09.2.- Data Cleansing-Missing_CU_18_20_infra_meteo_v_01

June 13, 2023

#

CU18_Infraestructuras_eventos

Citizenlab Data Science Methodology > II - Data Processing Domain *** > # 09.2.- Data Cleansing - Missing

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_18_09.1_20_diario_infra
- Output File: CU_18_09.2_20_diario_infra

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.

[1]:
```
Sys.setlocale(category = "LC_ALL", locale = "es_ES.UTF-8")
```

'LC_CTYPE=es_ES.UTF-8;LC_NUMERIC=C;LC_TIME=es_ES.UTF-8;LC_COLLATE=es_ES.UTF-8;LC_MONETARY=es_ES.UTF-8;LC_MESSAGES=en_US.UTF-8;LC_PAPER=es_ES.UTF-8;LC_NAME=C;LC_ADDRESS=C;LC_TELEPHONE=C;LC_MEASUREMENT=es_ES.UTF-8;LC_IDENTIFICATION=C'

0.4.2 Libraries to use

[2]:
```
library(readr)
library(dplyr)
# library(sf)
library(tidyr)
library(stringr)
```

```
Attaching package: 'dplyr'


The following objects are masked from 'package:stats':

    filter, lag
```

```
The following objects are masked from 'package:base':

    intersect, setdiff, setequal, union
```

0.4.3 Paths

[3]:
```
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

[4]:
```
# file_data <- tcltk::tk_choose.files(multi = FALSE)
```

OPCION B: Especificar el nombre de archivo

[5]:
```
iFile <- "CU_18_20_09.1_diario_infra.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_18_20_09.1_diario_infra.csv
```

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

[6]:
```
data <- read_csv(file_data)
```

```
Rows: 415370 Columns: 10
  Column specification


Delimiter: ","
dbl  (9): id_inf, capacidad, demanda, evento_infra, evento_zona, tmed,
prec,…
date (1): fecha

 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:

[7]:
```
data |> glimpse()
```

```
Rows: 415,370
Columns: 10
$ id_inf       <dbl> 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12,
13, 14, 15, 16, 17…
$ fecha        <date> 2019-01-01, 2019-01-01, 2019-01-01,
2019-01-01, 2019-01-…
$ capacidad    <dbl> 993, 996, 1036, 1020, 992, 1026, 1007,
976, 1037, 972, 94…
$ demanda      <dbl> 883, 888, 922, 1134, 1103, 1139, 897,
1086, 1150, 861, 83…
$ evento_infra <dbl> 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 1, 1, 0,
0, 0, 1, 0, 1, 1, …
$ evento_zona  <dbl> 1, 0, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 0,
1, 1, 1, 1, 0, 1, …
$ tmed         <dbl> 6.953211, 6.196420, 6.483569, 5.875797,
6.212680, 5.87854…
$ prec         <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, …
$ velmedia     <dbl> 0.6433886, 0.3417523, 0.4132169,
0.1820178, 0.2118110, 0.…
$ presMax      <dbl> 952.9357, 950.2191, 950.8051, 951.6768,
953.5118, 952.168…
```

Muestra de los primeros datos:

[8]:
```
data |> slice_head(n = 5)
```

A spec_tbl_df: 5 × 10

id_inf <dbl>	fecha <date>	capacidad <dbl>	demanda <dbl>	evento_infra <dbl>	evento_zona <dbl>	tmed <dbl>	p <dbl>
1	2019-01-01	993	883	1	1	6.953211	0
2	2019-01-01	996	888	0	0	6.196420	0
3	2019-01-01	1036	922	0	0	6.483569	0
4	2019-01-01	1020	1134	1	0	5.875797	0
5	2019-01-01	992	1103	1	1	6.212680	0

0.6 Missing Values

0.6.1 Missing Values Identification

Missing Values Per Sample

[9]:
```
missing_values <- is.na(data)

missing_values_per_sample <- rowSums(missing_values)
```

Missing Values Per Feature

[10]:
```
missing_values_per_feature <- colSums(missing_values)
missing_values_per_feature
```

id_inf 0 **fecha** 0 **capacidad** 0 **demanda** 0 **evento_infra** 0 **evento_zona** 0 **tmed** 25036 **prec** 9104 **velmedia** 3414 **presMax** 0

Zero Missing Values

[11]:
```
# Detecting columns with minimum value of zero (0).
# zero_values <- sapply(data, function(x) any(x == 0))
```

[12]:
```
# Frecuency of values by column
# value_counts <- sapply(data, table)
```

Select column to replace

[13]:
```
# Select column to replace
# column_to_replace <- "nombre_columna"
```

Operation

[14]:
```
# Replace zero missing values by nan
# data[[column_to_replace]] <- ifelse(data[[column_to_replace]] == 0, NA,␣
↪data[[column_to_replace]])
```

Other Missing Values Select column to replace

[15]:
```
# Select column to replace and missing value
```

Operation

[16]:
```
# Replace other missing values by nan
```

Null/NaN Missing Values

[17]:
```
# Intuitivamente: miramos nº datos en todas las columnas
# los null no los cuenta --> debe hacer el mismo nº por columna
```

[18]:
```
# Podemos mirar directamente info donde viene
```

[19]:
```
# Contamos los nulos de forma explícita
```

[20]:
```
# summarize the number of rows with missing values for each column
```

[ ]:

0.6.2 Delete Missing Values

Deleting Rows with Missing Values in Target Column

[21]:
```
# columna_objetivo <- "nombre_columna"
# data <- data[!is.na(data$columna_objetivo), ]
```

Deleting Rows with Missing Values Only in case of high data size

[22]:
```
# Eliminamos las filas con valores nulos
data <- na.omit(data)
```

Deleting Features with some Missing Values Only with many features and for non-relevant features

[23]:
```
# Selecciono las columnas con algún valor missing:
# data <- data[, colSums(is.na(data)) == 0]
```

Deleting Features using Rate Missing Values

[24]:
```
# Number of data
# num_total <- nrow(data)
```

[25]:
```
# Number of missing data
# num_missing <- colSums(is.na(data))
```

[26]:
```
# Rate (%) of missing data
# rate_missing <- num_missing / num_total
```

Select column to delete

[27]:
```
# Select column to delete
# umbral <- 0.2
# features_to_remove <- names(rate_missing[rate_missing > umbral])
```

Operation

[28]:
```
# Deleting Features selected
# data <- data[ , !(names(data) %in% features_to_remove)]
```

0.6.3 Basic Imputation

Imputation by Previous Row Value

[29]:
```
# Sustituimos valores null por otro valor: VALOR FILA ANTERIOR
# data_imputed_previous <- data %>% fill(everything(), .direction = "down")
```

Imputation by Next Row Value

[30]:
```
# Sustituimos valores null por otro valor: VALOR FILA SIGUIENTE
# data_imputed_next <- data %>% fill(everything(), .direction = "up")
```

0.6.4 Statistical Imputation

A popular approach for data imputation is to calculate a statistical value for each column (such as a mean) and replace all missing values for that column with the statistic.

Selection of Imputation Strategy

[31]:
```
# The mean accuracy of each approach can then be compared.
#
# Specific results may vary given the stochastic nature of
# the learning algorithm, the evaluation procedure, or
# differences in numerical precision. Consider running the
# example a few times and compare the average performance.
#
```

[32]:
```
# Plot model performance for comparison
# box and whisker plot is created for each set of results,
# allowing the distribution of results to be compared.
```

Constant Imputation Select constant value

[33]:
```
# Select constant value
constant = 0
```

Operation

[34]:
```
# Constant imputation
# data <- replace_na(data, list(everything() <- constant))
```

Mean Imputation

[35]:
```
# Identify numeric columns
# numeric_cols <- sapply(data, is.numeric)

# # Apply mean imputation only on numeric columns
# data_imputed <- data
# for(column in names(data)[numeric_cols]) {
#   # Check if the column is truly numeric
#   if(is.numeric(data[[column]])) {
#     # Check if column has missing values
#     if(anyNA(data[[column]])) {
#       data_imputed[is.na(data_imputed[[column]]), column] <-␣
 ↪mean(data[[column]], na.rm = TRUE)
#     }
```

```
#   } else {
#     print(paste("Column", column, "is not numeric. Mean imputation skipped."))
#   }
# }
```

[36]:
```
# Sustituyo
```

Median Imputation

[37]:
```
# Identify numeric columns
# numeric_cols <- sapply(data, is.numeric)

# # Apply median imputation only on numeric columns
# data_imputed <- data
# for(column in names(data)[numeric_cols]) {
#     data_imputed[is.na(data_imputed[,column]), column] <-␣
 ↪median(data[,column], na.rm = TRUE)
# }
```

[38]:
```
# Sustituyo
```

Most Frequent Imputation

[39]:
```
# Create a function to calculate the mode
# getmode <- function(v) {
#    uniqv <- unique(v)
#    uniqv[which.max(tabulate(match(v, uniqv)))]
# }

# # Apply most frequent imputation to each column
# data_imputed <- data
# for(column in names(data)) {
#     data_imputed[is.na(data_imputed[,column]), column] <-␣
 ↪getmode(data[,column])
# }
```

