

Компьютерный практикум по статистическому анализу данных Лаб-01

Гань Чжаолун
НФИбд-01-21
15/11/2024



Цели и задачи работы



Цель лабораторной работы



Подготовить рабочее пространство и инструментарии для работы с языком программирования Julia, на простейших примерах познакомиться с основами синтаксиса Julia.



Процесс выполнения лабораторной работы

Установить Julia, Jupyter

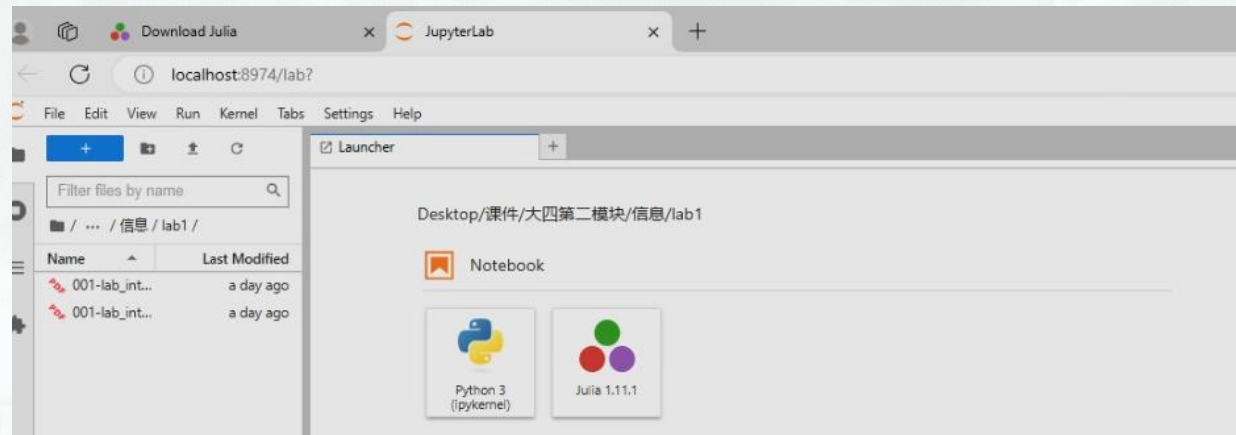


Рис. 1.1. интерфейс jupyter

Используя Jupiter Lab

повторите примеры

Функция `typeof` нужна для определения типа данных.

```
typeof(1), typeof(4.5), typeof(pi)
(Int64, Float64, Irrational{:π})

1.0/0.0, 1.0/(-0.0), 0.0/0.0
(Inf, -Inf, NaN)

typeof(1.0/0.0), typeof(1.0/(-0.0)), typeof(0.0/0.0)
(Float64, Float64, Float64)
```

Рис. 2.1. Пример функции `typeof`

Используя Jupiter Lab

повторите примеры

Для разных типов данных есть свои ограничения. Их можно узнать при помощи: Даны не все типы данных, в массив можно добавлять другие типы

```
for T in [Int8, Int16, Int32, Int64, Int128, UInt8, UInt16, UInt32, UInt64, UInt128]
    println("${lpad(T, 7)}: [$(typemin(T)), $(typemax(T))]" )
end
```

Int8: [-128, 127]
Int16: [-32768, 32767]
Int32: [-2147483648, 2147483647]
Int64: [-9223372036854775808, 9223372036854775807]
Int128: [-170141183460469231731687303715884105728, 170141183460469231731687303715884105727]
UInt8: [0, 255]
UInt16: [0, 65535]
UInt32: [0, 4294967295]
UInt64: [0, 18446744073709551615]
UInt128: [0, 340282366920938463463374607431768211455]

Рис. 2.2. Для разных типов данных есть свои ограничения

Используя Jupiter Lab

повторите примеры

Чтобы конвертировать из одного типа данных в другие можно использовать функцию `convert` или же имя типа данных, в которое надо конвертировать

```
Int64(2.0), Char(2)
(2, '\x02')

Int64(2.0), Char(2), typeof(Char(2))
(2, '\x02', Char)

convert(Int64, 2.0), convert(Char, 2)
(2, '\x02')

Bool(1), Bool(0)
(true, false)
```

Рис. 2.3. Пример функции convert



Используя Jupiter Lab

повторите примеры

Для приведения нескольких данных к единому типу можно использовать функцию `promote`

```
promote(Int8(1), Float16(4.5), Float32(4.1))  
(1.0f0, 4.5f0, 4.1f0)
```

Рис. 2.4. Пример функции `promote`

Используя Jupiter Lab

повторите примеры

Чтобы создать функцию нужно использовать ключевое слово **function** с указанием имени и параметров функции

```
function f(x)
x^2
end

f (generic function with 1 method)

f(4)

16
```

Рис. 2.5. Пример функции function

Используя Jupiter Lab повторите примеры

Также существует однострочный вариант записи функции

```
g(x)=x^2  
g (generic function with 1 method)  
g(8)  
64
```

Рис. 2.6. Однострочный вариант записи функции

Используя Jupiter Lab

повторите примеры

В julia встроены такие типы данных, как векторы, матрицы и т.д. Для транспонирования матрицы используется символ ' после названия переменной матрицы. Матрицы создаются и используется следующим образом

```
a = [4 7 6] # вектор-строка
b = [1, 2, 3] # вектор-столбец
a[2], b[2] # вторые элементы векторов a и b

(7, 2)

a = 1; b = 2; c = 3; d = 4 # присвоение значений
Am = [a b; c d] # матрица 2 x 2
Am[1,1], Am[1,2], Am[2,1], Am[2,2] # элементы матрицы

(1, 2, 3, 4)

aa = [1 2]
AA = [1 2; 3 4]
aa*AA*aa'
```

```
1×1 Matrix{Int64}:
 27
```

Рис. 2.7. Пример работы с массивами

Выполните задания для самостоятельной работы

```
?read()
read(io::IO, T)
Read a single value of type T from io, in canonical binary representation.

Note that Julia does not convert the endianness for you. Use ntoh or ltoh for this purpose.

read(io::IO, String)
Read the entirety of io, as a String (see also readchomp).
```

Для каждой функции я могу ввести символ ‘?’ слева, чтобы просмотреть соответствующую информацию.

Рис. 3.0. Пример справочная информация

Выполните задания для самостоятельной работы

`read()` – считывает один символ или байт из потока ввода.

```
io = IOBuffer("Hi,bro!")  
  
IOBuffer(data=UInt8[...], readable=true, writable=false, seekable=true, append=false, size=7, maxsize=Inf, ptr=1, mark=-1)  
  
io = IOBuffer("Hi,bro!");  
read(io, String)  
  
"Hi,bro!"
```

Функция `read()` используется для чтения данных из файла или потока. Её можно использовать для чтения данных в бинарном или текстовом формате. Эта функция возвращает все содержимое файла или потока.

Рис. 3.1. Пример функция `read()`

Выполните задания для самостоятельной работы

`readline()` – считывает одну строку из потока ввода.

```
write("GZL_lab1.txt", "你好.\n我是甘兆隆.\n");  
readline("GZL_lab1.txt")
```

"你好."

```
write("GZL_lab1.txt", "你好.\n我是甘兆隆.\n");  
readline("GZL_lab1.txt", keep=true)
```

"你好.\n"

Функция `readline()` используется для чтения одной строки из файла или потока.

Если `keep` имеет значение `false` (по умолчанию), эти конечные символы новой строки удаляются из строки до ее возврата. Когда `keep` имеет значение `true`, они возвращаются как часть строки.

Рис. 3.2. Пример функция `readline()`

Выполните задания для самостоятельной работы

`readlines()` – считывает все строки из потока ввода и возвращает массив строк.

```
readlines("GZL_lab1.txt")
```

```
2-element Vector{String}:  
"你好."  
"我是甘兆隆."
```

Функция `readline()` читает все строки из файла или потока и возвращает их в виде массива строк.

Каждая строка файла становится элементом массива.

Рис. 3.3. Пример функция `readlines()`

Выполните задания для самостоятельной работы

readdlm() – считывает данные из файла с разделителями и возвращает их в виде массива или матрицы.

```
using DelimitedFiles
x = [1; 2; 3; 4; 5];
y = [6; 7; 8; 9; 10];
open("delim_file.txt", "w") do io
    writedlm(io, [x y])
end;
readdlm("delim_file.txt", Int32)
```

```
5×2 Matrix{Int32}:
 1  6
 2  7
 3  8
 4  9
 5 10
```

```
readdlm("delim_file.txt", Float32)
```

```
5×2 Matrix{Float32}:
 1.0  6.0
 2.0  7.0
 3.0  8.0
 4.0  9.0
 5.0 10.0
```

Функция `readln()` используется для чтения данных, разделённых определёнными разделителями (например, запятыми, табуляцией и т.д.).

я воспользовалась утилитой `Delimited Files` для чтения и записи файлов с разделителями.

Рис. 3.4. Пример функция `readdlm()`

Выполните задания для самостоятельной работы

`print()` – выводит значения, разделяя их пробелами.

```
io = IOBuffer();  
print(io, "Hi", ' ', :bro!)  
String(take!(io))
```

"Hi bro!"

```
print("Hi, bro!")
```

Hi, bro!

Функция `print()` выводит переданное значение в стандартный вывод. Она не добавляет символ новой строки после вывода.

Рис. 3.5. Пример функция `print()`

Выполните задания для самостоятельной работы

`println()` - выводит значения, добавляя перевод строки в конце.

```
println("Hi, bro")  
Hi, bro  
  
io = IOBuffer();  
println(io, "Hi", ',', " bro.")  
String(take!(io))  
"Hi, bro.\n"
```

Функция `println()` аналогична `print()`, но после вывода добавляет символ новой строки.

Рис. 3.6. Пример функция `println()`

Выполните задания для самостоятельной работы

`show()` - отображает структуру данных с полной информацией о их содержимом.

```
show("Hi, bro!")  
"Hi, bro!"  
  
struct 月  
    n::Int  
end  
Base.show(io::IO, ::MIME"text/plain", d::月) = print(io, d.n, "月")  
月(4)  
  
4月
```

Функция `show()` используется для вывода представления значения, которое может быть полезным для разработчиков. Она может выводить данные в формате, более подходящем для отладки.

Рис. 3.7. Пример функция `show()`

Выполните задания для самостоятельной работы

`write()` – записывает данные в поток вывода.

```
io = IOBuffer();  
write(io, "Hi.", " bor.")  
String(take!(io))  
  
"Hi. bor."  
  
write(io, "Hi.") + write(io, " bor.")  
String(take!(io))  
  
"Hi. bor."
```

Функция `write()` используется для записи данных в файл или поток. Она записывает данные в бинарной форме, но также может записывать строки.

Рис. 3.8. Пример функция `write()`

Выполните задания для самостоятельной работы

Отличия между функциями:

Чтение данных:

`read()` — читает всё содержимое целиком.

`readline()` — читает одну строку.

`readlines()` — читает все строки и возвращает массив строк.

`readdelim()` — читает данные, разделённые определёнными символами.

Запись и вывод данных:

`print()` — выводит данные без перехода на новую строку.

`println()` — выводит данные с переходом на новую строку.

`show()` — выводит данные в формате, удобном для отладки.


`write()` — записывает данные в файл или поток (бинарный или текстовый формат).

Выполните задания для самостоятельной работы


Функция `parse()` в языке программирования Julia используется для преобразования строки в соответствующий ей тип данных. Например, она может преобразовать строку в целое число, число с плавающей запятой или другой тип данных.

```
parse{Int, "4568"}  
4568  
  
parse{Int, "4568", base=16}  
17768  
  
parse{Int, "1011111101000010111001", base=2}  
3133625
```

Рис. 3.9. Пример функция `parse()`



Выполните задания для самостоятельной работы



В Julia арифметические операции могут быть применены как к целым числам, так и к числам с плавающей запятой, с результатом соответствующего типа. Оператор деления `/` всегда возвращает число с плавающей точкой. Для возведения в степень используется символ `^`, а для извлечения квадратного корня - функция `sqrt()`. Логические операции выполняются с помощью операторов `&&` (логическое "и") и `||` (логическое "или"). также можно добавить к операции сравнения значение, большее или равное (`>=`), и значение, меньшее или равное (`<=`).

Выполните задания для самостоятельной работы

```
a=5
b=3

# сложение
println(a+b)

# вычитание
println(a-b)

# умножение
println(a*b)

# деление
println(a/b)

# деление с округлением до целого числа
println(div(a,b))
```

8
2
15
1.6666666666666667
1

```
# возведение в степень
println(a^b)

# извлечение корня
println(sqrt(25))
```

125
5.0

```
# сравнение

x=10
y=20

# равенство
println(x==y)

# неравенство
println(x!=y)

# больше или меньше
println(x<y)
println(x>y)

# больше или равно, меньше или равно
println(x<=y)
println(x>=y)
```

false
true
true
false
true
false

```
# логические операции


# and
println((x<y)&&(y>15))

# or
println((x>y) || (y==20))


# no
println(!(x==y))
```

true
true
true

Рис. 3.10. Примеры арифметических операций с данными



Выполните задания для самостоятельной работы



Действия с матрицами, векторами и т.д.

- Сложение и вычитание матриц выполняются покомпонентно.
- Скалярное произведение векторов вычисляется как сумма произведений соответствующих элементов векторов.
- Транспонирование матрицы приводит к замене строк на столбцы (и наоборот).
- Умножение матрицы на скаляр заключается в умножении каждого элемента матрицы на данный скаляр.

Выполните задания для самостоятельной работы

```
v1=[1, 4, 8]
v2=[6, 7, 9]

# сложение векторов
v_sum=v1+v2
println(v_sum)

# вычитание векторов
v_diff=v1-v2
println(v_diff)

# умножение на скаляр вектора
scalar=2
v_sca=scalar*v1
println(v_sca)

# скалярное произведение вектора
using LinearAlgebra

dot_product=dot(v1, v2)
println(dot_product)
```

[7, 11, 17]
[-5, -3, -1]
[2, 8, 16]
106

```
A=[1 4; 5 8]
B=[3 6; 7 9]

# сложение матрица
A_plus_B=A+B
println(A_plus_B)

# вычитание матрица
A_minus_B=A-B
println(A_minus_B)

# умножение на матрицы скаляр
scalar=3
A_sca=scalar*A
println(A_sca)
```

[4 10; 12 17]
[-2 -2; -2 -1]
[3 12; 15 24]

```
v=[3, 9]
A=[1 4; 5 8]
B=[2 0; 1 3]

# умножение матрицы на вектор
A_v_product=A*v
println(A_v_product)

# умножение 2 матриц
A_B_product=A*B
println(A_B_product)
```

[39, 87]
[6 12; 18 24]

```
# транспонирование
v=[4, 5, 8]
A=[1 4; 5 8]

# транспонирование вектора
v_transposed=v'
println(v_transposed)

# транспонирование матрицы
A_transposed=A'
println(A_transposed)
```

[4 5 8]
[1 5; 4 8]

Рис. 3.11. Примеры основные операции с матрицами



Выводы по проделанной работе

Вывод

По итогам выполнения работы, я выполнил задание лабораторной работы. Я подготовил рабочую область и инструменты для использования языка программирования Julia и использовал простейшие примеры, чтобы ознакомиться с основами грамматики Julia.



С п а с и б о