09.1.- Data Cleansing-Basic_45_06_turismo_origen_completo_v_01

June 11, 2023

#

CU45_Planificación y promoción del destino en base a los patrones en origen de los turistas

Citizenlab Data Science Methodology > II - Data Processing Domain *** > # 09.1.- Data Cleansing - Basic

Data Cleaning refers to identifying and correcting (or removing) errors in the dataset that may negatively impact a predictive model, replacing, modifying, or deleting the dirty or coarse data.

0.1 Tasks

Basic operations	Text data analysis
	Delete Needless/Irrelevant/Private Columns
	Inconsistent Data.
	Expected values
	Zeroes
	Columns with a Single Value
	Columns with Very Few Values
	Columns with Low Variance
	Duplicates (rows/samples) & (columns/features) Data
Missing Values	Missing Values Identification
	Missing Values Per Sample
	Missing Values Per Feature
	Zero Missing Values
	Other Missing Values
	Null/NaN Missing Values
	Delete Missing Values
	Deleting Rows with Missing Values in Target Column
	Deleting Rows with Missing Values
	Deleting Features with some Missing Values
	Deleting Features using Rate Missing Values
	Basic Imputation
	Imputation by Previous Row Value
	Imputation by Next Row Value
	Statistical Imputation
	Selection of Imputation Strategy
	Constant Imputation
	Mean Imputation
	Median Imputation
	Most Frequent Imputation
	Interpolation Imputation
	Prediction Imputation (KNN Imputation)
	Evaluating k-hyperparmeter in KNN Imputation
	Applying KNN Imputation
	Iterative Imputation
	Evaluating Different Imputation Order
	Applying Iterative Imputation
Outliers	Outliers - Univariate
	Visualizing Outliers
	Distribution
	Box Plots
	Isolation Forest
	Outliers Identification
	Grubbs' Test
	Z-Score
	Standard Deviation Method
	Interquartile Range Method
	Tukey's method
	Internally studentized residuals AKA z-score method
	Median Absolute Deviation method
	Outliers - MultiVariate
	Visualizing Outliers
	ScatterPlots
	Outliers Identification
	Mahalanobis Distance
	Robust Mahalanobis Distance
	DBSCAN Clustering
	PyOD Library
	Automatic Detection and Removal of Outliers
	Compare Algorithms
	LocalOutlierFactor
	IsolationForest
	Minimum Covariance Determinant

0.2 Consideraciones casos CitizenLab programados en R

- La mayoría de las tareas de este proceso se han realizado en los notebooks del proceso 05 Data Collection porque eran necesarias para las tareas ETL. En esos casos, en este notebook se referencia al notebook del proceso 05 correspondiente
- Por tanto en los notebooks de este proceso de manera general se incluyen las comprobaciones necesarias, y comentarios si procede
- Las tareas del proceso se van a aplicar solo a los archivos que forman parte del despliegue, ya que hay muchos archivos intermedios que no procede pasar por este proceso
- El nombre de archivo del notebook hace referencia al nombre de archivo del proceso 05 al que se aplica este proceso, por eso pueden no ser correlativa la numeración
- Las comprobaciones se van a realizar teniendo en cuenta que el lenguaje utilizado en el despliegue de este caso es R

0.3 File

- Input File: CU_45_06_03_turismo_receptor.csv
- Sampled Input File: CU_45_07_03_turismo_receptor.csv
- Output File: CU_45_09.1_03_turismo_receptor.csv

0.4 Settings

0.4.1 Encoding

Con la siguiente expresión se evitan problemas con el encoding al ejecutar el notebook. Es posible que deba ser eliminada o adaptada a la máquina en la que se ejecute el código.

[3]:
```
Sys.setlocale(category = "LC_ALL", locale = "es_ES.UTF-8")
```

```
Warning message in Sys.setlocale(category = "LC_ALL", locale = "es_ES.UTF-8"):
"OS reports request to set locale to "es_ES.UTF-8" cannot be honored"
"
```

0.4.2 Libraries to use

[4]:
```
library(readr)
library(dplyr)
library(tidyr)
library(stringr)
```

0.4.3 Paths

[5]:
```
iPath <- "Data/Input/"
oPath <- "Data/Output/"
```

0.5 Data Load

OPCION A: Seleccionar fichero en ventana para mayor comodidad

Data load using the {tcltk} package. Ucomment the line if using this option

[6]:
```
# file_data <- tcltk::tk_choose.files(multi = FALSE)
```

OPCION B: Especificar el nombre de archivo

[7]:
```
iFile <- "CU_45_06_03_turismo_receptor.csv"
file_data <- paste0(iPath, iFile)

if(file.exists(file_data)){
    cat("Se leerán datos del archivo: ", file_data)
} else{
    warning("Cuidado: el archivo no existe.")
}
```

```
Se leerán datos del archivo:  Data/Input/CU_45_06_03_turismo_receptor.csv
```

Data file to dataframe Usar la función adecuada según el formato de entrada (xlsx, csv, json, ...)

