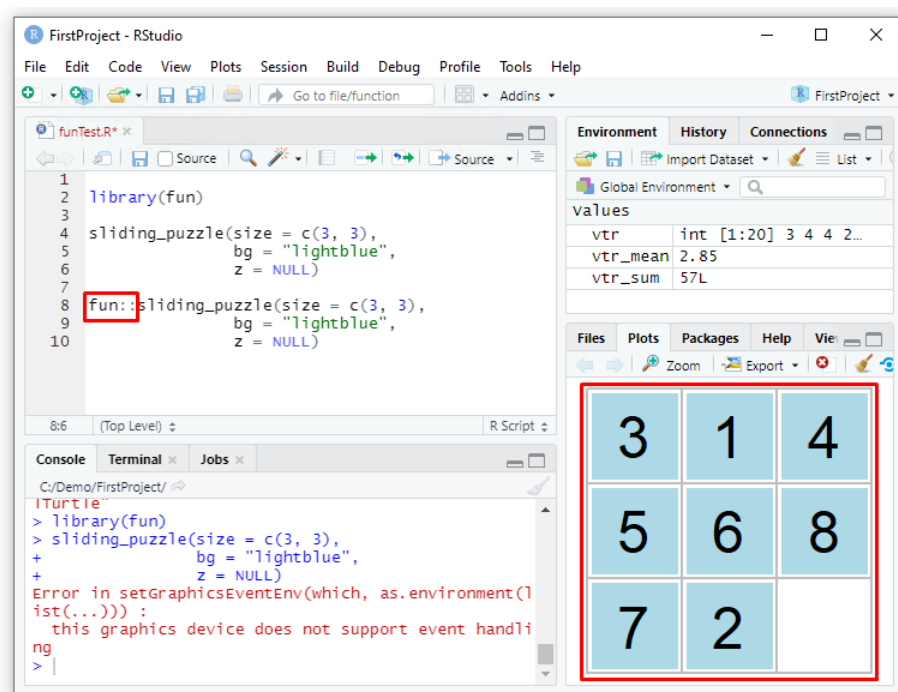


Figure 1.25: RStudio Desktop. Приклад використання пакету fun

Користувачі можуть не тільки завантажувати існуючі пакети, але і створювати власні та робити їх доступними для дослідників зі всього світу.

Chapter 2

Базові конструкції мови R: типи та структури даних. Частина 1

План

- Оголошення та ініціалізація змінних
 - Поняття змінних та оператор присвоєння
 - Правила іменування змінних
- Базові типи даних
 - Типізація в R
 - Перевірка та приведення типів даних
- Оператори
 - Арифметичні оператори
 - Оператори відношення
 - Логічні оператори
- Корисні математичні функції
 - Заокруглення чисел(round, ceiling, floor, trunc, signif)
 - Послідовності чисел (seq, rep)
 - Генерація псевдовипадкових чисел
 - Інші математичні функції та константи R
 -

2.1 Введення-виведення даних

2.2 Оголошення та ініціалізація змінних

2.2.1 Поняття змінних та оператор присвоєння

Базовим поняттям практично усіх мов програмування є **змінна**. Змінна дозволяє записати значення або об'єкт та назвати його для подальшого доступу, зміни, видалення по імені.

Наприклад, присвоєння змінній `my_variable` значення 10 записується так: `my_variable <- 10` або `my_variable = 10`.

Операція надання змінній певного значення у програмуванні називається **присвоєнням**.

Важливо! Зверніть увагу, що присвоєння (`<-`, `=`) та рівність (`==`) це різні поняття. Оператор `==` здійснює перевірку співпадіння значення двох змінних/об'єктів та повертає результат у вигляді логічного значення `TRUE` (якщо значення рівні) або `FALSE` (якщо значення не рівні).

Знак `<-` не є часто використовуваним у різних мовах програмування, зазвичай для присвоєння користуються `=`. Проте в R основним способом засобом початкової ініціалізації змінних є `<-`.

Також у програмуванні на R використовуються оператори присвоєння `<<-`, `->`, `->>`. Про них можна прочитати за ланками нижче.

Рекомендую почитати про різницю між операторами присвоєння у R `<-` та `=` тут:

1. Why do we use arrow as an assignment operator? (Colin FAY).
2. Difference between assignment operators in R (Ren Kun).
3. Assignment Operators.

Приклад:

```
x <- 45
y <- 10
z <- x + y # z = 45 + 10
z
```

```
## [1] 55
```

Розберемо приклад, описаний вище:

- У першому рядку оголошується змінна `x` і їй присвоюється значення 45.
- У другому рядку оголошується змінна `y` і їй присвоюється значення 10.
- У третьому рядку оголошується змінна `z` і їй присвоюється значення суми `x + y`. `#` у R використовується як коментар коду, текст написаний після нього ігнорується.
- У четвертому рядку відбувається виведення на консоль змінної `z`.

2.2.2 Правила іменування змінних

Є кілька основних правил іменування змінних у R: 1. Ім'я змінної може складатися з **букв** [a-z, A-z], **цифр** [0-9], **крапки** `.` та **нижнього підкреслювання** `_`. 2. Ім'я змінної повинно починатися з **букви або крапки**. Якщо воно починається з крапки, то наступним символом повинна бути буква. 3. Не можна використовувати зарезервовані ключові слова мови програмування для іменування змінних, наприклад, TRUE/FALSE.

Ім'я змінної не може містити пробіл (space). Якщо є потреба назвати об'єкт кількома словами, то їх зазвичай розділяють підкресленням `_` або крапкою `..` Наприклад, змінну можна назвати `my_variable_name` або `my.variable.name`. Назва `myVariableName` (camel case) теж буде коректно сприйнята мовою програмування R, проте такий запис тут вживається не часто.

Приклад коректного іменування змінних: `total`, `zminna`, `Sum`, `.length_of_something`, `Number123`, `x_1`.

Приклад неправильного іменування змінних: `tot@1`, `5x_1`, `_variable`, `FALSE`, `.0ne`.

2.3 Базові типи даних

2.3.1 Типізація в R

Усі мови програмування мають власну типізацію даних з якими працюють. Тип даних - це набір властивостей певних об'єктів та операцій, що можна з ними виконувати. Так, наприклад, з **цілими числами** можна виконувати арифметичні операції додавання, віднімання та інші. Набори символів (простими словами **текст**) зазвичай можуть використовуватися для пошуку у них елементів, редагування (шляхом видалення частини існуючого або додавання нового тексту), склеювання та розділення на частини.

У R, на відміну від строго типізованих мов програмування, тип даних визначається на основі поточного значення елемента і може змінюватися у процесі виконання.

Розглянемо приклад коду з мови програмування C# (мова родом із C/Java):

```
int a = 10;
a = "some text";
```

Подібний код у C# передбачає створення нової змінної `a` типу `int` (integer - ціле число), а потім відбувається присвоєння для `a` текстового фрагмента (тип `string` у #). Такий код не буде запущено і виникне **помилка компіляції**.

Розглянемо приклад коду з R:

```
a <- 10
a <- "some text"
a

## [1] "some text"
```

Такий код виконається і на консоль буде виведено `some text`, адже у 1 першому рядку було присвоєно ціле число, у другому - текст. Таким чином R має **динамічну типізацію**, що дозволяє у ту ж саму змінну записати значення різних типів. Проте варто пам'ятати, що попереднє значення буде втрачено.

До базових типів даних у R варто віднести:

- Числа з дробовою частиною (decimal numbers), як наприклад, 4.0, 15.214, що називаються `numeric(s)`.
- Натуральні числа (natural numbers), як наприклад, 4, 15, що називаються `integer(s)`.
- Логічні значення (boolean values), тобто TRUE та FALSE (які також можна скорочено записувати T та F), що називаються `logical`.
- Текст або рядки (string values), як наприклад, "Hello", "12 is number", що називаються `character(s)`.

Оголосимо для прикладу три змінні: `my_numeric` - число, `my_character` - текст, `my_logical` - логічне значення.

```
my_numeric <- 5
my_character <- "universe"
my_logical <- FALSE
```

Замінімо значення `my_character <- "5"` та спробуємо знайти суму значень:

```
my_character <- "5"  
my_sum <- my_numeric + my_character
```

У результаті виконання даного коду ми отримаємо помилку, адже значення 5 та "5" є елементами різних типів даних, перевіримо типи за допомогою функції `class()`:

```
class(5)
```

```
## [1] "numeric"
```

```
class("5")
```

```
## [1] "character"
```

Виконання коду `class(5)` показує нам, що 5 є значенням числового типу даних `numeric`, а `class("5")` відповідає тексту `character`, тому арифметична операція додавання між цими значеннями неможлива.

2.3.2 Перевірка та приведення типів даних

У випадку коли тип даних потрібно визначити у процесі виконання програми/коду та перетворити значення використовується приведення типів даних.

Приведення типів даних - операція перетворення значення з одного типу даних в інший. Важливо пам'ятати, що не завжди приведення типів даних може бути здійснено. Так, наприклад, значення "5" (`character`) можна досить просто привести до 5 (`numeric`), проте "five" не буде зрозумілим для інтерпритатора.

Для перевірки належності елемента до певного типу даних використовують спеціальну функцію `is. _ ()`. Ця функція повертає `TRUE`, якщо елемент належить даному типу і `FALSE`, якщо не належить.

Розглянемо приклад:

```
my_numeric <- 5  
my_character <- "five"  
my_logical <- FALSE  
  
is.numeric(my_numeric)
```

```
## [1] TRUE
```

```
is.character(my_numeric)
```

```
## [1] FALSE
```

Для перетворення типу даних можна скористатися функцією `as. _ ()`. У результаті виконання функції буде повернуто значення потрібного типу або пусте значення `NA`, якщо таке приведення не є можливим:

```
a <- 5
b <- "10"
c <- "10, 20"
as.numeric(b)
```

```
## [1] 10
```

```
as.numeric(c)
```

```
## Warning: NAs introduced by coercion
```

```
## [1] NA
```

Результат виконання функцій можна записувати у змінні і використовувати у наступних обчисленнях:

```
a <- 5
b <- "10"
b <- as.numeric(b)
a + b
```

```
## [1] 15
```

```
number <- as.integer(54)
typeof(number)
```

```
## [1] "integer"
```

```
class(number)
```

```
## [1] "integer"
```

Повний перелік типів та методів перевірки і приведення їх типів об'єктів нижче:

Назва типу	Метод перевірки типу	Метод приведення типу
Array	is.array	as.array
Character	is.character	as.character
Complex	is.complex	as.complex
Dataframe	is.data.frame	as.data.frame
Double	is.double	as.double
Factor	is.factor	as.factor
List	is.list	as.list
Logical	is.logical	as.logical
Matrix	is.matrix	as.matrix
Numeric	is.numeric	as.numeric
Raw	is.raw	as.raw
Time series (ts)	is.ts	as.ts
Vector	is.vector	as.vector

2.4 Оператори

2.4.1 Арифметичні оператори

R можна використовувати як звичайни калькулятор.

Розглянемо набір звичних арифметичних операторів, що відомі з початкової школи: * Додавання: +. * Віднімання: -. * Ділення: /. * Множення: *.

А також більш складні оператори: * Піднесення до степеня: ^ (вводиться з клавіатури як Shift+6 на ENG-розкладці клавіатури). * Остача від ділення (ще може називатися "ділення по модулю"): %% (вводиться з клавіатури як Shift+5). * Ділення націло: %/%.

Розглянемо приклад **додавання** чисел:

```
5 + 10
```

```
## [1] 15
```

```
5 + 4 + 15
```

```
## [1] 24
```

```
5 + 53 + 343
```

```
## [1] 401
```

```
(5 + 8) + (4 + 9)
```

```
## [1] 26
```

Примітка. Використання “круглих” дужок у програмуванні виразах має пріоритет аналогічний до загальноприйнятих у математиці.

Розглянемо приклад **віднімання** чисел:

```
47 - 21
```

```
## [1] 26
```

```
15 - (10 - 25)
```

```
## [1] 30
```

Примітка. Заміна знаків до/в “дужках” тут працює так само як працювала у школі :)

Приклади **множення** чисел:

```
5 * 3
```

```
## [1] 15
```

```
5 * (2 + 5)
```

```
## [1] 35
```

Приклади **ділення** чисел:

```
12 / 2
```

```
## [1] 6
```

```
(4 + 7) / 3
```

```
## [1] 3.666667
```

Піднесення до степеня за допомогою оператора \wedge є досить простим. Так, наприклад, 3^2 дорівнює 9, а 2^3 - це $2*2*2$ і дорівнює 8.

```
5^2
```

```
## [1] 25
```

```
(1+3)^3 + 100
```

```
## [1] 164
```

Остача від ділення дозволяє знайти залишок одного числа від ділення на інше число.

Наприклад, остача від ділення націло 5 на 2 дорівнює 1, бо $2 * 2 (=4) + 1 = 5$

```
28 %% 7
```

```
## [1] 0
```

```
17%%5
```

```
## [1] 2
```

Ділення націло залишає лише цілу частину від ділення двох чисел:

```
28 %/% 7
```

```
## [1] 4
```

```
17 %/% 5
```

```
## [1] 3
```

```
Sys.setlocale("LC_CTYPE", "ukrainian")
```

```
## [1] "Ukrainian_Ukraine.1251"
```

```
# ,
```

В окремих випадках інтерпритатори R можуть некоректно читати або взагалі ввжати за помилку наявність кирилиці у коді. Тоді варто вказати явно локалізацію, яку Ви бажаєте використовувати. Для української локалізації варто на початку коду додати рядок:

```
Sys.setlocale("LC_CTYPE", "ukrainian")
```

2.4.2 Оператори відношення

Оператори відношення відповідають за порівняння двох об'єктів між собою та повертають значення логічного типу TRUE, якщо результат істинний та FALSE, якщо результат хибний.

Перелік операторів відношення:

- Більше або дорівнює >=.
- Менше <.
- Менше або дорівнює <=.
- Дорівнює ==.
- Не дорівнює !=

Для демонстрації принципів роботи операторів відношення оголосимо 3 змінні a, b та c.

```
a <- 12
b <- 5
c <- 7
```

Розглянемо кілька прикладів використання описаних вище операторів.

Оператори, що відповідають за перевірку на “більше/менше”:

```
a > b
```

```
## [1] TRUE
```

```
b + c < a
```

```
## [1] FALSE
```

```
b + c <= a
```

```
## [1] TRUE
```

Оператори, що відповідають за перевірку на “рівність/нерівність”:

```
a != b
```

```
## [1] TRUE
```

```
a == b + c
```

```
## [1] TRUE
```

```
b == c
```

```
## [1] FALSE
```

2.4.3 Логічні оператори

До логічних операторів у R відносяться:

- **I** & (амперсанти, Shift-7) - виконання усіх умов одночасно.
- **A**БО **I** (вертикальна риска, Shift+\) - виконання однієї із умов.
- **HE** ! (знак оклику, Shift+1) - заперечення.

Важливо розуміти відмінності між цими операторами вміти використовувати результати їх роботи. Для початку варто розглянути таблицю істинності:

A	B	Оператор I	Оператор A БО I	Заперечення A (HE A)
FALSE	FALSE	FALSE	FALSE	TRUE
FALSE	TRUE	FALSE	TRUE	TRUE
TRUE	FALSE	FALSE	TRUE	FALSE
TRUE	TRUE	TRUE	TRUE	FALSE

Матеріали розділу не доповнені прикладами.

2.5 Корисні математичні функції

2.5.1 Заокруглення чисел (round, ceiling, floor, trunc, signif)

Як ми знаємо з математики, що заокруглення чисел буває “вверх”, “вниз” або відносно деякого значення, зазвичай пов’язаного із цифрою 5 (3.6 заокруглюємо до цілого як 4, а 3.2 як 3, вважючи 3.5 межею).

Увага! Заокруглення чисел у програмуванні може призводити до помилок у результатах обчислень. Для задач бізнесу або технічних процесів мінімальні відхилення можуть призводити до викривлених результатів або збоїв у системах.

Функція `round()`

Примітка. Тут і надалі функції будуть позначатися як `назва()` (назва і “круглі” дужки).

Для заокруглення дійних чисел (з дробовою частиною) за правилом <0.5 & ≥ 0.5 (не знаю як називається науково) використовується функція `round(x, y)`, де x - число, y - точність (кількість знаків після коми/крапки). Наприклад:

```
round(3.557, 2)
```

```
## [1] 3.56
```

```
round(3.241, 2)
```

```
## [1] 3.24
```

```
round(-3.557, 2)
```

```
## [1] -3.56
```

```
round(-3.241, 2)
```

```
## [1] -3.24
```

Також можна використати `round(x)` з одним параметром, тоді заокруглення відбудеться до цілої частини, наприклад:

```
round(124.345)
```

```
## [1] 124
```

Функція `floor()`

Для заокруглення до найближчого меншого цілого числа слід скористатися функцією `floor()`:

```
floor(3.557)
```

```
## [1] 3
```

```
floor(3.241)
```

```
## [1] 3
```

```
floor(-3.557)
```

```
## [1] -4
```

```
floor(-3.241)
```

```
## [1] -4
```

Функція `ceiling()`

Для заокруглення до найближчого більшого цілого числа слід скористатися функцією `ceiling()`:

```
ceiling(3.557)
```

```
## [1] 4
```

```
ceiling(3.241)
```

```
## [1] 4
```

```
ceiling(-3.557)
```

```
## [1] -3
```

```
ceiling(-3.241)
```

```
## [1] -3
```

Функція *trunc()*

Функція *trunc()* у R використовується для отримання найбільшого цілого числа, яке більше або рівне x . Простими словами це означає, що для чисел менших 0 ($x < 0$) *trunc()* працює як *ceiling()*, а для чисел більших нуля $x > 0$, як *floor()*:

```
x <- 5.34
```

```
print(paste("trunc:", trunc(x), "ceiling:", ceiling(x), "floor:", floor(x), sep = " "))
```

```
## [1] "trunc: 5 ceiling: 6 floor: 5"
```

```
x <- x * -1
```

```
print(paste("trunc:", trunc(x), "ceiling:", ceiling(x), "floor:", floor(x), sep = " "))
```

```
## [1] "trunc: -5 ceiling: -5 floor: -6"
```

Функція *signif()*

Іноколи виникає потреба заокруглити не десяткову частину числа, а десятки, сотні, тисячі і так далі. Розглядемо варіант, коли у нас є велике число 11 547 741.3 і нам потрібно коротко його записати як 11.5. Для таких задач можна використати функцію *signif(x,y)*, де x - число, яке потрібно заокруглити до певного порядку, y - порядок заокруглення (рахувати від початку). Наприклад:

```
big_number <- 11547741.3
```

```
rounded_big_number <- signif(big_number,3)
```

```
rounded_big_number
```

```
## [1] 11500000
```



```
rounded_big_number / 1000000
```

```
## [1] 11.5
```

2.5.2 Послідовності чисел (seq, rep)

Матеріали розділу у процесі підготовки.

2.5.3 Генерація псевдовипадкових чисел

Матеріали розділу у процесі підготовки.

```
runif(10)
```

```
## [1] 0.60089332 0.14275770 0.73405972 0.25181121 0.13240470 0.61392797
## [7] 0.10903451 0.98914813 0.02155678 0.46768025
```

```
sample(100)
```

```
## [1] 75 83 59 49 6 85 17 84 53 8 24 67 58 3 48 54 18 51
## [19] 66 14 73 46 62 12 23 20 29 1 74 61 80 52 33 10 95 94
## [37] 11 43 96 55 31 21 98 36 76 34 16 92 44 28 81 91 65 97
## [55] 78 57 68 99 40 82 70 27 88 39 38 13 22 4 19 42 56 69
## [73] 9 47 71 26 79 72 30 5 60 15 41 7 77 90 32 64 25 100
## [91] 63 45 86 89 93 35 87 50 37 2
```

2.5.4 Інші математичні функції та константи R

Окрім описаного вище набору функцій R містить дуже велику кількість реалізованих функцій з різних сфер науки, бізнесу, техніки тощо. Прочитати про них можна з офіційної документації пакетів, у яких вони реалізовані та знайти за допомогою функції `help()` або `?name`.

Далі розглянемо перелік найпоширеніших функцій, що використовуються для розв'язання навчальних задач під час вивчення основ програмування.

Функція	Призначення, опис
$\log(x)$	Логарифм числа x за основою e
$\log(x,n)$	Логарифм числа x за основою n
$\exp(x)$	e у степені x

Функція	Призначення, опис
<i>sqrt(x)</i>	Корінь квадратний числа x
<i>factorial(x)</i>	Факторіал числа x
<i>abs(x)</i>	Модуль числа x

Також у R доступні ряд тригонометричних функцій, які вивчалися у школі і не тільки, серед них $\cos(x)$, $\sin(x)$, $\tan(x)$, а також $\operatorname{acos}(x)$, $\operatorname{asin}(x)$, $\operatorname{atan}(x)$, $\operatorname{acosh}(x)$, $\operatorname{asinh}(x)$, $\operatorname{atanh}(x)$.

Детальніше про кожну з них можна почитати у документації за допомогою команди `help(function)`.

2.6 Введення-виведення даних

Матеріали розділу у процесі підготовки.

Chapter 3

Базові конструкції мови R: типи та структури даних. Частина 2

План

- Набори даних
 - Вектори (vectors)
 - Поняття та спосіб представлення
 - Оголошення векторів
 - Операції над векторами
-

3.1 Набори даних

Матеріали розділу у процесі підготовки.

3.2 Вектори (vectors)

3.2.1 Поняття та спосіб представлення

Вектори є найпростішим способом представлення колекції даних. З своїм змістом **вектор** - це послідовність однорідних елементів. Якщо ж говорити

про мову програмування, то **вектор** - це послідовність елементів одного типу, що розміщені за деяким порядком (індексом).

Вектор прийнято позначати, як $x = (x_1, x_2, \dots, x_n)$, де x - назва вектора, n - кількість елементів вектора,

3.2.2 Оголошення векторів

Вектор - базовий тип даних у R, що дозволяє записати колекцію елементів одного типу за допомогою `c()` або без нього, якщо це послідовність значень.

Примітка. По суті функція `c()` дозволяє об'єднати кілька векторів.

Розглянемо для прикладу звичайну змінну x :

```
x <- 10
```

По своїй суті x у даному випадку є вектором, що складається з одного значення 10. Ми можемо також записати кілька елементів у змінну x :

```
x <- c(1, 2, 2.5, 3)
x
```

```
## [1] 1.0 2.0 2.5 3.0
```

Елементами вектора можуть бути значення будь якого типу: `numeric`, `character`, `logical` тощо:

```
v1 <- c(1, 3, 4, 6, 7)
v2 <- c(T, F, F, T, F)
v3 <- c("Hello", "my", "friend", "!")
```

Елементами вектора також послідовності, створені на за допомогою функцій `rep()`, `seq()` та оператора `::`:

```
vtr <- 2:7
vtr
```

```
## [1] 2 3 4 5 6 7
```

```
vtr <- 7:2
vtr
```

```
## [1] 7 6 5 4 3 2
```

Якщо є потреба об'єднати кілька векторів, скористайтеся функцією `c()`:

```
x <- 2:3
y <- c(4,6,9)
z <- c(x, y, 10:12, 100)
z
```

```
## [1] 2 3 4 6 9 10 11 12 100
```

Переглянути коротку описову статистику по вектору можна за допомогою функції `summary()`:

```
summary(z)
```

```
##      Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##      2.00   4.00   9.00  17.44  11.00  100.00
```

3.2.3 Операції над векторами

Chapter 4

4. Збір та читання інформації з різноманітних джерел: файли, веб-сторінки, бази R

План

- Загальний опис структури даних
 - Опис набору даних TelecomUsers
 - CSV: поняття, читання та запис
 - Excel (xlsx): поняття, читання та запис
 - XML: поняття, читання та запис
-

4.1 Загальний опис структури даних

Матеріали розділу у процесі підготовки.

Усі дані, що використовуються для наукових досліджень можна розділити на 3 блоки:

- ☒ Структуровані
- ☒ Слабоструктуровані

☒ Неструктуровані

Структуровані дані зазвичай зрозумілі для користувача, сформовані відповідно до певних вимог та дозволяють швидко їх вивчити, без додаткових процедур підготовки.

Важливою для нас характеристикою **структурованих даних** є те, що вони досить просто піддаються машинній обробці, аналізу та візуалізації.

Слабоструктуровані дані часто можуть бути сприйняті людиною, проте форма їх подачі не дозволяє швидко проаналізувати її машиною.

Найпростішим прикладом структурованих даних є табличні дані, наприклад, в Microsoft Excel, де кожен рядок - це одне спостереження, а кожен стовпець - це одна характеристика.

Слабоструктуровані дані можуть бути перетворені до структурованих за допомогою підготовлених спеціалістом з по роботі з даними алгоритмом. Подібний етап зазвичай потребує детального аналізу полів, форми подання, визначення шаблонів помилок у даних тощо.

Неструктуровані дані

4.2 Опис набору даних Telecom Users

Опис інформації про набір даних взято з сервісу [kaggle.com](https://www.kaggle.com/radmirzosimov/telecom-users-dataset). Оригінальний набір розміщений за адресою <https://www.kaggle.com/radmirzosimov/telecom-users-dataset>.

Призначення:

Описаний датасет призначений для практикування спеціалістами з машинного навчання розв'язування задач класифікації.

Опис:

Any business wants to maximize the number of customers. To achieve this goal, it is important not only to try to attract new ones, but also to retain existing ones. Retaining a client will cost the company less than attracting a new one. In addition, a new client may be weakly interested in business services and it will be difficult to work with him, while old clients already have the necessary data on interaction with the service.

Accordingly, predicting the churn, we can react in time and try to keep the client who wants to leave. Based on the data about the services that the client uses, we can make him a special offer, trying to change his decision to leave the operator. This will make the task of retention easier to implement than the task of attracting new users, about which we do not know anything yet.

You are provided with a dataset from a telecommunications company. The data contains information about almost six thousand users, their demographic characteristics, the services they use, the duration of using the operator's services, the method of payment, and the amount of payment.

The task is to analyze the data and predict the churn of users (to identify people who will and will not renew their contract). The work should include the following mandatory items:

1. Description of the data (with the calculation of basic statistics);
2. Research of dependencies and formulation of hypotheses;
3. Building models for predicting the outflow (with justification for the choice of a particular model)
4. based on tested hypotheses and identified relationships;
4. Comparison of the quality of the obtained models.

Опис полів набору даних:

- ☒ customerID - customer id
- ☒ gender - client gender (male / female)
- ☒ SeniorCitizen - is the client retired (1, 0)
- ☒ Partner - is the client married (Yes, No)
- ☒ tenure - how many months a person has been a client of the company
- ☒ PhoneService - is the telephone service connected (Yes, No)
- ☒ MultipleLines - are multiple phone lines connected (Yes, No, No phone service)
- ☒ InternetService - client's Internet service provider (DSL, Fiber optic, No)
- ☒ OnlineSecurity - is the online security service connected (Yes, No, No internet service)
- ☒ OnlineBackup - is the online backup service activated (Yes, No, No internet service)
- ☒ DeviceProtection - does the client have equipment insurance (Yes, No, No internet service)
- ☒ TechSupport - is the technical support service connected (Yes, No, No internet service)
- ☒ StreamingTV - is the streaming TV service connected (Yes, No, No internet service)
- ☒ StreamingMovies - is the streaming cinema service activated (Yes, No, No internet service)
- ☒ Contract - type of customer contract (Month-to-month, One year, Two year)
- ☒ PaperlessBilling - whether the client uses paperless billing (Yes, No)
- ☒ PaymentMethod - payment method (Electronic check, Mailed check, Bank transfer (automatic), Credit card (automatic))
- ☒ MonthlyCharges - current monthly payment

- ☒ TotalCharges - the total amount that the client paid for the services for the entire time
- ☒ Churn - whether there was a churn (Yes or No)

4.3 Імпорт даних засобами RStudio

RStudio має ряд засобів для

4.4 CSV-файли: читання, запис

CSV - це тип файлів, у якому інформація розділена комами (Comma Separated Values). CSV є досить зручним форматом даних для передачі між різними машинами, адже по суті є текстовим файлом і дозволяє легко його зчитати.

Примітка. Наспереді кома не завжди є роздільником CSV-файлів. Це можуть бути і інші символи.

```
# lets check current working directory to write correct files path
getwd()
```

```
## [1] "E:/Repos/YuRa/r-science-dev"
```

You can use / or \ for writing correct path in R. For example:

```
path = "d:/projects/file.csv"
path = "d:\\projects\\file.csv"
```

To combine path use paste() or paste0() functions

```
work_dir = getwd()
work_dir
```

```
## [1] "E:/Repos/YuRa/r-science-dev"
```

```
file_name = "temp_file.csv"
file_path = paste0(work_dir, "/", file_name)
file_path
```

```
## [1] "E:/Repos/YuRa/r-science-dev/temp_file.csv"
```

```
file_path = paste(work_dir, file_name, sep = "/")
file_path
```

```
## [1] "E:/Repos/YuRa/r-science-dev/temp_file.csv"
```

There are few methods for reading/writing csv in base package:

- ☒ `read.csv()`, `write.csv` - default data separator is `,`, decimal is separator `.`
- ☒ `read.csv2()`, `write.csv2` - default data separator is `;`, decimal is separator `,`.

Before using any new function check it usage information with `help(function_name)` or `?function_name,example:?read.csv`.

You can read (current data set has NA values as example, there are no NA in original dataset):

```
data <- read.csv2("data/telecom_users.csv") # default reading
```

```
data <- read.csv2("data/telecom_users.csv",
  sep = ",", # comma not only possible separator
  dec = ".", # decimal separator can be different
  na.strings = c("", "NA", "NULL")) # you can define NA values
```

```
str(data) # check data structure / types / values
```

```
## 'data.frame':    5986 obs. of  22 variables:
## $ X              : int  1869 4528 6344 6739 432 2215 5260 6001 1480 5137 ...
## $ customerID     : chr   "7010-BRBUU" "9688-YGXVR" "9286-DOJGF" "6994-KERXL" ...
## $ gender          : chr   "Male" "Female" "Female" "Male" ...
## $ SeniorCitizen   : int    0 0 1 0 0 0 0 0 0 1 ...
## $ Partner         : chr   "Yes" "No" "Yes" "No" ...
## $ Dependents      : chr   "Yes" "No" "No" "No" ...
## $ tenure          : int   72 44 38 4 2 70 33 1 39 55 ...
## $ PhoneService    : chr   "Yes" "Yes" "Yes" "Yes" ...
## $ MultipleLines    : chr   "Yes" "No" "Yes" "No" ...
## $ InternetService : chr   "No" "Fiber optic" "Fiber optic" "DSL" ...
## $ OnlineSecurity  : chr   "No internet service" "No" "No" "No" ...
## $ OnlineBackup    : chr   "No internet service" "Yes" "No" "No" ...
## $ DeviceProtection: chr   "No internet service" "Yes" "No" "No" ...
## $ TechSupport     : chr   "No internet service" "No" "No" "No" ...
## $ StreamingTV     : chr   "No internet service" "Yes" "No" "No" ...
```

```
## $ StreamingMovies : chr "No internet service" "No" "No" "Yes" ...
## $ Contract : chr "Two year" "Month-to-month" "Month-to-month" "Month-to-month"
## $ PaperlessBilling: chr "No" "Yes" "Yes" "Yes" ...
## $ PaymentMethod : chr "Credit card (automatic)" "Credit card (automatic)" "Bank transfer (automatic)"
## $ MonthlyCharges : num 24.1 88.2 75 55.9 53.5 ...
## $ TotalCharges : num 1735 3973 2870 238 120 ...
## $ Churn : chr "No" "No" "Yes" "No" ...
```

```
head(data) # top 6 rows, use n = X, for viewing top X lines
```

```
##      X customerID gender SeniorCitizen Partner Dependents tenure PhoneService
## 1 1869 7010-BRBUU Male                0      Yes           Yes      72          Yes
## 2 4528 9688-YGXVR Female              0      No            No      44          Yes
## 3 6344 9286-DOJGF Female              1      Yes           No      38          Yes
## 4 6739 6994-KERXL Male                0      No            No       4          Yes
## 5 432 2181-UAESM Male                0      No            No       2          Yes
## 6 2215 4312-GVYNH Female              0      Yes           No     70          No
##      MultipleLines InternetService      OnlineSecurity      OnlineBackup
## 1              Yes           No No internet service No internet service
## 2              No      Fiber optic                No              Yes
## 3              Yes      Fiber optic                No              No
## 4              No      DSL                      No              No
## 5              No      DSL                      Yes              No
## 6 No phone service      DSL                      Yes              No
##      DeviceProtection      TechSupport      StreamingTV
## 1 No internet service No internet service No internet service
## 2              Yes                No              Yes
## 3              No                No              No
## 4              No                No              No
## 5              Yes                No              No
## 6              Yes              Yes              No
##      StreamingMovies      Contract PaperlessBilling      PaymentMethod
## 1 No internet service      Two year                No      Credit card (automatic)
## 2              No Month-to-month                Yes      Credit card (automatic)
## 3              No Month-to-month                Yes Bank transfer (automatic)
## 4              Yes Month-to-month                Yes      Electronic check
## 5              No Month-to-month                No      Electronic check
## 6              Yes      Two year                Yes Bank transfer (automatic)
##      MonthlyCharges TotalCharges Churn
## 1          24.10      1734.65      No
## 2          88.15      3973.20      No
## 3          74.95      2869.85      Yes
## 4          55.90       238.50      No
## 5          53.45       119.50      No
## 6          49.85      3370.20      No
```

```
is.data.frame(data) # if data is data.frame
```

```
## [1] TRUE
```

```
any(is.na(data)) # if dataframe contains any NA values
```

```
## [1] TRUE
```

```
apply(is.na(data), 2, any) #check NA by 2nd dimension - columns
```

```
##           X      customerID      gender  SeniorCitizen
##          FALSE          FALSE          FALSE          FALSE
##      Partner      Dependents      tenure  PhoneService
##          FALSE          FALSE          FALSE          FALSE
## MultipleLines  InternetService  OnlineSecurity  OnlineBackup
##          FALSE          FALSE          FALSE          FALSE
## DeviceProtection  TechSupport  StreamingTV  StreamingMovies
##          FALSE          FALSE          FALSE          FALSE
##      Contract  PaperlessBilling  PaymentMethod  MonthlyCharges
##          FALSE          FALSE          FALSE          TRUE
##      TotalCharges      Churn
##          TRUE          FALSE
```

Check MonthlyCharges: TRUE and TotalCharges: TRUE. These columns has NA-values.

Let's replace them with mean:

```
data[is.na(data$TotalCharges), "TotalCharges"] <- mean(data$TotalCharges, na.rm = T)
data[is.na(data$MonthlyCharges), "MonthlyCharges"] <- mean(data$MonthlyCharges, na.rm = T)
any(is.na(data)) # check for NA
```

```
## [1] FALSE
```

You can write data with `write.csv()`, `write.csv2()` from base package.

```
write.csv(data, file = "data/cleaned_data.csv", row.names = F)
# by default row.names = TRUE and file will contains first columns with row numbers 1,2, ..., N
```

One more useful package is `readr`. Examples of using:

```
#install.packages("readr")
library(readr)
data <- read_csv(file = "data/telecom_users.csv", ... )
data <- read_csv2(file = "data/telecom_users.csv", ... )
```

4.5 Excel (xlsx): читання, запис

There are many packages to read/write MS Excel files. `xlsx` one of the most useful.

```
#install.packages("xlsx") install before use it
library(xlsx)
```

```
any(grepl("xlsx", installed.packages())) # check if package installed
```

```
## [1] TRUE
```

?read.xlsx - review package functions and params

Let's read the data `telecom_users.xlsx`:

```
data <- read.xlsx("data/telecom_users.xlsx", sheetIndex = 1)
# sheetIndex = 1 - select sheet to read, or use sheetName = "sheet1" to read by Name

# You can also use startRow, endRow and other params to define how much data read
data <- read.xlsx("data/telecom_users.xlsx", sheetIndex = 1, endRow = 100)
```

Let's replace Churn values Yes/No by 1/0:

```
head(data$Churn)
```

```
## [1] "No" "No" "Yes" "No" "No" "No"
```

```
data$Churn <- ifelse(data$Churn == "Yes", 1, 0)
```

```
head(data$Churn)
```

```
## [1] 0 0 1 0 0 0
```

Write final data to csv:

```
write.xlsx(data, file = "data/final_telecom_data.xlsx")
```

4.5.1 Task 2.1

Download from kaggle.com and read dataset Default_Fin.csv: <https://www.kaggle.com/kmlidas/loan-default-prediction>

Description:

This is a synthetic dataset created using actual data from a financial institution. The data has been modified to remove identifiable features and the numbers transformed to ensure they do not link to original source (financial institution).

This is intended to be used for academic purposes for beginners who want to practice financial analytics from a simple financial dataset.

- ☒ Index - This is the serial number or unique identifier of the loan taker
- ☒ Employed - This is a Boolean 1= employed 0= unemployed
- ☒ Bank.Balance - Bank Balance of the loan taker
- ☒ Annual.Salary - Annual salary of the loan taker

- ☒ Defaulted - This is a Boolean 1= defaulted 0= not defaulted

1. Check what columns has missing values
2. Count default and non-default clients / and parts of total clients in %
3. Count Employed clients
4. Count Employed Default clients
5. Average salary by Employed clients
6. Rename columns to "id", "empl", "balance", "salary", "default"

4.5.2 Solution for Task 2.1

```
data <- read.csv("data/Default_Fin.csv")
head(data)
```

```
##   Index Employed Bank.Balance Annual.Salary Defaulted.
## 1     1       1     8754.36    532339.56          0
## 2     2       0     9806.16    145273.56          0
## 3     3       1    12882.60    381205.68          0
## 4     4       1     6351.00    428453.88          0
## 5     5       1     9427.92    461562.00          0
## 6     6       0    11035.08     89898.72          0
```

1. Check what columns has missing values

```
any(is.na(data))
```

```
## [1] FALSE
```

2. Count default and non-default clients / and parts of total clients in %

```
def_count <- nrow(data[data$Defaulted. == 1, ])
no_def_count <- nrow(data[data$Defaulted. == 0, ])
def_count
```

```
## [1] 333
```

```
no_def_count
```

```
## [1] 9667
```

```
def_count / nrow(data) * 100 # part defaults
```

```
## [1] 3.33
```

```
no_def_count / nrow(data) * 100 # part non-defaults
```

```
## [1] 96.67
```

3. Count Employed clients

```
empl <- data[data$Employed == 1, ]
nrow(empl)
```

```
## [1] 7056
```

4. Count Employed Default clients

```
empl <- data[data$Employed == 1 & data$Defaulted. == 1, ]
nrow(empl)
```

```
## [1] 206
```

5. Average salary by Employed clients


```
empl <- data[data$Employed == 1, ]
mean(empl$Annual.Salary)
```

```
## [1] 480143.4
```

6. Rename columns to "id", "empl", "balance", "salary", "default":

```
colnames(data) <- c("id", "empl", "balance", "salary", "default")
head(data)
```

```
##   id empl balance salary default
## 1  1   1  8754.36 532339.56      0
## 2  2   0  9806.16 145273.56      0
## 3  3   1 12882.60 381205.68      0
## 4  4   1  6351.00 428453.88      0
## 5  5   1  9427.92 461562.00      0
## 6  6   0 11035.08  89898.72      0
```

4.6 XML: читання, запис

XML - eXtensible Markup Language.

For our example we will use data from data/employees.xml. File contains records with info:

```
<RECORDS>
  <EMPLOYEE>
    <ID>1</ID>
    <NAME>Rick</NAME>
    <SALARY>623.3</SALARY>
    <STARTDATE>1/1/2012</STARTDATE>
    <DEPT>IT</DEPT>
  </EMPLOYEE>
  ...
</RECORDS>
```

```
#install.packages("XML")
library("XML")
#install.packages("methods")
library("methods")
```

```
result <- xmlParse(file = "data/employees.xml")
print(result)
```

```
## <?xml version="1.0"?>
## <RECORDS>
##   <EMPLOYEE>
##     <ID>1</ID>
##     <NAME>Rick</NAME>
##     <SALARY>623.3</SALARY>
##     <STARTDATE>1/1/2012</STARTDATE>
##     <DEPT>IT</DEPT>
##   </EMPLOYEE>
##   <EMPLOYEE>
##     <ID>2</ID>
##     <NAME>Dan</NAME>
##     <SALARY>515.2</SALARY>
##     <STARTDATE>9/23/2013</STARTDATE>
##     <DEPT>Operations</DEPT>
##   </EMPLOYEE>
##   <EMPLOYEE>
##     <ID>3</ID>
##     <NAME>Michelle</NAME>
##     <SALARY>611</SALARY>
##     <STARTDATE>11/15/2014</STARTDATE>
##     <DEPT>IT</DEPT>
##   </EMPLOYEE>
##   <EMPLOYEE>
##     <ID>4</ID>
##     <NAME>Ryan</NAME>
##     <SALARY>729</SALARY>
##     <STARTDATE>5/11/2014</STARTDATE>
##     <DEPT>HR</DEPT>
##   </EMPLOYEE>
##   <EMPLOYEE>
##     <ID>5</ID>
##     <NAME>Gary</NAME>
##     <SALARY>843.25</SALARY>
##     <STARTDATE>3/27/2015</STARTDATE>
##     <DEPT>Finance</DEPT>
##   </EMPLOYEE>
##   <EMPLOYEE>
##     <ID>6</ID>
##     <NAME>Nina</NAME>
##     <SALARY>578</SALARY>
##     <STARTDATE>5/21/2013</STARTDATE>
```

```
##      <DEPT>IT</DEPT>
##    </EMPLOYEE>
##  <EMPLOYEE>
##    <ID>7</ID>
##    <NAME>Simon</NAME>
##    <SALARY>632.8</SALARY>
##    <STARTDATE>7/30/2013</STARTDATE>
##    <DEPT>Operations</DEPT>
##  </EMPLOYEE>
##  <EMPLOYEE>
##    <ID>8</ID>
##    <NAME>Guru</NAME>
##    <SALARY>722.5</SALARY>
##    <STARTDATE>6/17/2014</STARTDATE>
##    <DEPT>Finance</DEPT>
##  </EMPLOYEE>
## </RECORDS>
##
```

```
rootnode <- xmlRoot(result) # reading rootnode of xml document
rootnode[[1]] # reading first record
```

```
## <EMPLOYEE>
##   <ID>1</ID>
##   <NAME>Rick</NAME>
##   <SALARY>623.3</SALARY>
##   <STARTDATE>1/1/2012</STARTDATE>
##   <DEPT>IT</DEPT>
## </EMPLOYEE>
```

```
rootnode[[1]][[2]] # reading first record in root node and second tag, its <NAME>
```

```
## <NAME>Rick</NAME>
```

```
xmldataframe <- xmlToDataFrame("data/employees.xml")
xmldataframe
```

```
##   ID   NAME SALARY STARTDATE   DEPT
## 1  1   Rick  623.3   1/1/2012     IT
## 2  2    Dan  515.2  9/23/2013 Operations
## 3  3 Michelle   611 11/15/2014     IT
## 4  4    Ryan   729  5/11/2014     HR
## 5  5    Gary 843.25  3/27/2015  Finance
## 6  6    Nina   578  5/21/2013     IT
## 7  7   Simon  632.8  7/30/2013 Operations
## 8  8    Guru  722.5  6/17/2014  Finance
```

4.7 JSON and API

JSON (JavaScript Object Notation) is a lightweight data-interchange format. It is easy for humans to read and write. It is easy for machines to parse and generate. It is based on a subset of the JavaScript Programming Language Standard.

API is the acronym for Application Programming Interface, which is a software intermediary that allows two applications to talk to each other.

One of the most popular packages for json is jsonlite.

```
#install.packages("jsonlite")
library(jsonlite)
```

Let's use reading information about BTC and USDT crypro currencies from Binance

```
market = 'BTCUSDT'
interval = '1h'
limit = 100

url <- paste0(url = "https://api.binance.com/api/v3/klines?symbol=", market, "&interval=", interval, "&limit=", limit)
print(url) # complete request URL
```

```
## [1] "https://api.binance.com/api/v3/klines?symbol=BTCUSDT&interval=1h&limit=100"
```

On the next stage you need use fromJSON() function to get data.

More details about requests to Binance at <https://github.com/binance/binance-spot-api-docs/blob/master/rest-api.md#klinecandlestick-data>

If you enter url value at browser response is going to be like this:

```
[
  [
    1499040000000, // Open time
    "0.01634790", // Open
    "0.80000000", // High
    "0.01575800", // Low
    "0.01577100", // Close
    "148976.11427815", // Volume
    1499644799999, // Close time
    "2434.19055334", // Quote asset volume
    308, // Number of trades
    "1756.87402397", // Taker buy base asset volume
  ]
]
```

```

    "28.46694368",      // Taker buy quote asset volume
    "17928899.62484339" // Ignore.
  ]
]

```

```

data <- fromJSON(url) # get json and transform it to list()
data <- data[, 1:7] # let's left only 1:7 columns (from Open time to Close time)
head(data)

```

```

##      [,1]      [,2]      [,3]      [,4]
## [1,] "1621051200000" "49627.02000000" "49861.20000000" "49150.92000000"
## [2,] "1621054800000" "49322.03000000" "49720.37000000" "49086.75000000"
## [3,] "1621058400000" "49668.92000000" "49700.00000000" "48628.72000000"
## [4,] "1.621062e+12" "48972.00000000" "49058.89000000" "48400.00000000"
## [5,] "1621065600000" "48469.71000000" "49072.49000000" "48203.91000000"
## [6,] "1621069200000" "48826.06000000" "49298.59000000" "48350.00000000"
##      [,5]      [,6]      [,7]
## [1,] "49322.03000000" "2921.96110400" "1621054799999"
## [2,] "49666.46000000" "2777.40806400" "1621058399999"
## [3,] "48971.99000000" "4475.98343900" "1621061999999"
## [4,] "48469.28000000" "4586.85031500" "1621065599999"
## [5,] "48826.08000000" "3843.62357800" "1621069199999"
## [6,] "48640.87000000" "3775.30603900" "1621072799999"

```

```

typeof(data) # check data type

```

```

## [1] "character"

```

```

data <- as.data.frame(data) # convert to dataframe
head(data)

```

```

##      V1      V2      V3      V4      V5
## 1 1621051200000 49627.02000000 49861.20000000 49150.92000000 49322.03000000
## 2 1621054800000 49322.03000000 49720.37000000 49086.75000000 49666.46000000
## 3 1621058400000 49668.92000000 49700.00000000 48628.72000000 48971.99000000
## 4 1.621062e+12 48972.00000000 49058.89000000 48400.00000000 48469.28000000
## 5 1621065600000 48469.71000000 49072.49000000 48203.91000000 48826.08000000
## 6 1621069200000 48826.06000000 49298.59000000 48350.00000000 48640.87000000
##      V6      V7
## 1 2921.96110400 1621054799999
## 2 2777.40806400 1621058399999
## 3 4475.98343900 1621061999999
## 4 4586.85031500 1621065599999
## 5 3843.62357800 1621069199999
## 6 3775.30603900 1621072799999

```

```
# fix columns names
colnames(data) <- c("Open_time", "Open", "High", "Low", "Close", "Volume", "Close_time")
head(data) # looks better, but columns are characters still
```

```
##      Open_time      Open      High      Low      Close
## 1 1621051200000 49627.02000000 49861.20000000 49150.92000000 49322.03000000
## 2 1621054800000 49322.03000000 49720.37000000 49086.75000000 49666.46000000
## 3 1621058400000 49668.92000000 49700.00000000 48628.72000000 48971.99000000
## 4 1.621062e+12 48972.00000000 49058.89000000 48400.00000000 48469.28000000
## 5 1621065600000 48469.71000000 49072.49000000 48203.91000000 48826.08000000
## 6 1621069200000 48826.06000000 49298.59000000 48350.00000000 48640.87000000
##      Volume      Close_time
## 1 2921.96110400 1621054799999
## 2 2777.40806400 1621058399999
## 3 4475.98343900 1621061999999
## 4 4586.85031500 1621065599999
## 5 3843.62357800 1621069199999
## 6 3775.30603900 1621072799999
```

```
is.numeric(data[,1]) # check 1st column type is numeric
```

```
## [1] FALSE
```

```
is.numeric(data[,2]) # check 2nd column type is numeric
```

```
## [1] FALSE
```

```
data <- as.data.frame(sapply(data, as.numeric)) # convert all columns to numeric
head(data) # good, its double now
```

```
##      Open_time      Open      High      Low      Close      Volume      Close_time
## 1 1.621051e+12 49627.02 49861.20 49150.92 49322.03 2921.961 1.621055e+12
## 2 1.621055e+12 49322.03 49720.37 49086.75 49666.46 2777.408 1.621058e+12
## 3 1.621058e+12 49668.92 49700.00 48628.72 48971.99 4475.983 1.621062e+12
## 4 1.621062e+12 48972.00 49058.89 48400.00 48469.28 4586.850 1.621066e+12
## 5 1.621066e+12 48469.71 49072.49 48203.91 48826.08 3843.624 1.621069e+12
## 6 1.621069e+12 48826.06 49298.59 48350.00 48640.87 3775.306 1.621073e+12
```

Final stage is to convert `Open_time` and `Close_time` to dates.

```
data$Open_time <- as.POSIXct(data$Open_time/1e3, origin = '1970-01-01')
data$Close_time <- as.POSIXct(data$Close_time/1e3, origin = '1970-01-01')
```

```
head(data)
```

```
##           Open_time      Open      High      Low      Close      Volume
## 1 2021-05-15 07:00:00 49627.02 49861.20 49150.92 49322.03 2921.961
## 2 2021-05-15 08:00:00 49322.03 49720.37 49086.75 49666.46 2777.408
## 3 2021-05-15 09:00:00 49668.92 49700.00 48628.72 48971.99 4475.983
## 4 2021-05-15 10:00:00 48972.00 49058.89 48400.00 48469.28 4586.850
## 5 2021-05-15 11:00:00 48469.71 49072.49 48203.91 48826.08 3843.624
## 6 2021-05-15 12:00:00 48826.06 49298.59 48350.00 48640.87 3775.306
##           Close_time
## 1 2021-05-15 07:59:59
## 2 2021-05-15 08:59:59
## 3 2021-05-15 09:59:59
## 4 2021-05-15 10:59:59
## 5 2021-05-15 11:59:59
## 6 2021-05-15 12:59:59
```

```
tail(data) # check last records
```

```
##           Open_time      Open      High      Low      Close      Volume
## 95 2021-05-19 05:00:00 40843.53 41299.98 40275.00 40325.01 8604.976
## 96 2021-05-19 06:00:00 40324.57 41049.99 40125.00 40591.15 9124.063
## 97 2021-05-19 07:00:00 40591.99 40846.97 38604.93 39307.66 17716.266
## 98 2021-05-19 08:00:00 39307.66 39941.00 38934.69 39200.01 12574.524
## 99 2021-05-19 09:00:00 39200.00 39878.33 39102.00 39378.54 8295.853
## 100 2021-05-19 10:00:00 39383.29 39680.00 38550.00 39433.83 5800.859
##           Close_time
## 95 2021-05-19 05:59:59
## 96 2021-05-19 06:59:59
## 97 2021-05-19 07:59:59
## 98 2021-05-19 08:59:59
## 99 2021-05-19 09:59:59
## 100 2021-05-19 10:59:59
```

4.8 Google Services

Chapter 5

5. Прийоми маніпулювання даними (з використанням можливостей бібліотеки dplyr)

План

- Загальний опис структури даних
- Опис набору даних TelecomUsers
- Імпорт даних засобами RStudio
- CSV: поняття, читання та запис
- XLSX: поняття, читання та запис
- JSON: поняття, читання та запис
- API: поняття, читання
- HTML: поняття, читання
- PDF: поняття, читання
- Сервіси Google
 - Spreadsheets
 - Keywords

5.1 Загальний опис структури даних

Матеріали розділу у процесі підготовки.

Усі дані, що використовуються для наукових досліджень можна розділити на 3 блоки:

- ☒ Структуровані
- ☒ Слабоструктуровані
- ☒ Неструктуровані

Структуровані дані зазвичай зрозумілі для користувача, сформовані відповідно до певних вимог та дозволяють швидко їх вивчити, без додаткових процедур підготовки.

Важливою для нас характеристикою **структурованих даних** є те, що вони досить просто піддаються машинній обробці, аналізу та візуалізації.

Слабоструктуровані дані часто можуть бути сприйняті людиною, проте форма їх подачі не дозволяє швидко проаналізувати її машиною.

Найпростішим прикладом структурованих даних є табличні дані, наприклад, в Microsoft Excel, де кожен рядок - це одне спостереження, а кожен стовпець - це одна характеристика.

Слабоструктуровані дані можуть бути перетворені до структурованих за допомогою підготовлених спеціалістом з по роботі з даними алгоритмом. Подібний етап зазвичай потребує детального аналізу полів, форми подання, визначення шаблонів помилок у даних тощо.

Неструктуровані дані

5.2 Опис набору даних Telecom Users

Опис інформації про набір даних взято з сервісу [kaggle.com](https://www.kaggle.com/radmirzosimov/telecom-users-dataset). Оригінальний набір розміщений за адресою <https://www.kaggle.com/radmirzosimov/telecom-users-dataset>.

Призначення:

Описаний датасет призначений для практикування спеціалістами з машинного навчання розв'язування задач класифікації.

Опис:

Any business wants to maximize the number of customers. To achieve this goal, it is important not only to try to attract new ones, but also to retain existing ones. Retaining a client will cost the company less than attracting a new one. In addition, a new client may be weakly interested in business services and it

will be difficult to work with him, while old clients already have the necessary data on interaction with the service.

Accordingly, predicting the churn, we can react in time and try to keep the client who wants to leave. Based on the data about the services that the client uses, we can make him a special offer, trying to change his decision to leave the operator. This will make the task of retention easier to implement than the task of attracting new users, about which we do not know anything yet.

You are provided with a dataset from a telecommunications company. The data contains information about almost six thousand users, their demographic characteristics, the services they use, the duration of using the operator's services, the method of payment, and the amount of payment.

The task is to analyze the data and predict the churn of users (to identify people who will and will not renew their contract). The work should include the following mandatory items:

1. Description of the data (with the calculation of basic statistics);
2. Research of dependencies and formulation of hypotheses;
3. Building models for predicting the outflow (with justification for the choice of a particular model)
4. based on tested hypotheses and identified relationships;
4. Comparison of the quality of the obtained models.

Опис полів набору даних:

- ☒ customerID - customer id
- ☒ gender - client gender (male / female)
- ☒ SeniorCitizen - is the client retired (1, 0)
- ☒ Partner - is the client married (Yes, No)
- ☒ tenure - how many months a person has been a client of the company
- ☒ PhoneService - is the telephone service connected (Yes, No)
- ☒ MultipleLines - are multiple phone lines connected (Yes, No, No phone service)
- ☒ InternetService - client's Internet service provider (DSL, Fiber optic, No)
- ☒ OnlineSecurity - is the online security service connected (Yes, No, No internet service)
- ☒ OnlineBackup - is the online backup service activated (Yes, No, No internet service)
- ☒ DeviceProtection - does the client have equipment insurance (Yes, No, No internet service)
- ☒ TechSupport - is the technical support service connected (Yes, No, No internet service)
- ☒ StreamingTV - is the streaming TV service connected (Yes, No, No internet service)

- ☒ StreamingMovies - is the streaming cinema service activated (Yes, No, No internet service)
- ☒ Contract - type of customer contract (Month-to-month, One year, Two year)
- ☒ PaperlessBilling - whether the client uses paperless billing (Yes, No)
- ☒ PaymentMethod - payment method (Electronic check, Mailed check, Bank transfer (automatic), Credit card (automatic))
- ☒ MonthlyCharges - current monthly payment
- ☒ TotalCharges - the total amount that the client paid for the services for the entire time
- ☒ Churn - whether there was a churn (Yes or No)

5.3 Імпорт даних засобами RStudio

RStudio має ряд засобів для

5.4 CSV-файли: читання,

CSV - це тип файлів, у якому інформація розділена комами (Comma Separated Values). CSV є досить зручним форматом даних для передачі між різними машинами, адже по суті є текстовим файлом і дозволяє легко його зчитати.

Примітка. Наспереді кома не завжди є роздільником CSV-файлів. Це можуть бути і інші символи.

Chapter 6

8. Представлення результатів аналізу даних.

Матеріали курсу у процесі наповнення. Слідкуйте за оновленнями

Chapter 7

7. Побудова та навчання математичних моделей, метрики оцінки їх якості.

Матеріали курсу у процесі наповнення. Слідкуйте за оновленнями

Chapter 8

Приклади задач та їх розв'язки

-
- Задачі
 - Послідовності та вектори
 - Функції
 - Рішення задач
 - Послідовності та вектори
 - Функції
-

8.1 Задачі

8.1.1 Послідовності

8.1.1.1 Задача

Написати програму, що обчислює $y = (x + 2)^2 + \ln(x)$, де x число з послідовності 100, 105, 110, ..., 200. Результат вивести у вигляді `data.frame` з колонками X та Y .

8.1.1.2 Задача

Написати програму, що обчислює $y = \frac{\sqrt{x+2}}{z}$, де x число з послідовності 10, 15, 20, ..., 100, а z - випадкові значення з діапазону $[-10, 10]$, що відповідає кількості елементів у x . Результат вивести у вигляді `data.frame` з колонками X , Z та Y .

8.1.1.3 Задача

Без використання спеціальних функцій написати програму, що сортує вектор за зростанням та спаданням. Елементи вектора є випадковими числами з діапазону $[10, 100)$. Кількість елементів у векторі 10.

8.1.1.4 Задача

Відсортувати вектор таким чином, щоб всі додатні елементи знаходилися на початку, а всі від'ємні – вкінці, і при цьому зберігся початковий порядок елементів в обох групах. Елементи вектора є випадковими числами з діапазону $[-100, 100]$. Кількість елементів у векторі 10.

Наприклад, якщо початковий вектор x складається з елементів:

```
x <- c(1, -5, 10, -8, -2, 5, 4, -9)
```

то після "сортування" він матиме вигляд:

```
## [1] 1 10 5 4 -5 -8 -2 -9
```

8.1.1.5 Задача

Задано натуральне число N (вводиться з клавіатури). Знайти суму його цифр.

8.1.1.6 Задача

Задано одновимірний масив. Знайти два серед його елементів, модуль різниці яких має найменше значення.

8.1.1.7 Задача

Знайти найбільший спільний дільник двох натуральних чисел, використавши алгоритм Евкліда. Алгоритм Евкліда полягає в наступному: від більшого числа віднімається менше до тих пір, поки вони не стануть рівними; отримане в результаті число і буде найбільшим спільним дільником.

8.1.1.8 Задача

Знайти мінімальний елемент серед тих елементів масиву A , які не є елементами масиву B .

8.1.1.9 Задача

Написати програму, що обчислює **середнє** значення серед парних елементів вектора. Елементи вектора генеруються випадковим чином у діапазоні $[1; 10)$. Кількість елементів вектора 10.

8.1.1.10 Задача

Написати програму, що знаходить середнє значення елементів вектора. Елементи вектора генеруються випадковим чином у діапазоні $[100; 200]$. Усі елементи вектора повинні бути кратними 7-ми.

Підзавдання 1. Генерацію випадкового числа кратного 7-ми винести в окрему функцію.

Підзавдання 2. Написати функцію, що приймає 1 значення size (довжина вектора) і повертає вектор цієї довжини.

```
func.genMod7 <- function() {  
  x <- 8  
  while(x %% 7 != 0) {  
    x <- sample(100:200, size = 1)  
  }  
  return(x)  
}  
  
func.genVectorMod7 <- function(size) {  
  v <- c()  
  
  for(i in 1:size) {  
    v <- c(v, func.genMod7())  
  }  
}
```

```

    }

    return(v)
}

vector <- func.genVectorMod7(100)
vector

```

```

##      [1] 175 133 126 105 147 189 168 182 112 161 140 196 147 140 175 182 140 175
##     [19] 196 119 196 105 161 189 126 182 182 189 161 189 140 126 112 168 133 126
##     [37] 175 119 182 175 154 175 196 175 196 154 161 140 147 175 182 147 182 175
##     [55] 182 133 196 175 168 175 140 140 182 133 119 182 147 119 175 147 147 133
##     [73] 105 147 182 126 119 189 196 168 133 133 119 133 182 196 133 126 189 147
##     [91] 175 168 175 182 147 126 119 161 126 154

```

8.1.1.11 Задача

Написати програму, що знаходить різницю між максимальним та мінімальним значенням елементів вектора, не використовуючи функції `min()` та `max()`. Елементи вектора генеруються випадковим чином у діапазоні `[10; 50]`.

8.1.1.12 Задача

Написати програму, що знаходить різницю між максимальним та мінімальним значенням елементів вектора, не використовуючи функції `min()` та `max()`. Елементи вектора генеруються випадковим чином у діапазоні `[10; 50]`.

8.1.1.13 Задача*

Написати програму, що дозволяє переставити значення деяких змінних без створення третьої змінної.

Наприклад, `a <- 10` та `b <- 25`, але після виконання деякого коду виведення має наступний вигляд:

```
print(a)
```

```
## [1] 25
```

```
print(b)
```

```
## [1] 10
```

8.1.1.14 Задача

Написати функцію, що...

8.1.1.15 Задача

Завантажити з сайту [kaggle.com](https://www.kaggle.com/ajaypalsinghlo/world-happiness-report-2021) інформацію про “індекс щастя” (<https://www.kaggle.com/ajaypalsinghlo/world-happiness-report-2021>). Завантажити дані в R.

8.1.2 Функції

8.1.2.1 Задача

Написати функцію, що обчислює для поданого вектора суму, середнє, медіану, мінімум та максимум. Результат роботи функції повертається у вигляді списку (*list*).

8.2 Рішення задач

8.2.1 Послідовності та вектори

Рішення до задачі 6.1.1.3:

```
x <- sample(10:100, size = 10)

for(j in 1:(length(x)-1)) {

  for(i in 1:(length(x)-1)) {

    if(x[i] > x[i+1]) {
      tmp = x[i]
      x[i] = x[i+1]
      x[i+1] = tmp
    }
  }
}
```

```
}  
}  
  
print(x)
```

```
## [1] 10 24 47 55 56 75 81 83 84 85
```

Рішення до задачі 6.1.1.4:

```
x <- sample(-100:100, size = 10)  
print("Vector before sort:")
```

```
## [1] "Vector before sort:"
```

```
print(x)
```

```
## [1] 40 -84 9 29 -9 -12 -82 -27 -62 -15
```

```
for(j in 1:(length(x)-1)) {  
  for(i in 1:(length(x)-1)) {  
    if(x[i] < 0 & x[i+1] > 0) {  
      tmp = x[i]  
      x[i] = x[i+1]  
      x[i+1] = tmp  
    }  
  }  
}  
print("Vector after sort:")
```

```
## [1] "Vector after sort:"
```

```
print(x)
```

```
## [1] 40 9 29 -84 -9 -12 -82 -27 -62 -15
```

Рішення до задачі 6.1.1.5:

```
#number <- as.numeric(readline(prompt = "      :"))
number <- 15783
sum <- 0

while(number > 0) {
  last_digit = number %% 10
  sum = sum + last_digit
  number = (number - last_digit) / 10
  print(paste0("Number: ", number, " | Sum: ", sum, " | Last: ", last_digit))
}
```

```
## [1] "Number: 1578 | Sum: 3 | Last: 3"
## [1] "Number: 157 | Sum: 11 | Last: 8"
## [1] "Number: 15 | Sum: 18 | Last: 7"
## [1] "Number: 1 | Sum: 23 | Last: 5"
## [1] "Number: 0 | Sum: 24 | Last: 1"
```

Рішення до задачі 6.1.1.9 (спосіб 1):

```
x <- sample(1:10, size = 4, replace = T)
x
```

```
## [1] 7 4 3 2
```

```
x_parni <- c()

for(i in 1:length(x)) {

  if(x[i] %% 2 == 0) {
    x_parni <- c(x_parni, x[i])
  }

}

mean(x_parni)
```

```
## [1] 3
```

Рішення до задачі 6.1.1.9 (спосіб 2):

```
x <- sample(1:10, size = 10, replace = T)
x
```

```
## [1] 8 10 4 3 2 4 5 8 6 3
```

```
sum <- 0
count <- 0

for(i in 1:length(x)) {

  if(x[i] %% 2 == 0) {
    sum = sum + x[i]
    count = count + 1
  }

}

mean_value = sum / count
mean_value
```

```
## [1] 6
```

8.2.2 Функції

Рішення до задачі 6.1.2.1:

```
x <- sample(10:100, size = 10)
print(x)
```

```
## [1] 66 72 39 31 57 78 44 46 21 33
```

```
vector.info <- function(vector) {
  x <- list()
  x$Sum <- sum(vector)
  x$Mean <- mean(vector)
  x$Median <- median(vector)
  x$Min <- min(vector)
  x$Max <- max(vector)
  return(x)
}

vector.info(x)
```



```
## $Sum
## [1] 487
##
## $Mean
## [1] 48.7
##
## $Median
## [1] 45
##
## $Min
## [1] 21
##
## $Max
## [1] 78
```

```
Sys.setlocale("LC_CTYPE", "ukrainian")
```


Джерела

Bibliography

(2021). *Anaconda. The World's Most Popular Data Science Platform*. Anaconda Inc, 206 379 Broadway Ave., Suite 310 New York, NY 10013, USA.

(2021). *Google Colaboratory*. Google LLC.

(2021). *RStudio official website*. RStudio, PBC, 250 Northern Ave, Boston, MA 02210.

(2021). *Visual Studio Code*. Microsoft.

(2021). *Visual Studio Community Edition*. Microsoft.

R Core Team (2020). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria.

А.Б. Шипунов, Е.М. Балдин, П.А. Волкова, А.И. Коробейников, С.А. Назарова, С.В. Петров, В.Г. Суфиянов. (2012). *Наглядная статистика. Используем R!* ДМК Пресс, Москва, Россия.