ОСНОВИ ПРОГНОЗУВАННЯ В R

Ігор Мірошниченко

2021-09-16

Table of Contents

[Основи прогнозування в R 2](#_Toc82680344)

[Передмова 2](#_Toc82680345)

[1 Вступ до R 2](#_Toc82680346)

[1.1 Встановлення R 2](#_Toc82680347)

[1.1.1 Встановлення на комп’ютер 3](#_Toc82680348)

[1.1.2 Використання хмарних сервісів 3](#_Toc82680349)

[1.2 RStudio 3](#_Toc82680350)

[1.3 Базові операції 6](#_Toc82680351)

[1.3.1 R та арифметика 6](#_Toc82680352)

[1.3.2 Коментарі до коду 7](#_Toc82680353)

[1.3.3 Пріоритети розрахунків 8](#_Toc82680354)

[1.3.4 Готові функції 8](#_Toc82680355)

[1.3.5 Вбудована документація 9](#_Toc82680356)

[1.3.6 Створення змінних 10](#_Toc82680357)

[1.3.7 Оператори порівняння 11](#_Toc82680358)

[1.4 Типи даних 11](#_Toc82680359)

[1.5 Структури даних 12](#_Toc82680360)

[1.5.1 Вектор 12](#_Toc82680361)

[1.5.2 Матриці 16](#_Toc82680362)

[1.5.3 Масиви 17](#_Toc82680363)

[1.5.4 Списки 18](#_Toc82680364)

[1.5.5 Дата фрейми 18](#_Toc82680365)

[1.6 Пакети в R 19](#_Toc82680366)

[1.6.1 Вбудовані пакети 20](#_Toc82680367)

[1.6.2 Встановлення пакетів з CRAN 20](#_Toc82680368)

[1.7 Завантаження даних в R 22](#_Toc82680369)

[1.8 Умовні конструкції 22](#_Toc82680370)

[1.9 Створення функцій в R 22](#_Toc82680371)

[1.10 Вступ у Tidyverse 22](#_Toc82680372)

[1.11 Створення звітів: R Markdown 22](#_Toc82680373)

[2 Візуалізація часових рядів 22](#_Toc82680374)

[3 Декомпозиція часових рядів 23](#_Toc82680375)

[4 Аналіз часового ряду 23](#_Toc82680376)

[5 Інструментарій прогнозування 23](#_Toc82680377)

[6 Регресійні моделі 23](#_Toc82680378)

[7 Експоненціальне згладжування 23](#_Toc82680379)

[8 ARIMA-моделі 23](#_Toc82680380)

[9 Prophet Facebook 23](#_Toc82680381)

[10 Кластеризація 23](#_Toc82680382)

# Основи прогнозування в R

*Ігор Мірошниченко*

*2021-09-16*

## Передмова

Наразі підручник в процесі розробки.

Якщо ви помітили неточності або помилки, будь-ласка напишіть мені: ihor.miroshnychenko@kneu.ua

# 1 Вступ до R

Мова програмування **R** — потужний інструмент, що широко використовується в різних сферах: статистика, візуалізація, наука про дані, машинне навчання, глибоке навчання тощо. Це безплатна система з відкритим кодом, з широкими можливостями, які досягаються внаслідок різноманіття пакетів (*packages*) доповнень. **R** повністю безплатний та доступний для встановлення на більшість операційних систем. Крім того, є можливість працювати у хмарних додатках. Разом з тим, **R** ідеально підходить для проведення економетричних досліджень. Вивчити **R** не складно і цей розділ підготовить Вас до базових операцій.

## 1.1 Встановлення R

Варіантів роботи з мовою програмування **R** існує досить багато. Розберемо основні.

### 1.1.1 Встановлення на комп’ютер

Для початку необхідно встановити саму мову програмування під свою операційну систему:

* [Windows](https://cran.r-project.org/bin/windows/base/)
* [MacOS](https://cran.r-project.org/bin/macosx/)
* [Linux](https://cran.rstudio.com/bin/linux/)

Для роботи в **R**, також слід встановити інтегроване середовище розробки (IDE) **RStudio**, що значно підвищує зручність, швидкість та ефективність роботи.

Щоб встановити **RStudio**, завантажте останню версію інсталятора для Вашої операційної системи з сайту <https://rstudio.com/products/rstudio/download/>

### 1.1.2 Використання хмарних сервісів

У випадках, коли не має можливості або бажання щось додатково встановлювати на ПК є декілька хмарних сервісів:

* [RStudio Cloud](https://rstudio.cloud/)
* [Google Colab](https://colab.research.google.com/#create=true&language=r)

## 1.2 RStudio

Я пропоную використовувати саме **RStudio**, оскільки це найзручніший інструмент для роботи з даними. При чому в ній можна працювати з різними мовами програмування, в тому числі й **Python**.

Якщо ваша операційна система — Windows, то після встановлення R та RStudio, на робочому столі у вас буде дві іконки:

 та 

Перша належати до самої мови програмування і не буде використовуватись в роботі, друга відноситься до RStudio і саме вона нам потрібна.