[8]:
```
data <- read_csv(file_data)
```

```
Rows: 50294 Columns: 8
  Column specification



Delimiter: ","
chr (5): mes, pais_orig_cod, pais_orig, mun_dest, CMUN
dbl (3): mun_dest_cod, turistas, Target

 Use `spec()` to retrieve the full column specification for this
data.
 Specify the column types or set `show_col_types = FALSE` to quiet
this message.
```

Visualizo los datos.

Estructura de los datos:

[9]:
```
data |> glimpse()
```

```
Rows: 50,294
Columns: 8
$ mes           <chr> "2019-07", "2019-07", "2019-07",
"2019-07", "2019-07", "…
$ pais_orig_cod <chr> "000", "010", "011", "030", "110",
"121", "123", "126", …
$ pais_orig     <chr> "Total", "Total Europa", "Total Unión
Europea", "Total A…
```

```
$ mun_dest_cod  <dbl> 28002, 28002, 28002, 28002, 28002,
28002, 28002, 28002, …
$ mun_dest      <chr> "Ajalvir", "Ajalvir", "Ajalvir",
"Ajalvir", "Ajalvir", "…
$ turistas      <dbl> 338, 290, 268, 37, 56, 54, 37, 40, 157,
116, 109, 8461, …
$ CMUN          <chr> "002", "002", "002", "002", "002",
"002", "002", "002", …
$ Target        <dbl> 338, 290, 268, 37, 56, 54, 37, 40, 157,
116, 109, 8461, …
```

Muestra de los primeros datos:

[10]:
```
data |> slice_head(n = 5)
```

A spec_tbl_df: 5 × 8

mes <chr>	pais_orig_cod <chr>	pais_orig <chr>	mun_dest_cod <dbl>	mun_dest <chr>	turistas <dbl>
2019-07	000	Total	28002	Ajalvir	338
2019-07	010	Total Europa	28002	Ajalvir	290
2019-07	011	Total Unión Europea	28002	Ajalvir	268
2019-07	030	Total América	28002	Ajalvir	37
2019-07	110	Francia	28002	Ajalvir	56

0.6 Text data analysis

Select columns

[11]:
```
# Select column
text_columns <- sapply(data, is.character)
```

Operation

[12]:
```
# Analizar datos de texto y verificar su corrección
# e.g. faltas ortografía, etc
```

[13]:
```
# pasar a mayúsculas todas las columnas de texto
data[, text_columns] <- lapply(data[, text_columns], function(x) toupper(x))
```

0.7 Delete Columns Needless/Irrelevant/Private

Select columns

[14]:
```
# Select columns
```

Operation

[15]:
```
# Eliminamos columnas que consideramos irrelevantes o innecesarias
```

Todas las columnas son relevantes, por lo que no aplica.

0.8 Inconsistent Data

[ ]:
```

```

Select columns and value

[16]:
```
# Select column and value
# e.g. age > 100
numeric_columns <- sapply(data, is.numeric)
```

Operation

[18]:
```
# Inconsistent data is unique to each data set and
# must be searched manually
data[, numeric_columns] <- lapply(data[, numeric_columns], function(x) {
  ifelse(is.na(x), ifelse(is.integer(x), as.integer(mean(x, na.rm = TRUE)),␣
 ↪mean(x, na.rm = TRUE)), x)
})
```

0.9 Expected values

[19]:
```
# Check for expected value
nan_counts <- colSums(is.na(data))
nan_counts
```

mes 0 **pais_orig_cod** 0 **pais_orig** 0 **mun_dest_cod** 0 **mun_dest** 0 **turistas** 0 **CMUN** 0 **Target** 0

0.10 Zeros

No aplica. ETL satisface los requisitos de calidad de los datos para valores cero.

0.11 Single Value

[20]:
```
# We obtain the number of different values of each column
distinct_counts <- sapply(data, function(x) n_distinct(x, na.rm = TRUE))
distinct_counts
```

mes 40 **pais_orig_cod** 146 **pais_orig** 146 **mun_dest_cod** 172 **mun_dest** 172 **turistas** 4103 **CMUN** 172 **Target** 4103

[21]:
```
# Columns with a single unique value
#
# Identify columns with a single unique value
cols_to_remove <- sapply(data, function(x) length(unique(x)) == 1)
data <- data[, !cols_to_remove]
```

0.12 Very Few Values

Select rate

[22]:
```
# Select rate
threshold <- 0.8
```

Operation

[23]:
```
# Show features with over rate rows being the same value
cols_to_keep <- sapply(data, function(x) {
  freqs <- table(x) / length(x)
  max(freqs) <= threshold
})
print(cols_to_keep)
```

```
         mes pais_orig_cod     pais_orig  mun_dest_cod      mun_dest
        TRUE          TRUE          TRUE          TRUE          TRUE
    turistas          CMUN        Target
        TRUE          TRUE          TRUE
```

