09.1.- Data Cleansing-Basic_CU_53_02_spi_v_01

June 13, 2023

#

CU53_impacto de las políticas de inversión en sanidad, infraestructuras y promoción turística en el SPI

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_53_08_02_spi
- Output File: CU_53_09.1_02_spi

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_53_08_02_spi.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_53_08_02_spi.csv
```

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

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

```
Rows: 2362 Columns: 18
  Column specification


Delimiter: ","
dbl (17): rank_score_spi, score_spi, score_bhn, score_fow, score_opp,
score_…
lgl  (1): is_train

 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: 2,362
Columns: 18
$ rank_score_spi <dbl> NA, 80, 97, 46, 84, 99, 150, 74, 105,
36, 143, 154, 69,…
$ score_spi      <dbl> NA, 67.59, 60.10, 73.96, 62.86, 61.43,
45.57, 66.56, 59…
$ score_bhn      <dbl> NA, 79.16, 74.55, 81.88, 79.45, 77.84,
47.15, 80.41, 66…
$ score_fow      <dbl> NA, 65.40, 51.25, 70.69, 61.22, 57.63,
45.21, 62.82, 54…
$ score_opp      <dbl> NA, 58.22, 54.49, 69.32, 47.92, 48.83,
44.34, 56.46, 57…
$ score_nbmc     <dbl> 75.01, 86.67, 72.88, 86.33, 83.91,
87.72, 54.66, 92.38,…
$ score_ws       <dbl> 83.09, 86.44, 83.35, 88.07, 77.71,
78.15, 47.82, 78.47,…
$ score_sh       <dbl> 73.63, 87.69, 77.17, 89.59, 85.11,
86.61, 36.59, 85.21,…
$ score_ps       <dbl> NA, 55.85, 64.81, 63.55, 71.08, 58.87,
49.53, 65.57, 50…
$ score_abk      <dbl> 84.43, 74.20, 47.04, 89.07, 65.15,
55.79, 50.36, 81.61,…
$ score_aic      <dbl> 28.35, 74.19, 37.15, 68.14, 51.25,
78.17, 33.84, 61.95,…
$ score_hw       <dbl> NA, 53.55, 64.58, 61.41, 62.00, 45.35,
36.99, 61.64, 41…
$ score_eq       <dbl> 62.42, 59.66, 56.22, 64.13, 66.47,
51.22, 59.66, 46.07,…
$ score_pr       <dbl> NA, 81.60, 71.05, 90.28, 61.56, 60.41,
69.20, 70.02, 74…
$ score_pfc      <dbl> NA, 60.29, 64.77, 67.65, 56.51, 58.62,
40.61, 62.49, 59…
$ score_incl     <dbl> NA, 40.24, 56.12, 68.48, 48.70, 35.57,
41.81, 36.89, 55…
$ score_aae      <dbl> NA, 50.73, 26.03, 50.87, 24.90, 40.72,
25.72, 56.45, 40…
$ is_train       <lgl> TRUE, TRUE, TRUE, TRUE, TRUE, TRUE,
TRUE, TRUE, TRUE, T…
```

Muestra de los primeros datos:

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

A spec_tbl_df: 5 × 18

rank_score_spi <dbl>	score_spi <dbl>	score_bhn <dbl>	score_fow <dbl>	score_opp <dbl>	score_nbmc <dbl>	score_ <dbl>
NA	NA	NA	NA	NA	75.01	83.09
80	67.59	79.16	65.40	58.22	86.67	86.44
97	60.10	74.55	51.25	54.49	72.88	83.35
46	73.96	81.88	70.69	69.32	86.33	88.07
84	62.86	79.45	61.22	47.92	83.91	77.71

0.6 Text data analysis

Select columns

[9]:
```
# Select column
```

Operation

[10]:
```
# Analizar datos de texto y verificar su corrección
# e.g. faltas ortografía, etc
```

[11]:
```
# pasar a mayúsculas todas las columnas de texto
```

0.7 Delete Columns Needless/Irrelevant/Private

Select columns

[12]:
```
# Select columns
```

Operation

[13]:
```
# Eliminamos columnas que consideramos irrelevantes o innecesarias
```

[14]:
```
# Eliminamos columnas que consideramos irrelevantes o innecesarias
```

0.8 Inconsistent Data

[ ]:
```

```

Select columns and value

[15]:
```
# Select column and value
# e.g. age > 100
```

Operation

[16]:
```
# Inconsistent data is unique to each data set and
# must be searched manually
```

0.9 Expected values

[17]:
```r
# Check for expected value
```

0.10 Zeros

[18]:
```r
# Check for zeroes in data
zero_counts <- colSums(data == 0, na.rm = TRUE)