Після першого запуску RStudio ви побачите інтерфейс на рисунку 1.1

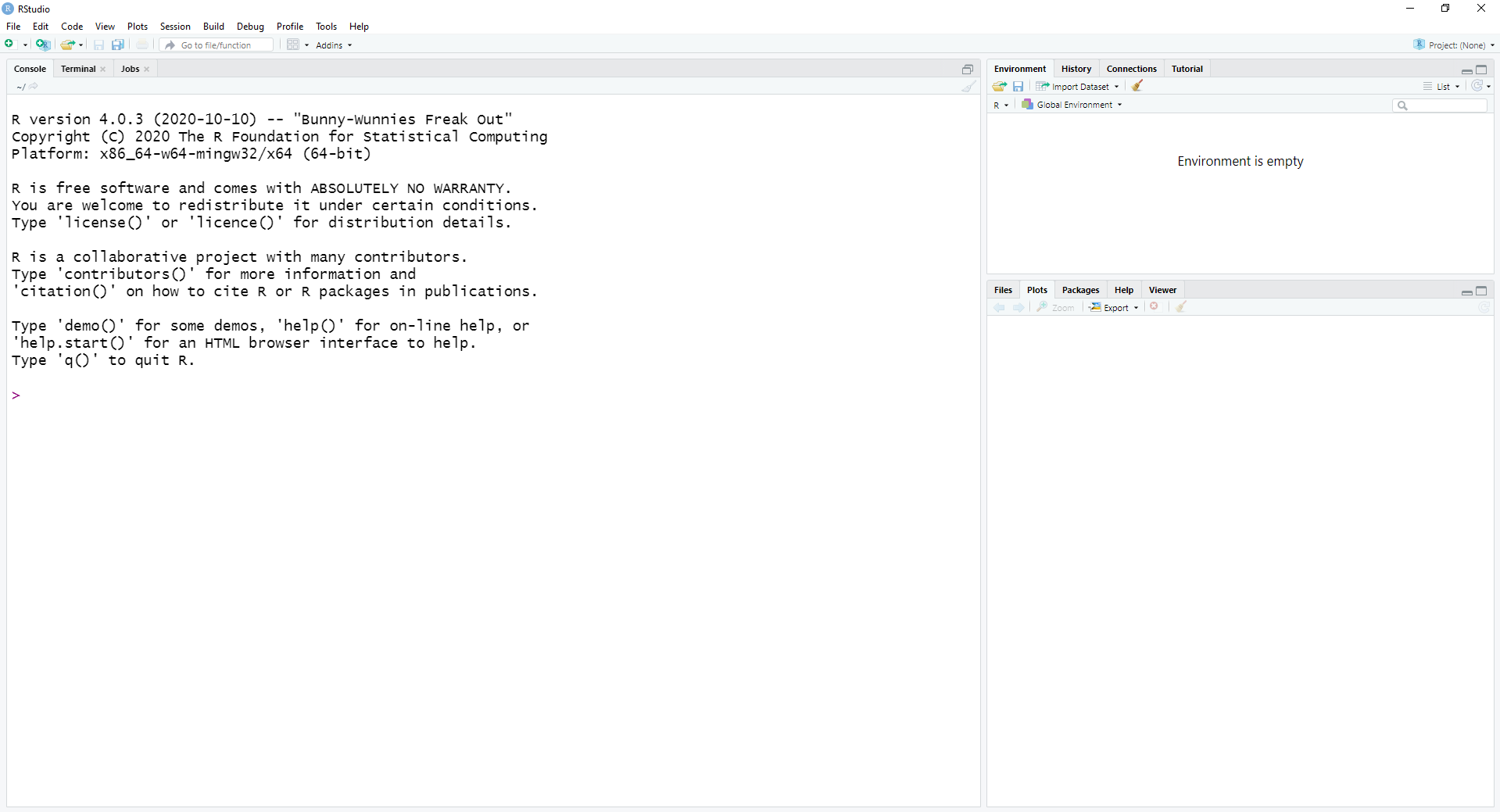


Рисунок 1.1: Базовий інтерфейс RStudio

Як правило код пишуть у скрипті (такий собі аналог текстового редактора), тож для його створення необхідно натиснути зелений хрестик в лівому верхньому куті та обрати пункт **R Script** або натиснути комбінацію клавіш **Ctrl+Shift+N**:

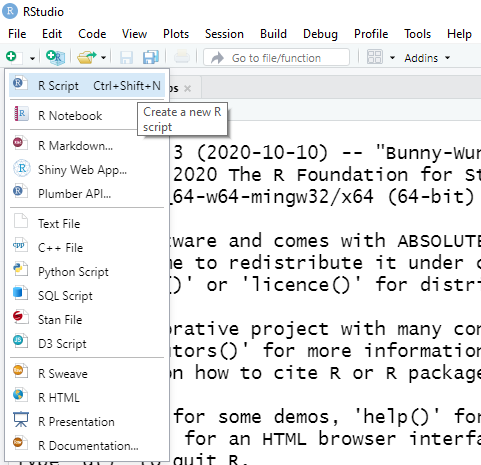


Рисунок 1.2: Створення скрипта в RStudio

Після чого, з’явиться додаткове вікно скрипту, де можна писати код, зберігати його та повертатися до нього за потреби.

Розберімо отриманий інтерфейс (рис. 1.3)

