R fundamentals

Jeikosd

04 September, 2017

Introduction

Installing R and R-Studio

- Base R **https://cran.r-project.org/mirrors.html**
- RStudio **https://www.rstudio.com/products/RStudio/**

What is R?

- it's a statistical software

What is R?

- it's a statistical software
- it's a object base

What is R?

- it's a statistical software
- it's a object base
    - Types of objects (scalar, vector, matrices, arrays and lists)

What is R?

- it's a statistical software
- it's a object base
    - Types of objects (scalar, vector, matrices, arrays and lists)
    - Assignment of objects

Why use R?

- Taken from Hadley Wickham “Fundamentally learning about the world through data is really, really good”
- it’s open source

R as calculator

```r
2+4
```

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

```r
sqrt(16)
```

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

```r
3*(2+4)
```

```
## [1] 18
```

More examples

Table 1: Operation Symbols

symbol	Meaning
+	Addition
-	Subtraction
*	Multiplication
/	Division
%%	Modulo (estimates remainder in a division)
^	Exponential

- **See http://www.statmethods.net/management/operators.html**

First Steps in R

Objects in R

- Objects in R obtain values by assignment.
- This is achieved by the gets arrow, <-, and not the equal sign, =.
- Objects can be of different kinds.

Types

- Primitives (numeric, integer, character, logical, factor)
- Data Frames
- Lists
- Tables
- Arrays
- Environments
- Others (functions, closures, promises..)

Simple Types - Vectors

The basic type unit in R is a vector

```
x <- c(1,2,3)
x
## [1] 1 2 3
x <- 1:3
x[1]
## [1] 1
x[0]
## integer(0)
x[-1]
## [1] 2 3
```

Generating Vectors

R provides lots of convenience functions for data generation:

```
rep(0, 5)
## [1] 0 0 0 0 0
seq(1,10)
##  [1]  1  2  3  4  5  6  7  8  9 10
seq(1,2,.1)
##  [1] 1.0 1.1 1.2 1.3 1.4 1.5 1.6 1.7 1.8 1.9 2.0
seq(1,2,length.out=6)
## [1] 1.0 1.2 1.4 1.6 1.8 2.0
```

Indexing

```
x <- c(1, 3, 4, 10, 15, 20, 50, 1, 6)
x > 10
## [1] FALSE FALSE FALSE FALSE  TRUE  TRUE  TRUE FALSE FALSE
which(x > 10)
## [1] 5 6 7
x[x>10]
## [1] 15 20 50
x[!x>10]
## [1]  1  3  4 10  1  6
x[x<=10]
## [1]  1  3  4 10  1  6
x[x>10 & x<30]
## [1] 15 20
```

Logical Operators

Table 2: Logical Operators

Operator	Description
$<$	less than
$<=$	less than or equal to
$>$	greather than
$>=$	greather than or equal to
$==$	exactly equal to
$!=$	not equal to

Functions

```r
square <- function(x) x^2
square(2)
## [1] 4

pow <- function(x, p=2) x^p
pow(10)
## [1] 100
pow(10,3)
## [1] 1000
pow(p=3,10)
## [1] 1000
```

Data Frames

- Data frames are the fundamental structure used in data analysis
- Similar to a database table in spirit (named columns, distinct types)

```
d <- data.frame(x=1:6, y="AUDUSD", z=c("one","two"))
d
```

```
##   x      y   z
## 1 1 AUDUSD one
## 2 2 AUDUSD two
## 3 3 AUDUSD one
## 4 4 AUDUSD two
## 5 5 AUDUSD one
## 6 6 AUDUSD two
```

Lists

```
d <- data.frame(x=1:6, y="AUDUSD", z=c("one","two"))
e <- data.frame(x=1:4, y="Center", z=c("one","two"))
f <- c(1, 2, 3)

g <- list(d, e,f)
f[[3]]
```

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

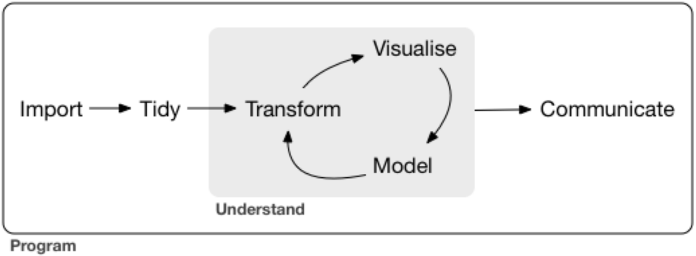
Installing Packages

There are some functions to make easier the management the information into R or to make a particular statistical method

```
install.packages('name')
```

World of Tidyverse

Why use tidyverse package



Why use tidyverse package

- Great for data exploration and transformation
- Intuitive to write and easy to read, especially when using the "chaining" syntax (covered below) Fast on data frames

Tidy Data

See the paper Tidy Data by Hadley Wickham in Journal of Statistical Software (2014)

- Each variable forms a column

Tidy Data

See the paper Tidy Data by Hadley Wickham in Journal of Statistical Software (2014)

- Each variable forms a column
- Each observation forms a row

Tidy Data

See the paper Tidy Data by Hadley Wickham in Journal of Statistical Software (2014)

- Each variable forms a column
- Each observation forms a row
- Each type of observational unit forms a table

Untidy Data

Table 3: Example of common untidy data

Station	Tmax.2014	Tmax.2015	Tmin.2014	Tmin.2015	Prec.2014	Prec.2015
1	32	33	25	26	0	200
2	28	26	19	20	164	0
3	19	18	12	14	0	10

Tidy Data

Table 4: Resulting tidy data set

Station	variable	year	Value
1	Tmax	2014	32
2	Tmax	2014	28
3	Tmax	2014	19
1	Tmax	2015	33
2	Tmax	2015	26
3	Tmax	2015	18
1	Tmin	2014	25
2	Tmin	2014	19
3	Tmin	2014	12
1	Tmin	2015	26
2	Tmin	2015	20
3	Tmin	2015	14

Installing Tidyverse

```r
install.packages('tidyverse')
```

Loading Packages

```
library('tidyverse')
```

Working with Tidyverse

Selecting

```
library('tidyverse')

x <- read_csv(file = 'data/weather.csv')
select(x, origin, temp)
select(x, origin, humid)
select(x, year, month, day, temp)
```

Filtering

```
filter(x, year == 2013)
filter(x, origin == 'EWR')
filter(x, origin == 'JFK')
filter(x, origin == 'JFK', temp >= 38, humid < 55)
```

Arranging

```r
arrange(x, temp)
arrange(x, desc(temp))
```

Mutate: Add new variables

```
mutate(x, temp = (temp - 32) * 5 /9)
mutate(x, dewp = (dewp - 32) * 5 /9)
mutate(x, y = temp / dewp)
```

“Chaining” or “Pipelining”

- Usual way to perform multiple operations in one line is by nesting.
- Can write commands in a natural order by using the %>% infix operator (which can be pronounced as “then”).
- Chaining increases readability significantly when there are many commands

```
x %>%
  select(origin, temp) %>%
  filter(origin == "EWR") %>%
  mutate(temp = (temp - 32) * 5 /9)
```

Summarise: Reduce variables to values

- Primarily useful with data that has been grouped by one or more variables
- group_by creates the groups that will be operated on
- summarise uses the provided aggregation function to summarise each group

```
x %>%
  group_by(origin) %>%
  summarise(avg_temp = mean(temp, na.rm = TRUE))


x %>%
  group_by(origin) %>%
  summarise(avg_temp = mean(temp, na.rm = TRUE),
            avg_dewp = mean(dewp, na.rm = TRUE))
```

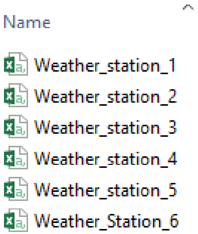
Summarise: Reduce variables to values

```
x %>%
  group_by(origin, month) %>%
  summarise(avg_temp = mean(temp, na.rm = TRUE))


x %>%
  group_by(month, hour) %>%
  summarise(avg_temp = mean(temp, na.rm = TRUE),
            avg_dewp = mean(dewp, na.rm = TRUE))
```

Looping

Bonus (How to load this information?)



A bad idea

```
library(tidyverse)
weather_station_1 <- read_csv(file = "data/climate/Weather_station_1.csv")
weather_station_2 <- read_csv(file = "data/climate/Weather_station_2.csv")
weather_station_3 <- read_csv(file = "data/climate/Weather_station_3.csv")
weather_station_4 <- read_csv(file = "data/climate/Weather_station_4.csv")
weather_station_5 <- read_csv(file = "data/climate/Weather_station_5.csv")
weather_station_6 <- read_csv(file = "data/climate/Weather_station_6.csv")
```