# Print variables with zero counts
for (variable in names(zero_counts[zero_counts > 0])) {
  num_zeroes <- zero_counts[variable]
  print(paste("Variable:", variable, "- Number of zeroes:", num_zeroes))
}
```

```
[1] "Variable: is_train - Number of zeroes: 471"
```

0.11 Single Value

[19]:
```r
# Obtener el número de valores diferentes en cada columna
unique_counts <- summarise(data, across(everything(), ~length(unique(.))))

# Imprimir los resultados
print(unique_counts)
```

```
# A tibble: 1 × 18
  rank_score_spi score_spi score_bhn score_fow score_opp score_nbmc score_ws
           <int>     <int>
<int>     <int>
<int>      <int>
<int>
1              170      1709      1624
1718      1739       1615     1786
#   11 more variables: score_sh <int>, score_ps <int>, score_abk
```

```
<int>,
#   score_aic <int>, score_hw <int>, score_eq <int>, score_pr <int>,
#   score_pfc <int>, score_incl <int>, score_aae <int>, is_train <int>
```

[20]:
```r
# Columns with a single unique value
#
# NOTE: WE MUST EXCLUDE TARGET COLUMN TO DELETE
```

0.12 Very Few Values

Select rate

[21]:
```
# Select rate
rate <- 0.1
```

Operation

[22]:
```
# Show features with over rate rows being the same value
features_with_same_values <- names(data)[apply(data, 2, function(x)␣
 ↪length(unique(x)) <= (1 - rate) * nrow(data))]
print(features_with_same_values)
```

```
 [1] "rank_score_spi" "score_spi"      "score_bhn"      "score_fow"
 [5] "score_opp"      "score_nbmc"     "score_ws"       "score_sh"
 [9] "score_ps"       "score_abk"      "score_aic"      "score_hw"
[13] "score_eq"       "score_pr"       "score_pfc"      "score_incl"
[17] "score_aae"      "is_train"
```

[23]:
```
# 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.

# Summarize the number of unique values in each column
unique_counts <- sapply(data, function(x) length(unique(x)))

# Calculate the percentage of unique values for each variable
percentage_unique <- unique_counts / nrow(data) * 100

# Create a data frame with the results
summary_data <- data.frame(Variable = names(unique_counts), Unique_Count =␣
 ↪unique_counts, Percentage_Unique = percentage_unique)

# Print the summary data
print(summary_data)
```

```
                    Variable Unique_Count Percentage_Unique
rank_score_spi rank_score_spi          170        7.19729043
score_spi           score_spi         1709       72.35393734
score_bhn           score_bhn         1624       68.75529213
score_fow           score_fow         1718       72.73497036
score_opp           score_opp         1739       73.62404742
score_nbmc         score_nbmc         1615       68.37425910
score_ws             score_ws         1786       75.61388654
score_sh             score_sh         1647       69.72904318
score_ps             score_ps         1643       69.55969517
score_abk           score_abk         1832       77.56138865
score_aic           score_aic         2016       85.35139712
score_hw             score_hw         1733       73.37002540
score_eq             score_eq         1794       75.95258256
```

```
score_pr          score_pr        1434       60.71126164
score_pfc         score_pfc       1728       73.15834039
score_incl        score_incl      1754       74.25910246
score_aae         score_aae       1758       74.42845047
is_train          is_train           2        0.08467401
```

Select percent of the number of rows

[24]:
```
# Select percent of the number of rows
percentage_threshold <- 0.05
```

Operation

[25]:
```
# Summarize columns that have unique values that are less than
# "porcentage_threshold" percent of the number of rows.
columns_to_delete <- names(data)[sapply(data, function(x) length(unique(x)) <␣
 ↪percentage_threshold * nrow(data))]
columns_to_delete <- setdiff(columns_to_delete, "target_column")  # Excluir la␣
 ↪columna objetivo
columns_to_delete
```

'is_train'

Select percent of the number of rows

[26]:
```
# Select percent of the number of rows
# print(colnames(data))
```

Operation

[27]:
```
# Delete columns that have unique values that are less than
# "porcentage_threshold" percent of the number of rows.
#
# NOTE: WE MUST EXCLUDE TARGET COLUMN TO DELETE