[ ]:
```
```

[24]:
```
# Summarize the number of unique values in each column
# followed by the percentage of unique values for each
# variable as a percentage of the total number of rows
# in the dataset.

# First, find the number of unique values in each column
num_unique_values <- sapply(data, function(x) length(unique(x)))

# Then, calculate the percentage of unique values as a proportion of total rows
percentage_unique_values <- num_unique_values / nrow(data) * 100

# Finally, create a data frame to summarize the results
summary_df <- data.frame(
  Column = names(data),
  UniqueValues = num_unique_values,
  PercentageOfUniqueValues = percentage_unique_values
)

# Print the summary
print(summary_df)
```

```
                    Column UniqueValues PercentageOfUniqueValues
mes                    mes           40               0.07953235
pais_orig_cod pais_orig_cod          146               0.29029308
pais_orig         pais_orig          146               0.29029308
mun_dest_cod   mun_dest_cod          172               0.34198910
```

```
mun_dest          mun_dest          172              0.34198910
turistas          turistas         4103              8.15803078
CMUN                  CMUN          172              0.34198910
Target              Target         4103              8.15803078
```

[ ]:

0.13 Low Variance

No aplica este preprocesamiento al caso tratado.

A) Calculating variances

[28]:
```
# # calculate variance for all columns
# variances <- sapply(data, var, na.rm = TRUE)

# # print the variances
# print(variances)
```

B) Automatic calculation and representation of variances

[ ]:

Define thresholds to check

[29]:
```
# # define thresholds to check
# thresholds = 0.8
```

Operation

[ ]:

C) Delete variables with low variance

Select column

[30]:
```
# # Identify numeric columns
# numeric_cols <- sapply(data, is.numeric)

# # Calculate variance for numeric columns only
# numeric_variances <- sapply(data[, numeric_cols], var, na.rm = TRUE)

# # Set a threshold for variance
# var_threshold = 0.1

# # Find columns that have variance greater than the threshold
# cols_to_keep <- c(!numeric_cols, numeric_cols_to_keep)
```

Operation

[82]:
```
# # Keep only those columns
# data <- data[, cols_to_keep]
```

[ ]:
```

```

0.14 Duplicates

Entendido como ERROR –> Eliminar duplicados

[31]:
```
data <- data[!duplicated(data), ]
```

0.15 Data Save

- Solo si se han hecho cambios
- No aplica

Identificamos los datos a guardar

[32]:
```
data_to_save <- data
```

Estructura de nombre de archivos:

- Código del caso de uso, por ejemplo “CU_04”
- Número del proceso que lo genera, por ejemplo ”_06”.
- Resto del nombre del archivo de entrada
- Extensión del archivo

Ejemplo: ”CU_04_06_01_01_zonasgeo.json, primer fichero que se genera en la tarea 01 del proceso 05 (Data Collection) para el caso de uso 04 (vacunas) y que se ha transformado en el proceso 06

Importante mantener los guiones bajos antes de proceso, tarea, archivo y nombre

0.15.1 Proceso 09.1

[33]:
```
caso <- "CU_45"
proceso <- '_09.1'
tarea <- "_03"
archivo <- ""
proper <- "_turismo_receptor"
extension <- ".csv"
```

OPCION A: Uso del paquete “tcltk” para mayor comodidad

- Buscar carpeta, escribir nombre de archivo SIN extensión (se especifica en el código)
- Especificar sufijo2 si es necesario
- Cambiar datos por datos_xx si es necesario

[34]:
```
# file_save <- paste0(caso, proceso, tarea, tcltk::tkgetSaveFile(), proper,␣
↪extension)
```

```
# path_out <- paste0(oPath, file_save)
# write_csv(data_to_save_xxxx, path_out)

# cat('File saved as: ')
# path_out
```

OPCION B: Especificar el nombre de archivo

- Los ficheros de salida del proceso van siempre a Data/Output/.

[35]:
```
file_save <- paste0(caso, proceso, tarea, archivo, proper, extension)
path_out <- paste0(oPath, file_save)
write_csv(data, path_out)

cat('File saved as: ')
path_out
```

```
File saved as:
```

'Data/Output/CU_45_09.1_03_turismo_receptor.csv'

Copia del fichero a Input Si el archivo se va a usar en otros notebooks, copiar a la carpeta Input

[36]:
```
path_in <- paste0(iPath, file_save)
file.copy(path_out, path_in, overwrite = TRUE)
```

TRUE

0.16 REPORT

A continuación se realizará un informe de las acciones realizadas

0.17 Main Actions Carried Out

- Si eran necesarias se han realizado en el proceso 05 por cuestiones de eficiencia

0.18 Main Conclusions

- Los datos están limpios para el despliegue

0.19 CODE TO DEPLOY (PILOT)

A continuación se incluirá el código que deba ser llevado a despliegue para producción, dado que se entiende efectúa operaciones necesarias sobre los datos en la ejecución del prototipo

Description

- No hay nada que desplegar en el piloto, ya que estos datos son estáticos o en todo caso cambian con muy poca frecuencia, altamente improbable durante el proyecto.

CODE

```
[ ]:
```