[40]:
```
# Sustituyo
```

Interpolation Imputation

[41]:
```
# Sustituimos valores null por otro valor: INTERPOLANDO
# Métodos de interpolación
# 'linear', 'time', 'index', 'values', 'nearest', 'zero', 'slinear',
# 'quadratic', 'cubic', 'barycentric', 'krogh', 'polynomial', 'spline'
# 'piecewise_polynomial', 'pchip'

# If not already installed, install the zoo package
```

```r
# if(!require(zoo)) install.packages('zoo')

# Load the zoo package
# library(zoo)

# Choose an interpolation method
# interpolation_method <- "linear"  # "linear" or "spline"

# Apply interpolation imputation
# data_imputed <- data
# if(interpolation_method == "linear") {
#     data_imputed <- na.approx(data_imputed, na.rm = FALSE)
# } else if(interpolation_method == "spline") {
#     data_imputed <- na.spline(data_imputed, na.rm = FALSE)
# } else {
#     stop("Invalid interpolation method")
# }
```

0.6.5 Prediction Imputation (KNN Imputation)

An approach to missing data imputation is to use a model to predict the missing values.

Evaluating k-hyperparmeter in KNN Imputation Select numbers of neighbors to evaluate

[42]:
```r
# Numbers of neighbors to evaluate
k_values <- c(3, 5, 7, 9, 11)
```

Operation

[ ]:
```r

```

Applying KNN Imputation

[43]:
```r
# If not already installed, install the VIM package
# if(!require(VIM)) install.packages('VIM')

# Load the VIM package
# library(VIM)

# Cross-validation
# cv_errors <- sapply(k_values, function(k) {
#     # Apply KNN imputation
#     data_imputed <- kNN(data, k = k)

#     # Calculate and return the mean squared error
#     mean((data - data_imputed)^2, na.rm = TRUE)
# })
```

```
# # Print the cross-validation errors
# print(cv_errors)
```

Select numbers of neighbors to evaluate

[44]:
```
# Number of neighbors
# optimal_k <- k_values[which.min(cv_errors)]
```

Operation

[45]:
```
# data_imputed <- kNN(data, k = optimal_k)
```

[ ]:
```

```

[46]:
```
# Generating de new Data dataframe
```

0.6.6 Iterative Imputation

Evaluating Different Imputation Order We can experiment with different imputation order strategies, such as descending, right-to-left (Arabic), left-to-right (Roman), and random.

[47]:
```
# compare iterative imputation strategies for the horse colic dataset
# If not already installed, install the mice package
# if(!require(mice)) install.packages('mice')

# Load the mice package
# library(mice)

# Determine the percentage of missing values in each column
# missing_values <- sapply(data, function(x) sum(is.na(x))/length(x))

# # Order the variables based on the percentage of missing values
# ascending_order <- order(missing_values)
# descending_order <- order(missing_values, decreasing = TRUE)
# random_order <- sample(length(missing_values))

# # Print the imputation orders
# print(ascending_order)
# print(descending_order)
# print(random_order)
```

Applying Iterative Imputation Select strategie

[48]:
```
# Selecting strategie
# strategies = ['ascending', 'descending', 'roman', 'arabic', 'random']
# imputation_order <- ascending_order  # or descending_order, or random_order
```

Operation

[49]:
```
# Perform iterative imputation
# mice_output <- mice(data[, imputation_order], m = 5, maxit = 50, method =␣
 ↪'pmm', seed = 500)

# Get the imputed data
# data_imputed <- complete(mice_output, 1)
```

[ ]:
```

```

[50]:
```
# Generating the new Data dataframe
```

0.7 Data Save

- Solo si se han hecho cambios
- No aplica

Identificamos los datos a guardar

[51]:
```
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

0.7.1 Proceso 09.2

[52]:
```
caso <- "CU_18"
proceso <- '_09.2'
tarea <- "_20"
archivo <- ""
proper <- "_diario_infra"
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

[53]:
```
# 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/.

[54]:
```
file_save <- paste0(caso, proceso, tarea, archivo, proper, extension)
path_out <- paste0(oPath, file_save)
write_csv(data_to_save, path_out)

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

```
File saved as:
```

'Data/Output/CU_18_09.2_20_diario_infra.csv'

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

[55]:
```
path_in <- paste0(iPath, file_save)
file.copy(path_out, path_in, overwrite = TRUE)
```

TRUE

0.8 REPORT

A continuación se realizará un informe de las acciones realizadas

0.9 Main Actions Carried Out

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

0.10 Main Conclusions

- Los datos están limpios para el despliegue

0.11 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

```
[ ]:
```