# data <- data[, -which(names(data) %in% columns_to_delete)]
```

[ ]:
```

```

0.13 Low Variance

A) Calculating variances

[28]:
```
# Calculate variance for all numeric variables
variance <- summarise(data, across(where(is.numeric), var))

# Print the variance
print(variance)
```

```
# A tibble: 1 × 17
  rank_score_spi score_spi score_bhn score_fow score_opp score_nbmc score_ws
           <dbl>     <dbl>     <dbl>     <dbl>     <dbl>      <dbl>    <dbl>
1             NA        NA        NA        NA        NA         NA       NA
# ℹ 10 more variables: score_sh <dbl>, score_ps <dbl>, score_abk
#   <dbl>, score_aic <dbl>, score_hw <dbl>, score_eq <dbl>, score_pr <dbl>,
#   score_pfc <dbl>, score_incl <dbl>, score_aae <dbl>
```

B) Automatic calculation and representation of variances

Define thresholds to check

[29]:
```
# define thresholds to check
threshold <- 0.05 # Example thresholds, adjust as needed
```

Operation

[30]:
```
data_without_date <- data[, !(names(data) %in% "fecha")]
variances <- apply(data_without_date, 2, function(x) var(x, na.rm = TRUE))
```

[31]:
```
variances
```

rank__score__spi 2381.17414898865 **score__spi** 239.902023437066 **score__bhn** 264.644125049509 **score__fow** 246.321588377973 **score__opp** 299.617035517918 **score__nbmc** 215.062075571483 **score__ws** 370.915934826313 **score__sh** 453.349212106243 **score__ps** 140.050038193952 **score__abk** 390.802908016272 **score__aic** 489.854111075393 **score__hw** 291.877915511645 **score__eq** 152.813230219569 **score__pr** 511.574711008349 **score__pfc** 219.653801921002 **score__incl** 397.081955740845 **score__aae** 365.315337694858 **is__train** 0.159711634982952

C) Delete variables with low variance

Select column

[32]:
```
# Select column
low_variance_columns <- names(variances)[variances < threshold]
```

[33]:
```
low_variance_columns
```

Operation

[34]:
```
# drop columns with low variance
data <- data[, !(names(data) %in% low_variance_columns)]
```

0.14 Duplicates

Entendido como ERROR –> Eliminar duplicados

0.14.1 Identificación de Duplicates

[35]:
```
# Buscamos filas duplicadas
duplicated_rows <- data[duplicated(data),]
```

0.14.2 Eliminación de primera fila duplicada

[36]:
```
# Eliminamos filas duplicadas
# Deja la primera y elimina el resto duplicadas
data_no_duplicates <- data[!duplicated(data),]
```

0.14.3 Eliminación de todas las filas duplicadas

[37]:
```
# Eliminamos filas duplicadas
# Elimina todas las filas que están duplicadas (no deja una)

# data_no_duplicates_all <- data %>% distinct()
```

0.14.4 Eliminación de filas duplicadas en columnas concretas

Select columns

[38]:
```
# Select columns
# columns <- c("columna1", "columna2")
```

Operation

[39]:
```
# Eliminamos filas duplicadas
# Elimina las filas que están duplicadas en columnas seleccionadas
# data_no_duplicates_columns <- data %>% distinct(across(all_of(columns)))
```

0.15 Data Save

- Solo si se han hecho cambios
- No aplica

Identificamos los datos a guardar

[40]:
```
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

[41]:
```
caso <- "CU_53"
proceso <- '_09.1'
tarea <- "_02"
archivo <- ""
proper <- "_spi"
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

[42]:
```
# 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/.

[43]:
```
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_53_09.1_02_spi.csv’

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

[44]:
```
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

[ ]: