Leer y escribir datos

HR Analytics: Teoría y Práctica

http://pablohaya.com/contact

09/2018

Leer un archivo CSV

R permite leer los formatos más populares (csv, json, xml, xls, xlsx...).

El formato **CSV** (comma-separated values) almacena tablas donde cada fila ocupa una línea, y las columnas se separan por comas.

```
library(here)
df <- read.csv(here("data", "core_dataset.csv"))
head(df)
```

```
##          Employee.Name Employee.Number State  Zip         DO
## 1           Brown, Mia      1103024456    MA 1450 11/24/198
## 2 LaRotonda, William      1106026572    MA 1460  4/26/198
## 3    Steans, Tyrone       1302053333    MA 2703   9/1/198
## 4     Howard, Estelle     1211050782    MA 2170  9/16/198
## 5         Singh, Nan      1307059817    MA 2330  5/19/198
## 6    Smith, Leigh Ann      711007713    MA 1844  6/14/198
##   MaritalDesc CitizenDesc Hispanic.Latino
```

La función `read.csv()` es equivalente a llamar a la función `read.table()` indicando que la primera fila es la cabecera que contiene el nombre de las variables, y que el separador de celdas es la coma.

```
library(here)
df <- read.table(here("data", "core_dataset.csv"),
                 header = TRUE, sep = ",")
head(df)
```

```
##           Employee.Name Employee.Number State  Zip         DO
## 1           Brown, Mia      1103024456    MA 1450 11/24/198
## 2 LaRotonda, William      1106026572    MA 1460  4/26/198
## 3     Steans, Tyrone      1302053333    MA 2703   9/1/198
## 4      Howard, Estelle      1211050782    MA 2170  9/16/198
## 5          Singh, Nan      1307059817    MA 2330  5/19/198
## 6     Smith, Leigh Ann       711007713    MA 1844  6/14/198
##   MaritalDesc CitizenDesc Hispanic.Latino
## 1     Married  US Citizen              No Black or African
## 2    Divorced  US Citizen              No Black or African
```

Lectura avanzada

El paquete `tidyverse()` incluye funciones mejoradas para importar datos. La función `read_csv()` permite leer un fichero csv siendo más rápida y versatil que la función por defecto. La función `read_delim()` es su versión más genérica.

```
library(tidyverse)
df <- read_csv(here("data","core_dataset.csv"))
glimpse(df)
```

```
## Observations: 302
## Variables: 21
## $ `Employee Name`      <chr> "Brown, Mia", "LaRotonda, Wil
## $ `Employee Number`    <chr> "1103024456", "1106026572", "
## $ State                <chr> "MA", "MA", "MA", "MA", "MA",
## $ Zip                  <chr> "01450", "01460", "02703", "0
## $ DOB                  <chr> "11/24/1985", "4/26/1984", "9
```

La función `read_csv()` devuelve un tipo de datos `tibble` que es una versión mejorada del `data.frame` que viene por defecto.

```
class(df)
```

```
## [1] "tbl_df"     "tbl"        "data.frame"
```

```
head(df)
```

```
## # A tibble: 6 x 21
##   `Employee Name` `Employee Numbe~ State Zip   DOB      Age
##   <chr>           <chr>            <chr> <chr> <chr> <int>
## 1 Brown, Mia      1103024456       MA    01450 11/2~    32
## 2 LaRotonda, Wil~ 1106026572       MA    01460 4/26~    33
## 3 Steans, Tyrone  1302053333       MA    02703 9/1/~    31
## 4 Howard, Estelle 1211050782       MA    02170 9/16~    32
## 5 Singh, Nan      1307059817       MA    02330 5/19~    29
## 6 Smith, Leigh A~ 0711007713       MA    01844 6/14~    30
## # ... with 14 more variables: MaritalDesc <chr>, CitizenDes
```

Limpieza del conjunto de datos

Es habitual que los conjuntos de datos que nos faciliten tengan errores de formato que dificulten el análisis. Vamos a ver un ejemplo sobre este conjunto de datos.

Una vez leido, comprobamos las columnas que tiene, y el tipo de dato de cada una.

```
glimpse(df)
```

```
## Observations: 302
## Variables: 21
## $ `Employee Name`      <chr> "Brown, Mia", "LaRotonda, Wil
## $ `Employee Number`    <chr> "1103024456", "1106026572", "
## $ State                <chr> "MA", "MA", "MA", "MA", "MA",
## $ Zip                  <chr> "01450", "01460", "02703", "0
## $ DOB                  <chr> "11/24/1985", "4/26/1984", "9
## $ Age                  <int> 32, 33, 31, 32, 29, 30, 33, 3
```

Los nombres de las columnas solo deberían tener caracteres alfanuméricos. No deberían contener espacios, aunque si se permite _ como separador.

Así que renombramos las columnas eleminándolos.

```
colnames(df) <- str_replace_all(colnames(df), "[ /]", "")
glimpse(df)
```

```
## Observations: 302
## Variables: 21
## $ EmployeeName      <chr> "Brown, Mia", "LaRotonda, Willian
## $ EmployeeNumber    <chr> "1103024456", "1106026572", "1302
## $ State             <chr> "MA", "MA", "MA", "MA", "MA", "MA
## $ Zip               <chr> "01450", "01460", "02703", "02170
## $ DOB               <chr> "11/24/1985", "4/26/1984", "9/1/1
## $ Age               <int> 32, 33, 31, 32, 29, 30, 33, 33, 3
## $ Sex               <chr> "Female", "Male", "Male", "Female
## $ MaritalDesc       <chr> "Married", "Divorced", "Single",
## $ CitizenDesc       <chr> "US Citizen", "US Citizen", "US C
```

El formato estándar de fecha es “YYYY-mm-dd”. En caso contrario, se interpretan como una cadena de caracteres.

Transformamos las variables que son de tipo fecha mediante la función `parse_cols()`

```
date_cols <- c("DOB", "DateofHire", "DateofTermination")
df[, date_cols] <- map(df[, date_cols],
                       parse_date, "%m/%d/%Y")
```

La función `read_csv()` evita definir factores, de manera que es el usuario el que explícitamente indica que lo son.

Empleamos la función `map()` para ejecutar una función, en este caso `factor()` sobre un conjunto de vectores.

```
fct_cols <- c("State","Zip", "Sex", "MaritalDesc",
              "CitizenDesc", "HispanicLatino",
              "RaceDesc", "ReasonForTerm",
              "EmploymentStatus", "Department", "Position",
              "EmployeeSource", "PerformanceScore")
df[, fct_cols] <- map(df[, fct_cols], factor)
```

```
glimpse(df)
```

```
## Observations: 302
## Variables: 21
## $ EmployeeName      <chr> "Brown, Mia", "LaRotonda, William
## $ EmployeeNumber    <chr> "1103024456", "1106026572", "1302
## $ State             <fct> MA, MA, MA, MA, MA, MA, MA, MA, M
## $ Zip               <fct> 01450, 01460, 02703, 02170, 02330
## $ DOB               <date> 1985-11-24, 1984-04-26, 1986-09-
## $ Age               <int> 32, 33, 31, 32, 29, 30, 33, 33, 3
## $ Sex               <fct> Female, Male, Male, Female, Femal
## $ MaritalDesc       <fct> Married, Divorced, Single, Marrie
## $ CitizenDesc       <fct> US Citizen, US Citizen, US Citize
## $ HispanicLatino    <fct> No, No, No, No, No, No, No, No, N
## $ RaceDesc          <fct> Black or African American, Black
## $ DateofHire        <date> 2008-10-27, 2014-01-06, 2014-09-
## $ DateofTermination <date> NA, NA, NA, 2015-04-15, NA, 2013
## $ ReasonForTerm     <fct> N/A - still employed, N/A - still
```

Comprobamos todos los niveles de los factores que hemos convertido, y nos damos cuenta que hay problemas con los valores en `Sex` y en `HispanicLatino`.

```
map(df[, fct_cols], table)
glimpse(df)
```

```
table(df$Sex)
```

```
##
## Female   male   Male
##    174      1    126
```

```
table(df$HispanicLatino)
```

```
##
##  no  No yes Yes
##   2 271   1  27
```

Corregimos los valores del género, y volvemos a convertir en factor.

```
df$Sex <- str_replace(df$Sex, "^m", "M")
df$Sex <- factor(df$Sex)
```

Corregimos los valores de la variable `HispanicLatino`, y volvemos a convertir en factor.

```
df$HispanicLatino <- str_replace(df$HispanicLatino,"no","No")
df$HispanicLatino <- str_replace(df$HispanicLatino,"yes","Yes"
df$HispanicLatino <- factor(df$HispanicLatino)
```

```
glimpse(df)
```

```
## Observations: 302
## Variables: 21
## $ EmployeeName      <chr> "Brown, Mia", "LaRotonda, Willian
## $ EmployeeNumber    <chr> "1103024456", "1106026572", "1302
## $ State             <fct> MA, MA, MA, MA, MA, MA, MA, MA, M
## $ Zip               <fct> 01450, 01460, 02703, 02170, 02330
## $ DOB               <date> 1985-11-24, 1984-04-26, 1986-09-
## $ Age               <int> 32, 33, 31, 32, 29, 30, 33, 33, 3
## $ Sex               <fct> Female, Male, Male, Female, Femal
## $ MaritalDesc       <fct> Married, Divorced, Single, Marrie
## $ CitizenDesc       <fct> US Citizen, US Citizen, US Citize
## $ HispanicLatino    <fct> No, No, No, No, No, No, No, No, N
## $ RaceDesc          <fct> Black or African American, Black
## $ DateofHire        <date> 2008-10-27, 2014-01-06, 2014-09-
## $ DateofTermination <date> NA, NA, NA, 2015-04-15, NA, 2013
## $ ReasonForTerm     <fct> N/A - still employed, N/A - still
```

Escribir datos

Se pueden emplear para escribir en formato csv las funciones equivalentes `write.csv()` y `write.table()`

```
write.csv(df, file = here("data","core_dataset_clean.csv"))
```

De la misma manera, se pueden emplear las funciones para exportar datos `write_csv()` y `write_deliim()` del paquete `tidyverse`

```
write_csv(df,  here("data","core_dataset_clean.csv"))
```

Guardar datos binarios

Si se quieren guardar datos de manera temporal es más efectivo emplear la función `save()` que los guarda en binario. Veremos en el siguiente tema como se recupera con la función `load()`.

```
save(df, file = here("data","core_dataset_clean.RData"))
```

Referencias y recursos online

- How to share data with a statiscian: resumen de cómo organizar el conjunto de datos, y los metadatos asociados para que sea más sencillo realizar los análisis.
- Human Resource Dataset: conjunto de datos publicado en Kaggle que se utiliza para este curso
- Documentación here: paquete que permite determinar la ruta de un fichero relativa al directorio donde comienza el proyecto, e independiente de donde esté ubicado el script.
- Documentación stringr: paquete que incluye funcionaes para manipular cadenas de caracteres (ej. `str_replace` que reemplaza una cadena por otra).
- RStudio Import Cheatsheet: los comanddos más utilizados en la importación de datos.