R for Geoscience

Shuxin Ji

2023-08-13

Contents

About

shuxin R R Arcgis Qgis ENVI SNAP
R

githu

emo

-
-
- ArcGIS QGIS R Python Javascript
- Linux shell

practice

Part I

Chapter 1

R

1.1 What is R?

R Ross Ihaka Robert Gentleman 20 90 R GNU

R R

R (IDE) RStudio R SQL Python Ex-
cel

1.2 Why R?

R

- R
- R
- R
- R
- R SQL Python Excel

R

1.3

1.3.1 R

R R R

1.3.2 RStudio

RStudio R R RStudio Desktop Max OS- Linux

1.3.3 Rstudio

——Rstudio

1.3.4 R

R R RStudio R ——R -

```
install.packages("terra")
```

-

```
install.packages(c("terra", "pacman", "tidyverse", "leaflet"))
```

- pacman

```
ifelse(!"pacman" %in% installed.packages(), install.packages("pacman"),
       library(pacman))
p_load(terra, tidyverse, leaflet)
```

Attention, Please!
Please do not use any Chinese charactor( ) to set your path!

1.4 comments

R R # Ctrl+Shift+C

-
-
-
-

```
1 + 2     # this is use to sum 1 and 2
```

1.5

R R R R The R Base Package

Chapter 2

2.1 Vector( )

Vector R

1 5 5

```r
x <- c(1,2,3,4,5)
x
#> [1] 1 2 3 4 5
```

`c()` 1 2 3 4 5 `<-` `x` 1 2 3 4 5 `x` R R `c()` ??

R google :, R

```r
x <- c(1:5)
x
#> [1] 1 2 3 4 5
```

vector `typeof()`

```r
typeof(x)
#> [1] "integer"
```

length ,`length()`

```
length(x)
#> [1] 5
```

R seq()

```
seq(1, 9, 0.5)
#>  [1] 1.0 1.5 2.0 2.5 3.0 3.5 4.0 4.5 5.0 5.5 6.0 6.5 7.0 7.5
#> [15] 8.0 8.5 9.0
```

1 9 0.5 3 ?? Console
??seq
Help :: :: package::function cli ar
generation
Description Usage,seq(...) seq (), A

vector

```
# Vector of logical values
log_values <- c(TRUE, FALSE, TRUE, FALSE)

log_values
#> [1]  TRUE FALSE  TRUE FALSE
```

R

```
fruits <- c("beijing", "shanghai", "guangzhou", "shenzhen", "xianggang", "50")
fruits
#> [1] "beijing"   "shanghai"  "guangzhou" "shenzhen"
#> [5] "xianggang" "50"
```

[] brackets, fruits "beijing" "shenzhen"

```
fruits[c(1,4)]
#> [1] "beijing"  "shenzhen"
```

```
fruits[1:4]
#> [1] "beijing"   "shanghai"  "guangzhou" "shenzhen"
```

"beijing

```
fruits[-1]
#> [1] "shanghai"  "guangzhou" "shenzhen"  "xianggang"
#> [5] "50"
```

`sort`,

```
fruits <- c("beijing", "shanghai", "guangzhou", "shenzhen", "xianggang")
numbers <- c(13, 3, 5, 7, 20, 2)

sort(fruits)  # Sort a string
#> [1] "beijing"   "guangzhou" "shanghai"  "shenzhen"
#> [5] "xianggang"
sort(numbers) # Sort numbers
#> [1]  2  3  5  7 13 20
```

2.2 Lists( )

R `list()`

```
thislist <- list(
  a = c("shanghai", "beijing", "cherry"),
  b = c(1,2,5,6,7,9),
  c = c(TRUE, FALSE, TRUE)
)
# Print the list
thislist
#> $a
#> [1] "shanghai" "beijing"  "cherry"
#>
#> $b
#> [1] 1 2 5 6 7 9
#>
#> $c
#> [1]  TRUE FALSE  TRUE
```

```
typeof(thislist)
#> [1] "list"
```

```r
length(thislist)
#> [1] 3
```

2.3 Matrices( )

(column) (row) `matrix()`
,

```r
# Create a matrix
thismatrix <- matrix(c(1,2,3,4,5,6), nrow = 3, ncol = 2)

# Print the matrix
thismatrix
#>      [,1] [,2]
#> [1,]    1    4
#> [2,]    2    5
#> [3,]    3    6
```

NOTE: `c()`

```r
thismatrix <- matrix(c("shanghai", "beijing", "cherry", "guangzhou"), nrow = 2, ncol =

thismatrix
#>      [,1]       [,2]
#> [1,] "shanghai" "cherry"
#> [2,] "beijing"  "guangzhou"
```

Access Matrix Items You can access the items by using [ ] brackets. The first number “1” in the bracket specifies the row-position, while the second number “2” specifies the column-position:

```r
thismatrix <- matrix(c("shanghai", "beijing", "cherry", "guangzhou"), nrow = 2, ncol =

thismatrix[1, 2]
#> [1] "cherry"
```

The whole row can be accessed if you specify a comma after the number in the bracket:

```
thismatrix <- matrix(c("shanghai", "beijing", "cherry", "guangzhou"), nrow = 2, ncol = 2)

thismatrix[2,]
#> [1] "beijing"   "guangzhou"
```

The whole column can be accessed if you specify a comma before the number in the bracket:

```
thismatrix <- matrix(c("shanghai", "beijing", "cherry", "guangzhou"), nrow = 2, ncol = 2)

thismatrix[,2]
#> [1] "cherry"    "guangzhou"
```

Access More Than One Row More than one row can be accessed if you use the c() function:

```
thismatrix <- matrix(c("shanghai", "beijing", "cherry", "guangzhou","grape", "pineshanghai", "pea

thismatrix[c(1,2),]
#>      [,1]       [,2]        [,3]
#> [1,] "shanghai" "guangzhou" "pear"
#> [2,] "beijing"  "grape"     "melon"
```

Access More Than One Column More than one column can be accessed if you use the c() function:

```
thismatrix <- matrix(c("shanghai", "beijing", "cherry", "guangzhou","grape", "pineshanghai", "pea

thismatrix[, c(1,2)]
#>      [,1]       [,2]
#> [1,] "shanghai" "guangzhou"
#> [2,] "beijing"  "grape"
#> [3,] "cherry"   "pineshanghai"
```

Add Rows and Columns Use the cbind() function to add additional columns in a Matrix:

```
thismatrix <- matrix(c("shanghai", "beijing", "cherry", "guangzhou","grape", "pineshanghai", "pea

newmatrix <- cbind(thismatrix, c("strawberry", "blueberry", "raspberry"))

# Print the new matrix
newmatrix
```

```
#>      [,1]       [,2]           [,3]    [,4]
#> [1,] "shanghai" "guangzhou"    "pear"  "strawberry"
#> [2,] "beijing"  "grape"        "melon" "blueberry"
#> [3,] "cherry"   "pineshanghai" "fig"   "raspberry"
```

Use the rbind() function to add additional rows in a Matrix:

```
thismatrix <- matrix(c("shanghai", "beijing", "cherry", "guangzhou","grape", "pineshang

newmatrix <- rbind(thismatrix, c("strawberry", "blueberry", "raspberry"))

# Print the new matrix
newmatrix
#>      [,1]         [,2]           [,3]
#> [1,] "shanghai"   "guangzhou"    "pear"
#> [2,] "beijing"    "grape"        "melon"
#> [3,] "cherry"     "pineshanghai" "fig"
#> [4,] "strawberry" "blueberry"    "raspberry"
```

Remove Rows and Columns Use the c() function to remove rows and columns in a Matrix:

```
thismatrix <- matrix(c("shanghai", "beijing", "cherry", "guangzhou", "shenzhen", "pine

#Remove the first row and the first column
thismatrix <- thismatrix[-c(1), -c(1)]

thismatrix
#> [1] "shenzhen"     "pineshanghai"
```

Check if an Item Exists To find out if a specified item is present in a matrix, use the %in% operator:

```
thismatrix <- matrix(c("shanghai", "beijing", "cherry", "guangzhou"), nrow = 2, ncol =

"shanghai" %in% thismatrix
#> [1] TRUE
```

Number of Rows and Columns Use the dim() function to find the number of rows and columns in a Matrix:

```
thismatrix <- matrix(c("shanghai", "beijing", "cherry", "guangzhou"), nrow = 2, ncol =

dim(thismatrix)
#> [1] 2 2
```

Matrix Length Use the length() function to find the dimension of a Matrix:

```
thismatrix <- matrix(c("shanghai", "beijing", "cherry", "guangzhou"), nrow = 2, ncol = 2)

length(thismatrix)
#> [1] 4
```

Combine two Matrices Again, you can use the rbind() or cbind() function to combine two or more matrices together:

```
# Combine matrices
Matrix1 <- matrix(c("shanghai", "beijing", "cherry", "grape"), nrow = 2, ncol = 2)
Matrix2 <- matrix(c("guangzhou", "shenzhen", "pineshanghai", "watermelon"), nrow = 2, ncol = 2)

# Adding it as a rows
Matrix_Combined <- rbind(Matrix1, Matrix2)
Matrix_Combined
#>      [,1]        [,2]
#> [1,] "shanghai"  "cherry"
#> [2,] "beijing"   "grape"
#> [3,] "guangzhou" "pineshanghai"
#> [4,] "shenzhen"  "watermelon"

# Adding it as a columns
Matrix_Combined <- cbind(Matrix1, Matrix2)
Matrix_Combined
#>      [,1]       [,2]     [,3]        [,4]
#> [1,] "shanghai" "cherry" "guangzhou" "pineshanghai"
#> [2,] "beijing"  "grape"  "shenzhen"  "watermelon"
```

2.4 Data Frame( )

data.frame()

```
# Create a data frame
Data_Frame <- data.frame (
  Training = c("Strength", "Stamina", "Other"),
  Pulse = c(100, 150, 120),
  Duration = c(60, 30, 45)
)
```

```r
# Print the data frame
Data_Frame
#>   Training Pulse Duration
#> 1 Strength   100       60
#> 2  Stamina   150       30
#> 3    Other   120       45
```

Use the `summary()` function to summarize the data from a Data Frame:

```r
summary(Data_Frame)
#>    Training             Pulse          Duration   
#>  Length:3           Min.   :100.0   Min.   :30.0  
#>  Class :character   1st Qu.:110.0   1st Qu.:37.5  
#>  Mode  :character   Median :120.0   Median :45.0  
#>                     Mean   :123.3   Mean   :45.0  
#>                     3rd Qu.:135.0   3rd Qu.:52.5  
#>                     Max.   :150.0   Max.   :60.0
```

`[]` `[[ ]]` `$`

```r
Data_Frame[1]
#>   Training
#> 1 Strength
#> 2  Stamina
#> 3    Other

Data_Frame[["Training"]]
#> [1] "Strength" "Stamina"  "Other"

Data_Frame$Training
#> [1] "Strength" "Stamina"  "Other"
```

`rbind()`

```r
# Add a new row
New_row_DF <- rbind(Data_Frame, c("Strength", 110, 110))

# Print the new row
New_row_DF
#>   Training Pulse Duration
#> 1 Strength   100       60
#> 2  Stamina   150       30
#> 3    Other   120       45
#> 4 Strength   110      110
```

cbind()

```
# Add a new column
New_col_DF <- cbind(New_row_DF, Steps = c(1000, 6000, 2000,5000))

# Print the new column
New_col_DF
#>   Training Pulse Duration Steps
#> 1 Strength   100       60  1000
#> 2  Stamina   150       30  6000
#> 3    Other   120       45  2000
#> 4 Strength   110      110  5000
```

rbind() R

```
Data_Frame1 <- data.frame (
  Training = c("Strength", "Stamina", "Other"),
  Pulse = c(100, 150, 120),
  Duration = c(60, 30, 45)
)

Data_Frame2 <- data.frame (
  Training = c("Stamina", "Stamina", "Strength"),
  Pulse = c(140, 150, 160),
  Duration = c(30, 30, 20)
)

New_Data_Frame <- rbind(Data_Frame1, Data_Frame2)
New_Data_Frame
#>   Training Pulse Duration
#> 1 Strength   100       60
#> 2  Stamina   150       30
#> 3    Other   120       45
#> 4  Stamina   140       30
#> 5  Stamina   150       30
#> 6 Strength   160       20
```

cbind() R

```
Data_Frame3 <- data.frame (
  Training = c("Strength", "Stamina", "Other"),
  Pulse = c(100, 150, 120),
  Duration = c(60, 30, 45)
)
```

```
Data_Frame4 <- data.frame (
  Steps = c(3000, 6000, 2000),
  Calories = c(300, 400, 300)
)

New_Data_Frame1 <- cbind(Data_Frame3, Data_Frame4)
New_Data_Frame1
#>   Training Pulse Duration Steps Calories
#> 1 Strength   100       60  3000      300
#> 2  Stamina   150       30  6000      400
#> 3    Other   120       45  2000      300
```