Loops?

```
climate <- list()
for(i in 1:6){

climate[[i]] <-
  read_csv(file = paste('data/climate/Weather_station_' , i, '.csv', sep = ''))

}


climate <- list.files('data/climate/', full.names = T)

climate <- lapply(climate, read_csv)

climate <- list.files('data/climate/', full.names = T) %>%
  lapply(read_csv)
```

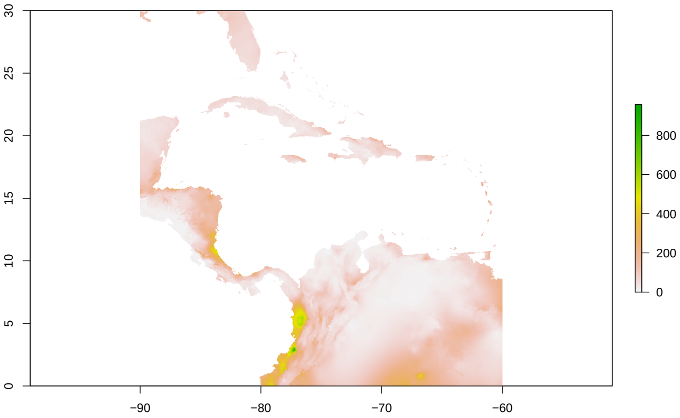
Spatial data into R

Libraries needed for the handling of spatial data

- raster
- rgdal
- sf
- sp

Loading a Raster file

```
library(raster)
prec <- raster('data/raster/prec/prec1_23.tif')
plot(prec)
```



Croping and Masking

```
library(raster)
prec <- raster('data/raster/prec/prec1_23.tif')
mask <- shapefile('data/shapefile/municipios_wgs84.shp')
mask_transf <- spTransform(mask, crs(prec))

crop(prec, mask_transf)
```

```
## class       : RasterLayer
## dimensions  : 1608, 1787, 2873496  (nrow, ncol, ncell)
## resolution  : 0.008333333, 0.008333333  (x, y)
## extent      : -81.74167, -66.85, 0, 13.4  (xmin, xmax, ymin, ymax)
## coord. ref. : +proj=longlat +datum=WGS84 +no_defs +ellps=WGS84 +towgs84=0,0,0
## data source : in memory
## names       : prec1_23
## values      : 0, 958  (min, max)
```

```
mask(prec, mask_transf)
```

```
## class       : RasterLayer
## dimensions  : 3600, 3600, 12960000  (nrow, ncol, ncell)
## resolution  : 0.008333333, 0.008333333  (x, y)
## extent      : -90, -60, 0, 30  (xmin, xmax, ymin, ymax)
## coord. ref. : +proj=longlat +datum=WGS84 +no_defs +ellps=WGS84 +towgs84=0,0,0
## data source : in memory
## names       : prec1_23
## values      : 0, 958  (min, max)
```

Loading multiple raster files

Let's GO!

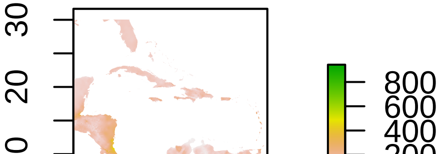
Loading multiple raster files

```
library(raster)
library(tidyverse)
prec <- list.files('data/raster/prec/', full.names = T) %>%
  lapply(raster)

prec_stack <- stack(prec)

avg_prec <- mean(prec_stack)
min_prec <- min(prec_stack)
max_prec <- max(prec_stack)

plot(avg_prec)
```



Loading multiple raster files

```
library(raster)
library(tidyverse)

prec <- list.files('data/raster/prec/', full.names = T) %>%
  stack()

tmax <- list.files('data/raster/tmax/', full.names = T) %>%
  stack()

tmin <- list.files('data/raster/tmin/', full.names = T) %>%
  stack()
```

Simple Features

Package sf

- it is in the world of tidyverse
- :)

Working with sf

```
library(tidyverse)
library(sf)
library(ggplot2)
library(viridis)
prd <- st_read(dsn = 'data/shapefile/Produccion_ton.shp')

plot(st_geometry(prd))
plot(prd["AREA_OF"])

filter(prd, NOM_DEP == 'META')
filter(prd, NOM_DEP == 'META', AREA_OF >= 6000)

avg <- group_by(prd, NOM_DEP) %>%
  summarise(avg_area= mean(AREA_OF, na.rm = TRUE))

# devtools::install_github("tidyverse/ggplot2")

ggplot() +
  geom_sf(data = avg, aes(fill = avg_area)) +
  scale_fill_viridis("Area") +
  ggtitle("") +
  theme_bw()
```