Universidad Internacional de La Rioja

Facultad de Ingeniería y Tecnología

Máster Universitario en Análisis y Visualización de Datos Masivos / Visual Analitics & Big Data

Paso a paso Limpieza de un Dataset

Actividad de estudio presentado por:	Juan David Escobar Escobar
Tipo de trabajo:	Actividad
Modalidad:	Individual
Director/a:	Marlon Cardenas Bonett
Fecha:	Diciembre 2021

1.1. Interpretación de datos crudos

Esta es la primera etapa del proceso, para llevarla a cabo se deben tener presente los siguientes pasos:

1.1.1. Codificación del archivo fuente.

Para el ejercicio vamos a trabajar con el archivo codificado en UTF-8 y no en ASCCI (American Standard Code for International Interchange), ya que solo cuenta con 8 bits (128 caracteres) mientras que Unicode UTF (Unicode Transformation Format) cuenta con 135.000 caracteres lo cual nos da la tranquilidad de incluir caracteres especiales del idioma español, tales como los caracteres tildes, ñ, entre otros (https://www.youtube.com/watch?v=M_yNoV3c8DY).

1.1.2. Análisis o exploración del Dataset.

En este paso se debe identificar el tipo de fuente de donde provienen los datos (Base de datos o texto plano), formatos (TXT, PARQUET, CSV, JSON, ABRO, ORC), codificación (UTF-8, Windows, entre otros), si posee cabeceras, carácter separador de línea, carácter salto de línea, estructura, tipos de datos, entre otros. Para nuestro ejercicio contamos con un archivo en formato CSV, el cual tiene una estructura que se compone de la primera línea compuesta por los nombres o cabeceras de la información, el separador es el carácter ",", el salto de línea esta dado por el carácter "CRLF".

```
"""
Descripción: Retorna boolean que determina si el archivo cuenta con el
encoding UTF-8.
Responsables: Juan David Escobar E
Fecha: 30/11/2021
"""

def is_valid_encoding_csv(ar_file):
    this_encoding = 'UTF-8'
    result = chardet.detect(open(ar_file, 'rb').read())
    charenc = result['encoding']
    return  True if this_encoding in charenc.upper() else False
```

1.1.3. Lectura del Dataset con PySpark Dataframes.

La lectura de datos con Python se puede realizar diversas maneras, en este caso

estamos utilizando la función spark.read() de pySpark, es importante parametrizar la función con los parámetros descritos en el paso que disponemos del archivo fuente a leer, ya que son muy importantes para interpretar el archivo de manera correcta de acuerdo a su estructura, características o propiedades, para realizar esta operación se encierra dicha instrucción en un bloque de código TRY CATCH, para identificar posibles errores asociados a la estructuración y formato correcto del archivo, dichos errores pueden ser almacenados en una base datos de auditoria para llevar a cabo un seguimiento de los archivos leídos con éxito o error, para complementar este proceso se puede agregar un folder llamado error donde se muevan los archivos con problemas, y retornados a la carpeta llamada crudos (raw), cuando sean corregidos desde la fuente dueña de la información.

```
def read_csv():

  # File location (https://www.youtube.com/watch?v=-tZbkgTnGs4)
  file_location = '/Users/juandavidescobarescobar/Documents/Unir/Materias/BD Big Data/Actividad 1/data_act_03.csv'
  file_type = 'csv'

  # CSV options
  infer_schema = 'true'
  first_row_is_header = 'true'
  delimiter = ';'

  # Validate encoding UTF-8
  is_valid_encode = is_valid_encoding_csv(file_location)

  if is_valid_encode:

    try:
      # The applied options are for CSV files. For other types, these will ignored.
      df = spark.read.format(file_type) \
                .option('inferSchema', infer_schema) \
                .option('header', first_row_is_header) \
                .option('sep', delimiter) \
                .load(file_location)
    except Exception as error:

      print('Error leyendo el archivo: ' + str(error))

  return df
```

1.1.4. Análisis descriptivo de las variables

Una vez almacenado el Dataset “Registros crímenes” en un objeto en memoria por medio de Python en nuestro ambiente de desarrollo, vamos a proceder con una fase exploratoria de los datos a través de los metadatos que proporciona un objeto Dataframe, funciones estadísticas, tipos de datos y algunos gráficos, lo anterior nos permitirá identificar la distribución y la calidad de la información que se almacena en dicho Dataset.

La función df.limit(4).toPandas().head(), muestra un top 4 de los registros del contenido de la tabla de datos que cargamos en el Dataframe.

CrimeId	OriginalCrimeType	OffenseDate	CallTime	CallDateTime	Dispositio	Address	City	State	Agen	Range	AddressType
160903280	Assault / Battery	30/03/16	18:42	30/03/16 18:42	REP	100 Block Of Chilton Av	San Francisco	CA	1	None	Premise Address
160912272	Homeless Complaint	31/03/16	15:31	31/03/16 15:31	GOA	2300 Block Of Market St	San Francisco	CA	1	None	Premise Address
160912590	Susp Info	31/03/16	16:49	31/03/16 16:49	GOA	2300 Block Of Market St	San Francisco	CA	1	None	Premise Address
160912801	Report	31/03/16	17:38	31/03/16 17:38	GOA	500 Block Of 7th St	San Francisco	CA	1	None	Premise Address

La función df_pd.info(), nos muestra la estructura interna de la tabla, indicando los tipos de datos, columnas, si aceptan valores nulos, la cantidad de valores nulos y no nulos que contienen, cantidad de registros almacenados y la memoria que ocupa dicha información almacenada.

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10051 entries, 0 to 10050
Data columns (total 12 columns):
 #   Column                 Non-Null Count  Dtype
---  ------                 --------------  -----
 0   CrimeId                10051 non-null  int32
 1   OriginalCrimeTypeName  10051 non-null  object
 2   OffenseDate            10051 non-null  datetime64[ns]
 3   CallTime               10051 non-null  object
 4   CallDateTime           10051 non-null  datetime64[ns]
 5   Disposition            10051 non-null  object
 6   Address                10051 non-null  object
 7   City                   9730 non-null   object
 8   State                  10048 non-null  object
 9   AgencyId               10051 non-null  object
 10  Range                  0 non-null      object
 11  AddressType            10051 non-null  object
dtypes: datetime64[ns](2), int32(1), object(9)
memory usage: 903.1+ KB
```

```
root
 |-- CrimeId: integer (nullable = true)
 |-- OriginalCrimeTypeName: string (nullable = true)
 |-- OffenseDate: timestamp (nullable = true)
 |-- CallTime: string (nullable = true)
 |-- CallDateTime: timestamp (nullable = true)
 |-- Disposition: string (nullable = true)
 |-- Address: string (nullable = true)
 |-- City: string (nullable = true)
 |-- State: string (nullable = true)
 |-- AgencyId: string (nullable = true)
 |-- Range: string (nullable = true)
 |-- AddressType: string (nullable = true)
```

Los datos descritos en el resultado no son muy acordes o no están asociados a los valores y las variables observadas en la tabla obtenida a partir de la consulta de los datos crudos del Dataset mediante la opción de esquema inferido de manera automática, Python no logra inferir totalmente todos los tipos de datos de manera correcta, y opta por asignarles un tipo de dato genérico, el cual es String u Object, lo cual debemos proceder a corregir, a medida que vamos limpiando y corrigiendo cada una de las inconsistencias de nuestro Dataset, un ejemplo de ello es tener valores numéricos y valores alfanuméricos asociados a

una misma variable.

La función df_pd.describe(), nos muestra las estadísticas (count, unic, top, freq, first, last, mean, std, min, 25%, 50%, 75%, max) para cada una de las variabels de nuestro Dataset, como se menciono en la descripción del comando anterior, esta descripción aún no es muy precisa debido a que tenemos asignados tipos de datos a nuestras variables que aún no se acoplan perfectamente, debido a que existen ciertas inconsistencias en los datos.

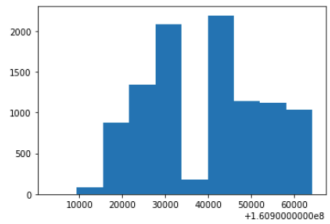
	CrimeId	OriginalCrimeTyp	OffenseDate	CallTime	CallDateTime	Disposition	Address	City	State	AgencyId	Range	AddressType
count	10051000000	10051	10051	10051	10051	10051	10051	9730	10048	10051	0	10051
unique	NaN	575	9	1416	5116	19	5387	8	1	2	0	6
top	NaN	Traffic Stop	2/04/16 0:00	17:39	4/04/16 12:23	HAN	900 Block Of M	San Francisc	CA	1	NaN	Premise Address
freq	NaN	1215	2259	19	8	2820	58	9665	10048	10048	NaN	5059
first	NaN	NaN	4/04/13 0:00	NaN	30/03/16 18:42	NaN	NaN	NaN	NaN	NaN	NaN	NaN
last	NaN	NaN	4/04/25 0:00	NaN	5/04/16 23:54	NaN	NaN	NaN	NaN	NaN	NaN	NaN
mean	160939400000000	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
std	13270060000	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
min	160903300000000	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
25%	160930300000000	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
50%	160940800000000	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
75%	160951300000000	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
max	160964200000000	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN

De la anterior tabla podemos deducir lo siguiente:

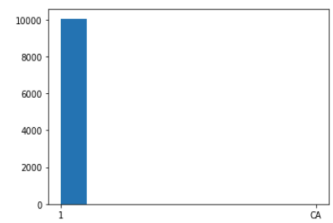
- **Numero total de registros:** 10051.
- **Numero total de variables**: 12, 1 numérica y 9 cadena y 2 fecha.
- **Rango temporal "OriginCrimeTypeName":** Desde 2013-04-04 hasta 2025-04-04.
- **Rango temporal "CallDateTime":** Desde 2016-03-30 hasta 2016-04-05.

Utilizaremos el diagrama de histograma para analizar mas detalladamente el comportamiento de las variables numéricas de nuestro Dataset:

```
''' Generamos los histogramas de dos variables numéricas que presenta
    la tabla de datos: CrimeId y a AgencyId
'''

data_crime_id = list(df_pd["CrimeId"])

plt.hist(data_crime_id, 10)
plt.show()
```




```
''' Generamos los histogramas de dos variables numéricas que presenta
    la tabla de datos: CrimeId y a AgencyId
'''

data_agency_id = list(df_pd["AgencyId"])

plt.hist(data_agency_id, 10)
plt.show()
```



Analizando las dos graficas anteriores podemos concluir que no tenemos valores cercanos a cero por lo cual no tenemos un sesgo en los datos. Hemos finalizado el primer análisis descriptivo, el cual es recomendado ejecutar cada vez que

modifiquemos la tabla, para verificar de esta manera el efecto que producen sobre los cambios aplicados en cada nueva fase.

1.1.5. Ajuste de tipos de variables

En este paso procedemos a validar cada variable con el tipo de valor correspondiente, es decir las variables numéricas y cualitativas, los tipos tipos de datos identificados en el Dataset son los siguientes:

Variable	Tipo	Nullable	Formato	Muestra	Registros	Nulos	Val. Unicos	Val. Duplicados
CrimeId	integer	FALSE		160903280	10051	0	10047	4
OriginalCrimeTypeName	string	TRUE		Assault	10051	0	575	9476
OffenseDate	timestamp	TRUE	yyyy-mm-dd	30/03/16	10051	0	9	10042
CallTime	string	TRUE	HH:mm	18:42	10051	0	1416	8635
CallDateTime	timestamp	TRUE	yyyy-MM-dd'T'HH:mm:ss	30/03/16 18:42	10051	0	5116	4935
Disposition	string	TRUE	[A-Z]{3}	REP	10051	0	19	10032
Address	string	TRUE		100 Block Of	10051	0	5387	4664
City	string	TRUE		San Francisco	9730	321	8	9722
State	string	TRUE	[A-Z]{2}	CA	10048	3	1	10047
AgencyId	integer	TRUE		1	10051	0	2	10049
Range	string	TRUE		None	0	10051	0	0
AddressType	string	TRUE		Premise Address	10051	0	6	10045

Se procede con la ejecución del comando df_pd = df_pd.drop_duplicates() para proceder con la eliminación de registros duplicados, es decir registros o filas que contengan exactamente los mismos valores, para evitar inconsistencias, para nuestro ejercicio no se encontraron valores duplicados.

A continuación, procedemos con la validación de registros duplicados por el valor que hace referencia a la llave primaria de cada registro "**CrimeId**", la cual por regla de normalización de tablas en bases de datos no debe contener valores duplicados, para realizar dicha operación se construyo una función personalizada la cual arroja como resultado una lista de los identificadores que contienen valores duplicados, el resultado que nos arrojo la función list_duplicates = get_duplicates(df, df_pk), es el siguiente:

```
1
{'is_error': True, 'msg_error': '[CrimeId: 160950496 - Cantidad: 3],
[CrimeId: 160913455 - Cantidad: 3]'}
```

Lo cual nos indica que tenemos 2 valores que se repiten 3 veces cada uno, lo mas lógico en este caso es proceder con la eliminación de dos registros de cada identificador, pero se nos presenta el siguiente problema, el cual podemos observar en la tabla a continuación la cual es el resultado obtenido a partir de la

consulta de uno de los identificadores:

	CrimeId	OriginalCrimeTypeName	OffenseDate	CallTime	CallDateTime	Disposition	Address
0	160950496	Passing Call	2016-04-04	6:51	2016-04-04 06:51:00	HAN	University St/felton St
1	160950496	Suspicious Vehicle	2016-04-04	6:51	2016-04-04 06:51:00	ND	1400 Block Of Cabrillo St
2	160950496	Trespasser	2016-04-04	6:51	2016-04-04 06:51:00	CAN	Block Of Hampshire St

Como podemos ver cada valor del mismo identificador **160950496** tiene valores diferentes para cada fila o registro, lo cual nos impide eliminar estos subconjuntos de datos por medio de la función df.drop_duplicates(subset = [df_pk]), debido a que desconocemos cual es el registro correcto, cuando esta situación sucede, la manera de solucionar el problema es yendo a la fuente de datos y validar con el dueño de la información proveída al equipo de datos cual es la información correcta acompañado de las correctivas validaciones de lo que pudo haber sucedido al momento de generar el Dataset original **"archivo fuente CSV".** El problema se presenta de igual manera para ambos identificadores **160950496 y 160913455,** para el ejercicio actual estos registros se dejarán tal cual, ya que no somos los dueños de la información y no tenemos conocimiento de cual es el correcto o si se presento un error a la hora de generar este conjunto de datos. Solo queda agregar que para la variable "**CrimeId"** el tipo de dato inferido como **Integer** es correcto y no posee valores nulos, por lo cual se dejara tal cual.

El siguiente paso será la validación de formatos, nos centraremos en las variables OffenseData, CallTime, CallDateTime, Disposition y State, los cuales presentan un formato o patrón de caracteres estándar, tal como se detalla en la siguiente tabla:

Variable	Tipo	Formato	Muestra
OffenseDate	timestamp	yyyy-mm-dd	30/03/16
CallTime	string	HH:mm	18:42
CallDateTime	timestamp	yyyy-MM-dd'T'HH:mm:ss	30/03/16 18:42
Disposition	string	[A-Z]{3}	REP
State	string	[A-Z]{2}	CA

Para las variables tipo fecha o estampa de tiempo “OffenseDate, CallTime, CallDateTime” se construyo una función que permite validar el formato de cada una de las variables de manera paramétrica, casteando los valores a un tipo de dato timestamp y catalogándolos entre correctos, incorrectos e ignorando los valores nulos, para nuestro Dataset se encontraron los formatos y valores correctos. Se noto redundancia de información, ya que la columna CallDateTime ya contiene toda la información que almacenan OffenseDate y CallTime, para este caso es recomendado no almacenar información redundante y en caso de ser requerida de manera segmentada en la fuente de datos de destino, se pueden manejar a través de columnas derivadas o calculadas, el siguiente es el resultado obtenido de la validación de formatos de tiempo y fecha.

```
1 list_bad_rec_offense_date:  []
2 list_bad_rec_call_time:  []
3 list_bad_rec_call_date_time:  []
```

Para las variables Disposition y State, se procederá a validar el formato para datos tipo texto que se observa como factor común mediante la evaluación de expresiones regulares, para la variable “State”, todos los valores cumplen con el patrón especificado como expresión regular, la variable “Disposition” tiene cuatro valores que no cumplen con la expresión regular, exactamente no se sabe el significado de los valores encontrados, pero se puede deducir que ND: es Not Disposition y Not recorderded es un valor Dummy que remplaza un nulo, por lo cual no se modificarán y se dejarán tal cual, el siguiente es el resultado obtenido:

	CrimeId	Disposition	Disposition_reg
0	160913078	ND	False
1	160913455	ND	False
2	160913619	ND	False
3	160913643	ND	False
4	160913872	Not recorded	False

Para continuar, se procede a eliminar los espacios en blanco a la derecha y a la

izquierda para las variables tipo texto (OriginalCrimeTypeName, Disposition, Address, City, State y Range), para tener la información almacenada de una manera mas limpia y concisa:

```
# Eliminación de espacios en blancos para las variables tipo texto
# [OriginalCrimeTypeName, Disposition, Address, City, State y Range]

list_cols_str = [item[0] for item in df.dtypes if
item[1].startswith('string')]

for col_name in list_cols_str:
    df = df.withColumn(col_name, trim(f.col(col_name)))

df.limit(20).toPandas().head()
```

1.1.6. Detección y tratamiento de datos ausentes

En esta fase validaremos la presencia de datos perdidos, celdas vacías o valores nulos, mediante la función count_missings(df), identificamos los valores perdidos por cada columna, las columnas que presentan valores perdidos son 3 Range, City y State, la variable Range es eliminada ya que presenta un valor mayor al 50% de los datos y no aporta información relevante.

	count
Range	10051
City	321
State	3

Continuando con el tratamiento de datos ausentes para el resto de las columnas City y State, son variables tipo texto, por lo cual no se rellenarán aquellos espacios vacíos con el valor Dummy seleccionado “-99”.

1.1.7. Identificación de datos atípicos

Aquellos valores que no concuerdan con la descripción o el tipo de dato de una variable pueden afectar la calidad de los datos, tareas de análisis y pueden considerarse como datos anómalos. Lo que haremos será detectarlos para disminuir el daño en fases posteriores, para llevar a cabo la detección se identifican y se agrupan las variables según su tipología numéricas y categóricas, por medio de la función distinct(), la creación de histogramas mediante las

funciones draw_hist_num_vars(list_var_num) y draw_hist_cat_vars(list_var_cat), dichas funciones nos permiten tener una visión mas clara de los valores almacenados en cada una de las variables, a continuación se detalla una descripción de los valores atípicos encontrados para cada variable

Variable	Tipo	Tipología	Observación valores atipicos
OriginalCrimeTypeName	string	Categorica - Nominal	La información es alfa númerica, se identifican valores numericos, decimales, direcciones Ip y algunos caracteres sin sentido
Disposition	string	Categorica - Nominal	La información cumple un patron de 3 letras mayusculas, se identifica un valor numerico "22" atipico y un valor Dummy NotRecorded
Address	string	Categorica - Nominal	NA
City	string	Categorica - Nominal	Se encuentra un valor atipico "S" y dos vacios, los cuales serán remplazado por el valor mas cercano "SanFrancisco", la filas 5773, 8023 y 8475 tiene un error para las columnas "City", "AgencyId" y "AddressType", ya que en el formato original los valores se movieron una posicion hacia la derecha
State	string	Categorica - Nominal	Tiene 3 valores Null, los cuales srán remplazados con el valor "CA", ya que se identifico un error en el cual este valor se movio una posición a la derecha en el archivo original fuente CSV.
AddressType	string	Categorica - Nominal	Se encuentran 3 registros con valores atipicos igual a "1", estos serán remplazados por vacio, ya que el CSV no almaceno la información correctamente y estos valores igual a "1" pertenecen a la columna "AgencyId"
OffenseDate	timestamp	Numerica - Continuo	NA, Columna redundate se elimina
CallTime	string	Numerica - Continuo	NA, Columna redundate se elimina
CallDateTime	timestamp	Numerica - Continuo	NA
CrimeId	integer	Numerica - Discreta	NA
AgencyId	integer	Numerica - Discreta	Factor comun el valor numerico "1", hay 3 valores atipico tipo texto "CA" error formato del archivo CSV (Lineas 5773, 8023 y 8475). Estos valores serán remplazados por el valor 1, ya que se movio a la derecha y se encuentran en la columna "AddressType".
Range	string	Sin valores	NA, no posee información relevante y tiene una muestra nula mayor al 50%, se opta por eliminar del dataset

Los dos datos variables numéricas del Dataset se analizaron mediante gráficos del tipo Histograma y Bigotes, los valores atípicos solo se hallaron en la variable AgencyId la solución para dichos valores se encuentra descrita en la tabla anterior.

Para las variables nombradas como categóricas también se realizo un análisis mediante grafico de barras, las cuales nos ofrecieron una mejor visualización de los valores atípicos encontrados en dichas variables (Disposition, City, State y AddressType), los valores atípicos para las variables numéricas y categóricas son:

Para concluir nuestra fase de ajustes a los tipos de variable, se creará una copia del Dataframe original, el cual almaceno los datos con un esquema inferido de forma automática, al nuevo Dataframe le especificaremos un nuevo esquema ajustado a la naturaleza de los datos y variables asociadas, ya que contamos con la tranquilidad de que validamos los tipos de datos y formatos de estos en pasos previos, en este paso se ignoraran las variables redundantes.