Chapter 3

Raster

3.1 Resolution

What is resolution of a satellite image? How can we understand it? Generally speaking we can say there are **three** different resolutions for a satellite image in Geoscience.

3.1.1 Spatial resolution

Spatial resolution is the detail in pixels of an image. High spatial resolution means more detail and a smaller grid cell size. Whereas, lower spatial resolution means less detail and larger pixel size. Overall, spatial resolution describes the quality of an image and how detailed objects are in an image. If the grid cells are smaller, this means the spatial resolution has more detail with more pixels.

3.1.2 Temporal resolution

Same definition of temporal resolution can be applied to polar orbiting satellites. But defining it more precisely, temporal resolution for a polar orbiting satellite is the amount of time that the satellite takes to revisit and recapture a particular site. It is also commonly referred to as a satellite's revisit period.

3.1.3 Spectral resolution

Spectral resolution is determined by the width of each band in a wavelength. The more bands in an image, the more complex the color will be.

Figure 3.1: spatial resolution comparsion

Mission	Number of satellites	Temporal resolution (single satellite)	Temporal resolution (constellation)
SENTINEL-1	2	12 days	6 days
SENTINEL-2	2	10 days	5 days
LANDSAT 7	1	16 days	16 days
WorldView-3	1	1 day	1 day
Terra	1	16 days	16 days

Figure 3.2: Temporal resolution of some popular satellites

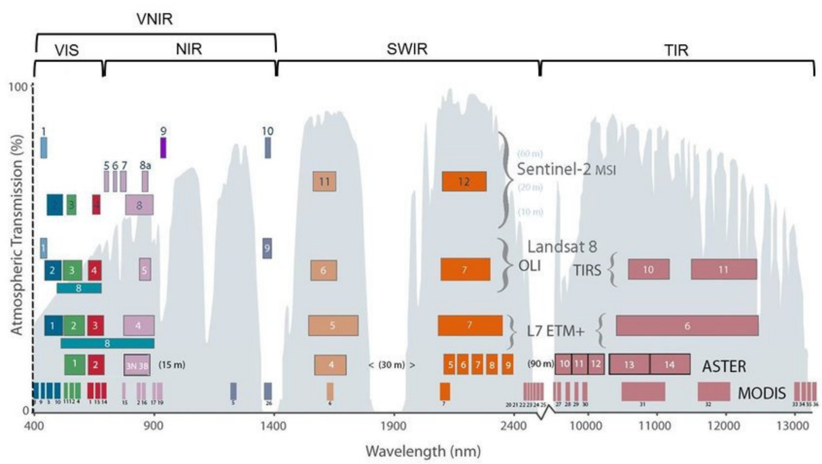

Figure 3.3: Spectral resolution of currently available optical satellite sensors grouped by different domains of the electromagnetic spectrum (VIS = visible, NIR = near infrared, VNIR = visible near infrared, SWIR = shortwave infrared, TIR = thermal infrared)

3.2 CRS

A Coordinate reference system (CRS) defines, with the help of coordinates, how the two-dimensional, projected map is related to real locations on the earth. There are two different types of coordinate reference systems: **`Geographic Coordinate Systems`** and **`Projected Coordinate Systems`**. CRS is very important for becoming a back-end developer of GIS.

3.3 Manipulation in R

How can we manipulate remote sensing imagery in R after we have a general understanding of raster data? Here I want to introduce to you a package: `terra`, which is a tweaked version or even a more powerful package than the well-known one: `raster`. As for the reason why the author choose to rebuild some functions, you can check it here.

3.4 Reference

- http://modern-rstats.eu/index.html#note-to-the-reader

Chapter 4

Multispectral sensors

In this section I'll introduce you to three well-known high-resolution multispectral sensor satellites in this section: MODIS, Landsat, and Sentinel-2.