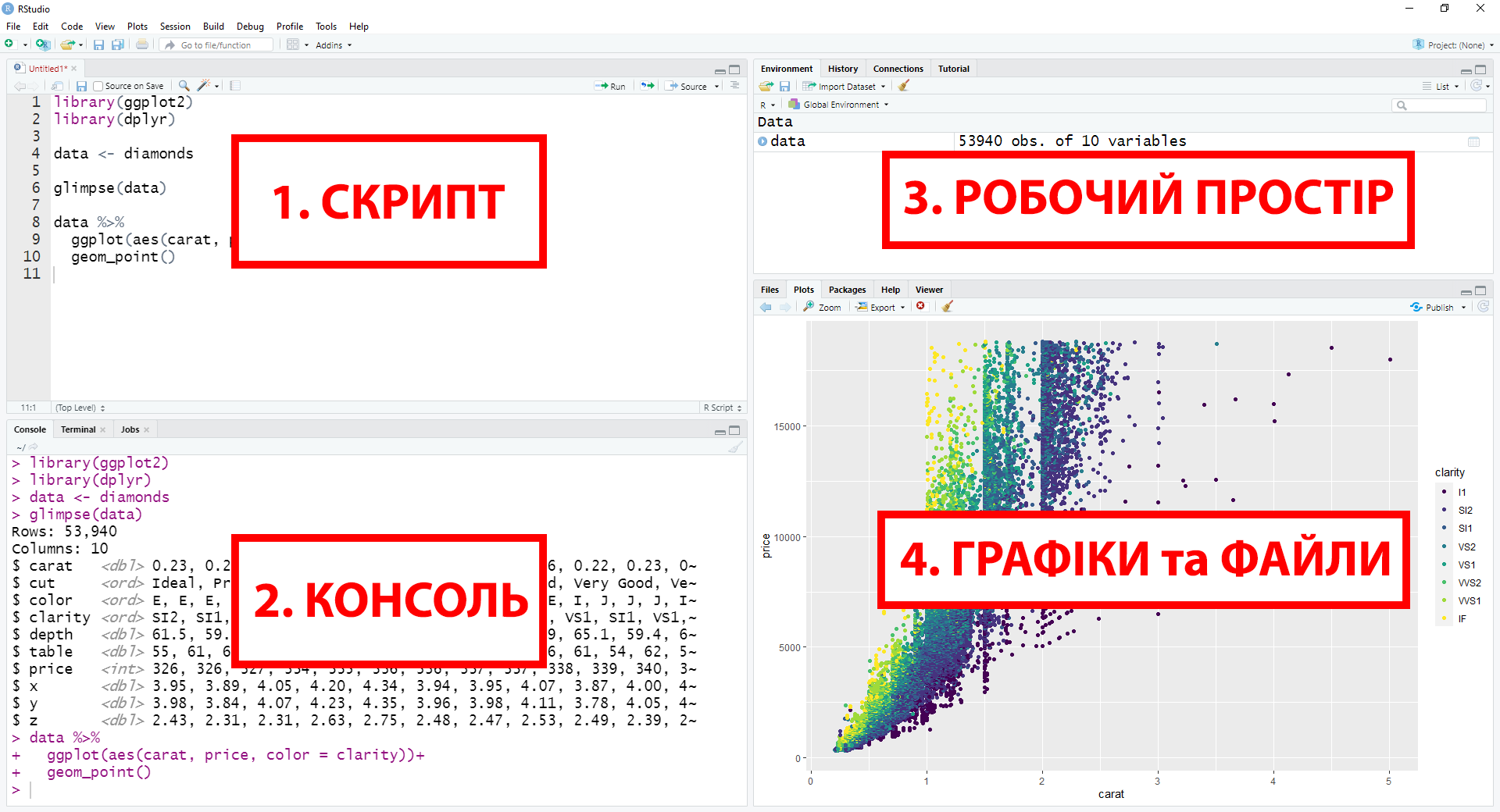


Рисунок 1.3: Інтерфейс RStudio

**1. СКРИПТ** — відбувається основна робота з кодом.

**2. КОНСОЛЬ** — повертаються результати виконання скриптів. Хоча тут так само писати код, проте він не буде збережений.

**3. РОБОЧИЙ ПРОСТІР** — тут зберігаються змінні, завантажені дата сети та побудовані моделі. Крім того, є окрема вкладинка історії останніх команд.

**4. ГРАФІКИ та ФАЙЛИ** — тут в окремих закладинках демонструються графіки, є невеличкий файловий менеджер, менеджер пакетів (про них трошки пізніше) та довідка по функціях (сюди будемо звертатися досить часто).

## 1.3 Базові операції

R - мова програмування з широкими можливостями. З її допомогою можна будувати математичні моделі, проводити статистичні тести, візуалізувати дані тощо. Але почнемо з чогось максимально простого, наприклад, арифметики.

### 1.3.1 R та арифметика

Вже з “коробки,” без додаткових пакетів, R надає можливість проводити арифметичні розрахунки. Всі ці операції виконуються за допомогою типових операторів, до яких ми звикли ще з часів навчання в школі або використання Excel.

Так, додавання двох чисел використовується оператор +:

2 + 2  
## [1] 4

Для віднімання — оператор -:

5 - 2  
## [1] 3

Множення через оператор \*:

3 \* 5  
## [1] 15

Ділення — /:

25 / 5  
## [1] 5

Піднесення до степеня через ^:

3 ^ 3   
## [1] 27

Як бачимо, нічого складного. Розгляньмо ще два оператори.

Залишок від ділення — %%:

5 %% 3  
## [1] 2

Цілочисельне ділення — %/%:

17 %/% 5  
## [1] 3

Узагальнений перелік арифметичних операторів я помістив у наступну таблицю.

|  |  |  |
| --- | --- | --- |
| Оператор | Опис | Приклад |
| + | Додавання | a + b |
| - | Віднімання | a - b |
| \* | Множення | a \* b |
| / | Ділення | a / b |
| ^ | Степінь | a ^ b |
| %% | Залишок від ділення | a %% b |
| %/% | Цілочисельне ділення | a %/% b |

### 1.3.2 Коментарі до коду

Важливою складовою написання коду, окрім його зрозумілості та лаконічності, є коментарі, які допомагають орієнтуватися, що відбувається в даному шматку коду.

Для додавання коментарів використовується знак решітки або, як його ще називають, дієз — #. Все, що написано у рядку після # ігнорується R:

# look at this awesome code  
2 + 2 - 3 + 1  
## [1] 2

Якщо необхідно “закоментувати” частину коду, тобто поставити # на початку кожного рядка, то для цього є комбінація клавіш **Ctrl + Shift + C**.

### 1.3.3 Пріоритети розрахунків

Після знайомства з арифметикою в R постає логічне питання з приводу пріоритетів операторів. На справді тут все просто, пріоритети працюють як в математиці. Тож не забувайте правильно розставляти дужки.

2 + 3 \* 4  
## [1] 14  
  
(2 + 3) \* 4  
## [1] 20

### 1.3.4 Готові функції

Крім звичайних арифметичних операторів в мові програмування R одразу вбудовано багато різноманітних функцій, в тому числі й математичних.

Для розрахунку кореня квадратного з числа використовується функція sqrt(), яка може бути вам знайома, якщо ви користувалися англомовною версією Excel.

9^0.5  
## [1] 3  
  
9^(1/2)  
## [1] 3  
  
sqrt(9)  
## [1] 3

***Мова програмування R чутлива до регістру***, тож варіанти Sqrt(9), SQRT(9) працювати не будуть.

Разом з тим, в середині функції можна проводити математичні розрахунки:

sqrt((2 + 3) \* 4)  
## [1] 4.472136

Функція abs() розраховує абсолютне значення:

abs(3 - 5)  
## [1] 2

Розрахунок натурального логарифма через функцію log():

log(15)  
## [1] 2.70805

Але це ще не все, оскільки для логарифма притаманно мати основу, то в таких випадках слід додати в середині дужок додатковий аргумент base =:

log(x = 10, base = 3)  
## [1] 2.095903

Аргументи функцій можна змінювати місцями:

log(base = 3, x = 10)  
## [1] 2.095903

До речі не обов’язково писати назви аргументів функцій. Якщо ви впевнені в їх порядку запис може виглядати наступним чином:

log(10, 3)  
## [1] 2.095903

Взагалі використання функцій — важлива особливість мови програмування R. Ми можемо створювати власні функції, використовувати функції як аргументи інших функцій тощо.

### 1.3.5 Вбудована документація

Вже на цьому етапі може виникнути ряд питань:

* де знайти опис функції?
* які існують аргументи функцій?
* чи є якісь приклади використання функцій?

Відповідь досить проста — всі функції супроводжуються детальною документацією. Для її виклику можна скористуватися функцією help(), де в середині дужок вказати назву функції:

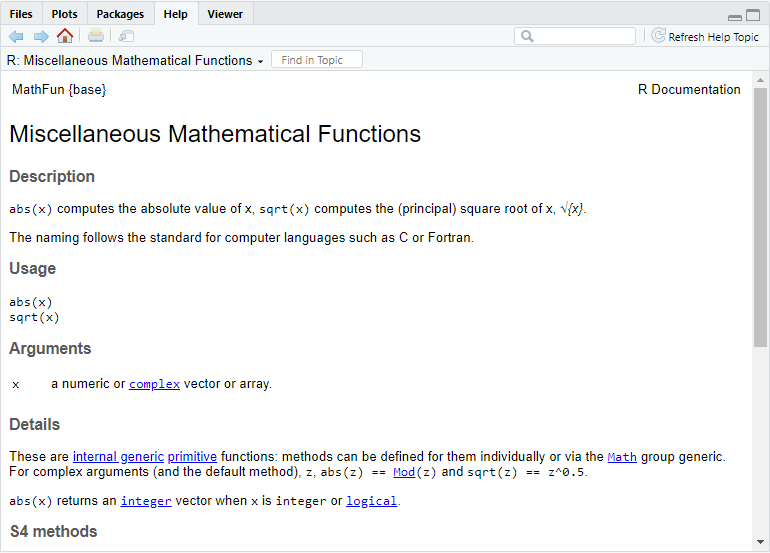
help(abs)

Інший варіант — це написати в консолі знак питання і після цього назву функції:

?abs()

Але на мою думку, **найшвидший варіант** — це написати функцію і **натиснути клавішу F1** на клавіатурі.

Всі зазначені варіанти приведуть вас до вікна документації у правому нижньому куті RStudio:



### 1.3.6 Створення змінних

Ще один важливий пункт у багатьох мовах програмування — можливість зберігати значення у змінних.

В R це робиться за допомогою символів **<-**. Для швидкого написання існує зручна комбінація клавіш Alt + -.

Ліворуч від символу **<-** записується назва майбутньої змінної. Праворуч — значення або вираз, яке необхідно зберегти в цю змінну.

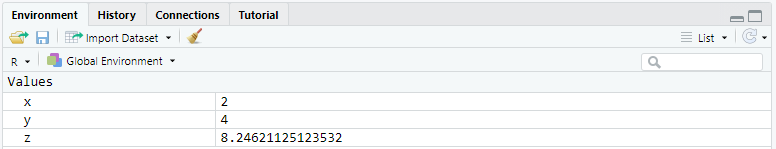
x <- 2  
y <- 4  
z <- sqrt(x^2 + y^3)

Для присоювання можна використовувати і знак **=** але серед спільноти користувачів R це не прийнято.

Слід звернути увагу, що при присвоюванні результат розрахунку не виводиться в консоль. Якщо ж хочеться переглянути результат слід використати функцію print() або просто звернутися до створеної змінної:

print(z)  
## [1] 8.246211  
  
z  
## [1] 8.246211

Після створення змінна з’являється у вкладинці **Environment** робочого простору RStudio:



### 1.3.7 Оператори порівняння

В процесі роботи з даними, досить часто працюємо з задачами порівняння. Для розв’язання таких питань в мові програмування R є зручні та зрозумілі оператори:

|  |  |  |
| --- | --- | --- |
| Оператор порівняння | Опис | Приклад |
| == | Дорівнює | a == b |
| != | Не дорівнює | a != b |
| > | Більше | a > b |
| < | Менше | a < b |
| >= | Більше або дорівнює | a >= b |
| <= | Менше або дорівнює | a <= b |

Окремо зауважу, що для порівняння двох змінних використовується оператор з подвійним знаком дорівнює ==, а не з одним =. Це досить популярна помилка.

## 1.4 Типи даних

Інформація зберігається в різних типах даних. Це може бути число, текст, булева змінна тощо.

1. Один з найпопулярніших варіантів збереження даних — це **числовий формат**. В мові програмування R для нього є окремий клас — **numeric**. При цьому існує три типи numeric:

* Цілі - **integer**.
* Дробові - **double**.
* Комплексні - **complex**.

В більшості випадків R буде сам конвертувати числа в потрібний формат. Але якщо необхідно задати конкретний тип числа, то можна використати функції as.integer(), as.double() та as.complex().

В мові програмування R досить багато функцій, які починаються на as., які переводять об’єкт до конкретного класу. Іншими словами просить читати дані в середині дужок у відповідному форматі.

Для створення цілочислового значення можна в кінці поставити символ L, щоб примусово оголосити число як integer:

is.integer(10)  
## [1] FALSE  
  
is.integer(10L)  
## [1] TRUE

Функції, що починаються на is. перевіряють, чи належить об’єкт до обраного класу.

1. Для роботи з **текстовими даними** є клас **character**. Вони записуються в лапках, при цьому можна використовувати як подвійні ", так і одинарні '.

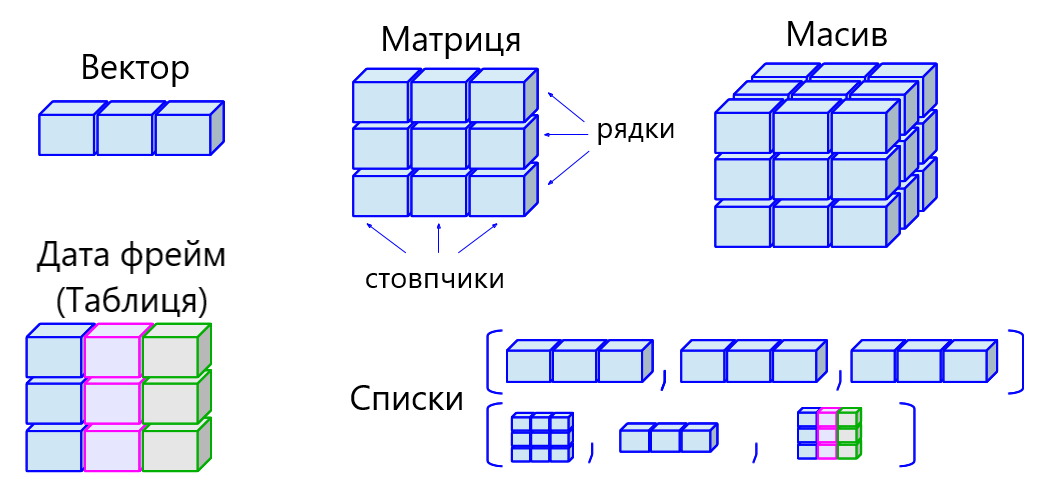
is.character("Ash nazg durbatulûk, ash nazg gimbatul,  
 ash nazg thrakatulûk, agh burzum-ishi krimpatul.")  
## [1] TRUE

1. **Логічні (logical) данні** - це тип даних які приймають лише значення TRUE або FALSE. Ми з ними вже зустрічалися коли використовували оператори порівняння.

Для перевірки типу даних використовується функція class():

class(5)  
## [1] "numeric"

## 1.5 Структури даних

R працює з багатьма структурами даних: вектори, матриці, масиви, дата фрейми та списки. Вони відрізняються за способом створення, структурою, складністю будови та способом звернення до їх елементів. Схематично ці структури даних зображено на рисунку 1.4 

### 1.5.1 Вектор

Почнемо з найпростішого. **Вектор** - це одновимірна послідовність елементів **одного типу**. Для створення вектору використовується функція c().

c(2, 4, 8, -2, -6, 0)  
## [1] 2 4 8 -2 -6 0  
  
c("два", "чотири", "шість")  
## [1] "два" "чотири" "шість"  
  
c(TRUE, TRUE, FALSE)  
## [1] TRUE TRUE FALSE

Для створення послідовностей з кроком 1 зручно використовувати оператор :

-5:5  
## [1] -5 -4 -3 -2 -1 0 1 2 3 4 5  
  
3:-2  
## [1] 3 2 1 0 -1 -2

Для складніших послідовностей є вбудована функція seq() (не забувайте дивитися довідку по функціях)

seq(1, 10, by = 2)  
## [1] 1 3 5 7 9

Крім того, можна задати не лише крок послідовності (аргумент by =), а й довжину (аргумент length.out =)

seq(1, 10, length.out = 3)  
## [1] 1.0 5.5 10.0

Інша корисна функція rep() дозволяє створити вектор з повторюваними значеннями. Перший аргумент - значення яке слід повторювати, другий аргумент - скільки разів повторювати.

rep(3, 5)  
## [1] 3 3 3 3 3

При цьому і перший і другий аргумент може бути вектором:

rep(1:3, 5)  
## [1] 1 2 3 1 2 3 1 2 3 1 2 3 1 2 3  
  
rep(1:3, 1:3)  
## [1] 1 2 2 3 3 3

Крім того, можна об’єднувати вектори:

v1 <- c("Say", "friend")  
  
v2 <- c("and", "enter")  
  
c(v1, v2)  
## [1] "Say" "friend" "and" "enter"

В означенні вектора в мові програмування R сказано, що **всі елементи вектори мають бути одного типу**. Одразу хочеться перевірити, що буде, якщо ця умова не буде виконуватись. В більшості мов програмування ми б отримали помилку. Мова програмування R, при розбіжності типів, буде зводити все до “спільного знаменника,” тобто конвертувати дані за певними правилами - приведення типів (coercion).

Виділяють два типи:

* **неявне приведення типів (implicit coercion)**, коли все відбувається автоматично за вбудованими правилами.
* **явне приведення типів (explicit coercion)**, коли ми самі вирішуємо до якого типу перевести дані, якщо це можливо.

Приклад неявного приведення типів:

c(TRUE, 2, FALSE)  
## [1] 1 2 0  
  
3 - TRUE  
## [1] 2

TRUE перетворився в 1, а FALSE в 0. В цьому випадку всі дані приведено до текстового типу:

c(TRUE, 2, "Hello")  
## [1] "TRUE" "2" "Hello"

В R є своя ієрархія типів: NULL < raw < logical < integer < double < complex < character < list < expression

Для явного приведення типів даних використовується сімейство функцій, що починається на as.:

as.numeric(c(TRUE, 2, FALSE, FALSE))  
## [1] 1 2 0 0  
  
as.character(c(TRUE, 2, FALSE, FALSE))  
## [1] "1" "2" "0" "0"

#### 1.5.1.1 Операції з векторами

Всі арифметичні операції, що ми розглядали раніше, можна використовувати й **до векторів однакової довжини**:

q <- 1:5  
  
w <- 2:6  
  
q + w  
## [1] 3 5 7 9 11  
  
q - w  
## [1] -1 -1 -1 -1 -1  
  
q \* w  
## [1] 2 6 12 20 30  
  
q / w  
## [1] 0.5000000 0.6666667 0.7500000 0.8000000 0.8333333  
  
w ^ q + q - w \* q  
## [1] 1 5 55 609 7751  
  
sqrt(q)  
## [1] 1.000000 1.414214 1.732051 2.000000 2.236068

#### 1.5.1.2 Правило переписування

Якщо **вектори не однакової довжини** й ми хочемо провести з ними певні операції, то в такому випадку спрацює **правило переписування (recycling rule)**: якщо коротший вектор кратний довжині довшого, короткий буде повторюватися необхідну кількість разів.

q <- 1:2  
w <- 1:4  
q \* w  
## [1] 1 4 3 8

Операції з вектором та окремим значенням можна вважати окремим випадком рестайлінгу: окреме значення буде повторюватися необхідну кількість разів:

w \* 2  
## [1] 2 4 6 8

Якщо коротший вектор не кратний довшому (наприклад, перший довжиною 2, а інший - 3), то R все одно порахує результат, але видасть попередження:

q + c(1, 5, 7)  
## Warning in q + c(1, 5, 7): longer object length is not a multiple of shorter object length  
## [1] 2 7 8

#### 1.5.1.3 Індексація векторів

Індексація - задача вибору окремого елемента структури даних. Для цього використовуються квадратні дужки [].

Найпростіший варіант індексація по номеру, тобто порядкове значення елемента:

e <- c(-5:-3, 2, 7, -6, 4:2)  
e[1]  
## [1] -5  
  
e[5]  
## [1] 7

**Важливий факт - індексація в мові програмування R починається з 1.**

За допомогою індексації можна не лише звертатися до окремого елементу, а й заміняти його значення:

e[5] <- 15

Можна використовувати цілі вектори для індексації:

e[2:5]  
## [1] -4 -3 2 15  
  
e[6:1]  
## [1] -6 15 2 -3 -4 -5

Індексація зі знаком мінус видасть всі значення вектора крім обраних:

e[-1]  
## [1] -4 -3 2 15 -6 4 3 2  
  
e[c(-1, -5)]  
## [1] -4 -3 2 -6 4 3 2

### 1.5.2 Матриці

**Матриця (matrix)** — це двовимірний масив даних, в якому кожен елемент має однаковий тип (числовий, текстовий, логічний). Іншими словами, матриця - це двовимірний вектор, у якого є довжина та ширина.

Створення матриці відбувається за допомогою функції matrix(), в якій слід вказати кількість рядків та стовпчиків:

matrix(1:16, nrow = 4, ncol = 4)  
## [,1] [,2] [,3] [,4]  
## [1,] 1 5 9 13  
## [2,] 2 6 10 14  
## [3,] 3 7 11 15  
## [4,] 4 8 12 16

За замовчуванням значення матриці заповнюються по стовпчиках. Але це можна змінити за допомогою аргументу byrow = TRUE

matrix(1:16, nrow = 4, ncol = 4, byrow = TRUE)  
## [,1] [,2] [,3] [,4]  
## [1,] 1 2 3 4  
## [2,] 5 6 7 8  
## [3,] 9 10 11 12  
## [4,] 13 14 15 16

Оскільки матриця — це двовимірний масив, то для індексації використовуються два індекси, що розділені комою: перший відповідає за рядок, другий за стовпчик.

my\_matrix <- matrix(1:16, nrow = 4, ncol = 4)  
my\_matrix  
## [,1] [,2] [,3] [,4]  
## [1,] 1 5 9 13  
## [2,] 2 6 10 14  
## [3,] 3 7 11 15  
## [4,] 4 8 12 16  
  
my\_matrix[2, 3]  
## [1] 10  
  
my\_matrix[1:2, 3:4]  
## [,1] [,2]  
## [1,] 9 13  
## [2,] 10 14

Якщо один з індексів залишити пустим — отримаємо всі значення рядка/стовпчика в залежності від того, який індекс ми не вказуємо.

my\_matrix[, 1:2]  
## [,1] [,2]  
## [1,] 1 5  
## [2,] 2 6  
## [3,] 3 7  
## [4,] 4 8  
  
my\_matrix[1:2, ]  
## [,1] [,2] [,3] [,4]  
## [1,] 1 5 9 13  
## [2,] 2 6 10 14

Аналогічно до вектора, за допомогою індексації можна переписувати частину матриці:

my\_matrix[1:2, 3:4] <- 0  
my\_matrix  
## [,1] [,2] [,3] [,4]  
## [1,] 1 5 0 0  
## [2,] 2 6 0 0  
## [3,] 3 7 11 15  
## [4,] 4 8 12 16

### 1.5.3 Масиви

**Масиви даних (array)** — схожі на матриці, але мають понад два виміри. Створюються вони за допомогою функції array(), де слід вказати вектор, з якого буде створено масив, і його розмірність:

my\_array <- array(1:16, c(4, 2, 2))  
my\_array  
## , , 1  
##   
## [,1] [,2]  
## [1,] 1 5  
## [2,] 2 6  
## [3,] 3 7  
## [4,] 4 8  
##   
## , , 2  
##   
## [,1] [,2]  
## [1,] 9 13  
## [2,] 10 14  
## [3,] 11 15  
## [4,] 12 16

### 1.5.4 Списки

**Списки** — це впорядкований набір об’єктів. Я представляю собі списки, як блокнот, в якому на кожній сторінці зберігається інформація у певному форматі. Вони можуть зберігати різні дані, в тому числі вектори, матриці, дата фрейми й інші списки.

Списки створюються за допомогою функції list():

my\_list <- list(vec = c(1:5),  
 gendalf = "You shall not pass",  
 my\_matrix = matrix(1:4, ncol = 2))  
my\_list  
## $vec  
## [1] 1 2 3 4 5  
##   
## $gendalf  
## [1] "You shall not pass"  
##   
## $my\_matrix  
## [,1] [,2]  
## [1,] 1 3  
## [2,] 2 4

В цьому випадку vec, gendalf та my\_matrix — назви елементів списку.

Для звернення до елементів списку можна використовувати індекси або імена (через символ $). При зверненні через індекс необхідно використовувати подвійні квадратні дужки, наприклад:

my\_list[[2]]  
## [1] "You shall not pass"  
  
my\_list$gendalf  
## [1] "You shall not pass"  
  
my\_list[['gendalf']]  
## [1] "You shall not pass"

Списки досить часто використовуються в R, наприклад результати побудови математичних моделей, статистичних тестів зберігаються у вигляді списків. тож важливо вміти з ними працювати.

### 1.5.5 Дата фрейми

Нарешті ми перейшли до найголовнішого - **дата фрейми (data frames)**. Саме з такою структурою даних працюють найчастіше. Головною особливістю їх є те, що різні стовпчики можуть містити різний тип даних (але їх довжина має бути однаковою). Для створення дата фрейму використовується функція data.frame().

my\_df <- data.frame(name = c("Frodo", "Eowyn", "Legolas", "Arwen"),  
 sex = c("male", "female", "male", "female"),  
 age = c(51, 24, 2931, 2700),  
 one\_ring = c(TRUE, FALSE, FALSE, FALSE))  
my\_df

|  |  |  |  |
| --- | --- | --- | --- |
| name | sex | age | one\_ring |
| Frodo | male | 51 | TRUE |
| Eowyn | female | 24 | FALSE |
| Legolas | male | 2931 | FALSE |
| Arwen | female | 2700 | FALSE |

Переглянути структуру дата фрейму можна за допомогою функції str():

str(my\_df)  
## 'data.frame': 4 obs. of 4 variables:  
## $ name : chr "Frodo" "Eowyn" "Legolas" "Arwen"  
## $ sex : chr "male" "female" "male" "female"  
## $ age : num 51 24 2931 2700  
## $ one\_ring: logi TRUE FALSE FALSE FALSE

В нашому випадку перший та другий стовпчик - *текстові*, третя - *числова*, четверта - *логічна*.

Переглянути назви стовпців або надати їм нову назву можна за допомогою функції names().

names(my\_df)  
## [1] "name" "sex" "age" "one\_ring"

Індексація аналогічно до матриці та списків можлива через [] та знак $.

my\_df$name  
## [1] "Frodo" "Eowyn" "Legolas" "Arwen"  
  
my\_df$name[2:3]  
## [1] "Eowyn" "Legolas"  
  
my\_df[2,3]  
## [1] 24  
  
my\_df[2:3, "name"]  
## [1] "Eowyn" "Legolas"

Для перегляду дата фрейму в RStudio використовується функція View() або можна просто натиснути на назву змінної у розділі Environment. Ви побачите таблицю, дещо схожу на Excel або Google Spreadsheets.

## 1.6 Пакети в R

R - мова програмування з дуже широкими можливостями. Однак рано чи пізно ми почнемо стикатися з задачами, які потребуватимуть додаткових можливостей. Ці можливості можна розширити за допомогою додаткових **пакетів (packages)**.

В більшості випадків основним змістом пакетів є набір додаткових функцій, даних або нових структур даних.

До найпопулярніших пакетів у сфері науки про дані (data science) можна віднести пакети екосистему пакетів **tidyverse**, пакети **data.table**, **mlr3** та ще багато-багато інших.

### 1.6.1 Вбудовані пакети

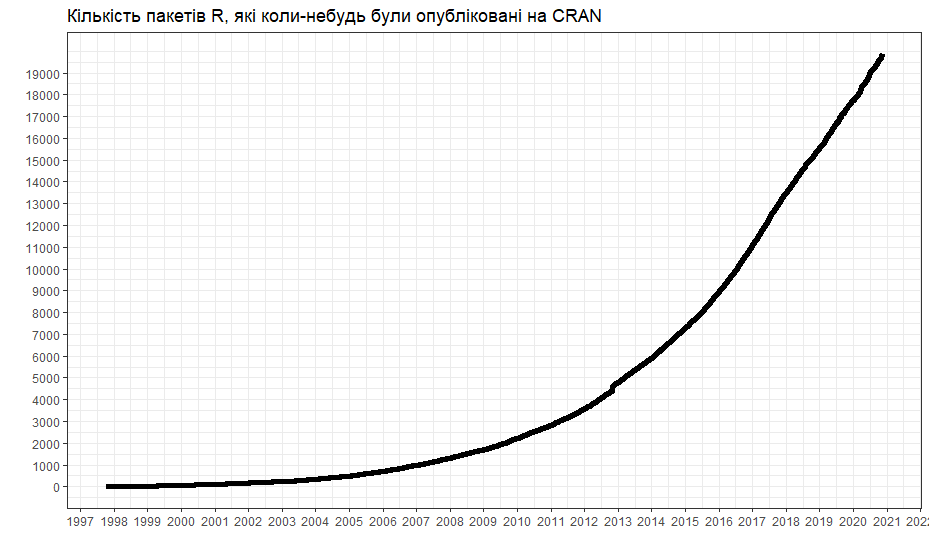
Разом з самою мовою програмування R поставляються пакети, які не потрібно встановлювати: основний base та декілька інших, такі як stats, utils, graphics та інші. Повний перелік можна переглянути за наступним кодом:

rownames(installed.packages(priority = "base"))  
## [1] "base" "compiler" "datasets" "graphics" "grDevices" "grid" "methods"   
## [8] "parallel" "splines" "stats" "stats4" "tcltk" "tools" "utils"

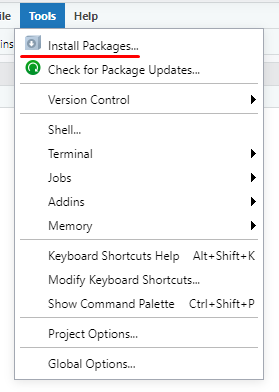
### 1.6.2 Встановлення пакетів з CRAN

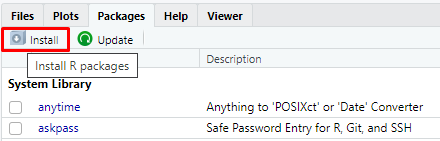
Пакети які пройшли перевірку зберігаються в **Comprehensive R Archive Network (CRAN)**. Для перегляду кількості опублікованих пакетів на CRAN можна використати наступний код:

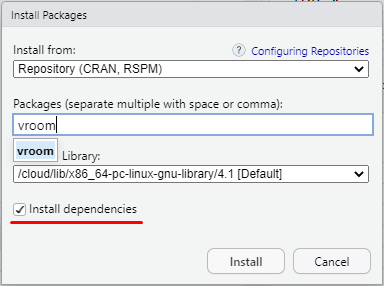
nrow(available.packages())  
## [1] 18180

І з кожним роком їх стає все більше:  Функція install.packages() дозволяє звантажувати та встановлювати пакети з CRAN. Для прикладу встановимо пакет vroom, для зчитування файлів

install.packages("vroom")

Інший спосіб встановлення пакетів - через окреме меню **Tools -> Install Packages** (рис.1.5)  Або через кнопку в розділі **Packages** (рис. 1.6)

 Після чого з’явиться меню (рис. 1.7) в якому можна вказати назву потрібного пакету. Бажано залишити галочку **“install dependencies”**, оскільки деякі пакети мають залежність від інших пакетів і їх також слід встановити. Може статися ситуація, коли завантажуючи один пакет разом з ним буде додатково встановлено ще десятки інших.

 ### Встановлення пакетів з GitHub {#r\_git} Бувають випадки, коли автори пакетів не пройшли або не захотіли проходити перевірку через CRAN (насправді причин може бути безліч). В таких випадках пакет (який ви напевно знайшли через пошук в Google) скоріш за все буде зберігатися на репозиторію GitHub. Для таких випадків нам необхідно встановити пакет devtools та використати з нього функцію install\_github(). Покажу на прикладі пакету xaringan, який я використовую для створення інтерактивних презентацій.

devtools::install\_github('yihui/xaringan')

Посилання yihui/xaringan — це адреса пакету в [GitHub](https://github.com/yihui/xaringan)

## 1.7 Завантаження даних в R

R може імпортувати цілий спектр різноманітних файлів

## 1.8 Умовні конструкції

## 1.9 Створення функцій в R

## 1.10 Вступ у Tidyverse

## 1.11 Створення звітів: R Markdown

# 2 Візуалізація часових рядів

text

# 3 Декомпозиція часових рядів

text

# 4 Аналіз часового ряду

text

# 5 Інструментарій прогнозування

text

# 6 Регресійні моделі

text

# 7 Експоненціальне згладжування

text

# 8 ARIMA-моделі

text

# 9 Prophet Facebook

text

# 10 Кластеризація

text

text