

RM CLEANING Pt.ipynb

Limpieza y análisis exploratorio de la base de datos de Registro Mercantil de la Cámara de Comercio de Cali

1. Se cargan los paquetes o librerías **pandas** y **numpy**.

Pandas es una biblioteca de software escrita como extensión de Numpy para manipulación y análisis de datos para lenguaje de programación Python. En particular ofrece estructuras de datos y operaciones para manipular tablas numéricas y series temporales. El nombre deriva del término “datos de panel”, término de econometría que designa datos que combinan una dimensión temporal con otra dimensión transversal.

```
[1] ▶ ▶≡ MI
#Cargamos los paquetes que vamos a necesitar
import pandas as pd
import numpy as np
#from pandas_profiling import ProfileReport

# Visualisation
#import matplotlib.pyplot as plt
#import seaborn as sns
#import plotly.express as px
#import plotly.graph_objects as go
```

2. En esta línea se crea un Data Frame en el que se va a cargar una BD que está en formato Excel. Además, se le va a indicar que las variables CODIGO CIIU, MATRICULA, NIT, FECHA_RENOVACION y TELEFONO son objetos. Esto debido a que son variables que por su naturaleza no se pueden tomar como un dato numérico.

```
df = pd.read_excel('RM_DIC2020_V1(ENE2021_EEE3).xlsx',
dtype={'CODIGO_CIIU':object, 'MATRICULA': object,
'NIT':object, 'Año de FECHA_RENOVACION':object, 'TELEFONO':
object})
```

En esta línea, buscamos configurar cuáles son las características de las variables y asegurarnos que las anteriores hayan quedado como objetos. Con la configuración buscamos que se muestren todas las columnas de la BD, es decir, le indicamos None para no tener un máximo de variables a mostrar.

```
pd.set_option('display.max_columns', None)
```

En estas líneas con `df.info` imprimimos información acerca del Data Frame, incluido el tipo de índice y los tipos de columna, los valores no nulos y el uso de memoria. Con `df.head` se retorna las primeras n filas del objeto según la posición. Es útil para probar rápidamente si el objeto tiene el tipo correcto de datos.

```
pd.set_option('display.max_columns', None)
print(df.info())
df.head(2)
```

3. En esta línea se cambian los nombres de las variables originales en el data frame por unas nuevas que permitan facilitar su interpretación. El `inplace = TRUE` es un parámetro que ayuda a decidir cómo se desea afectar los datos subyacentes del objeto Pandas. ¿Desea realizar un cambio en el objeto de marco de datos en el que está trabajando y sobrescribir lo que había antes? (no. `Inplace = false`) (yes `inplace = true`).

¿o deseas hacer una copia del objeto de marco de marco de datos y asignarlo a una variable diferente para poder modificar esos datos originales más adelante? Esas son las dos preguntas que uno debe hacerse. Las respuestas ayudaran a determinar si se necesita establecer el parámetro `in situ` en verdadero o falso.

```
# Se cambian los nombres de las variables para facilitar su
interpretación
df.rename(columns={'Año de
FECHA_RENOVACION': 'ULTIMO_AÑO_RENOVADO',

'FECHA_MAT_REN': 'FECHA_MATRICULA_RENOVACION_HORA',
'TOT_ACTIVOS': 'ACTIVO_TOTAL',
'UTILILIDAD_PERDIDA':
'UTILILIDAD_OPERACIONAL',
'VENTAS' :
'INGRESOS_ACTIVIDAD_ORDINARIA',
'TOT_PASIVO': 'PASIVO_TOTAL',
'UTILILIDAD_BRUTA':
'RESULTADO_PERIODO',
'MAT_REN' : 'MATRICULA_RENOVACION'})
,
inplace=True)
```

4. En esta línea se separa la fecha para facilitar el análisis. Antes del igual ponemos las variables que vamos a crear. Después del igual ponemos la variable que vamos a separar.

El código `series.str.split(pat = Ninguno, n = -1, expandir = False)` divide la cadena en Series / Index desde el principio, en la cadena delimitadora especificada. Equivalente a **`str.split()`**.

Parámetros:

pat: str, opcional

cadena o expresión regular para dividir. Si no se especifica, divide en espacios en blanco.

n: int, predeterminado -1 (todos)

limite el número de divisiones en la salida. *None*, o *-1* se interpretarán como devolver todas las divisiones.

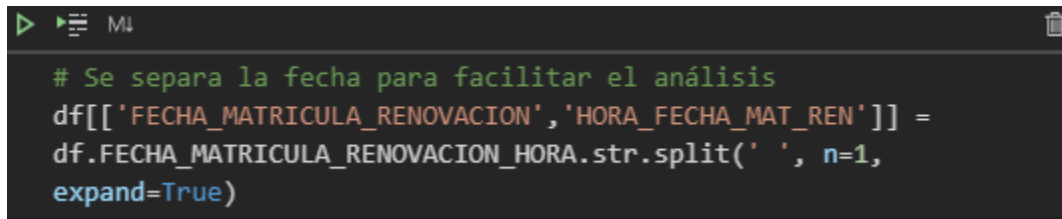
expandir: bool, predeterminado Falso

Expanda las cadenas divididas en columnas separadas

*Si True, devuelve DataFrame/MultiIndex expandiendo la dimensionalidad

*Si False, devuelve Series/Index, que contiene listas de cadenas

Devoluciones: Serie, índice, DataFrame o MultiIndex



```
# Se separa la fecha para facilitar el análisis
df[['FECHA_MATRICULA_RENOVACION', 'HORA_FECHA_MAT_REN']] =
df.FECHA_MATRICULA_RENOVACION_HORA.str.split(' ', n=1,
expand=True)
```

Para más información:
<https://pandas.pydata.org/docs/reference/api/pandas.Series.str.split.html#:~:text=When%20using%20expand%3DTrue%20%2C%20the,the%20columns%20during%20the%20split.&text=For%20slightly%20more%20complex%20use,p arameter%20settings%20can%20be%20used.>

5. Se va a dividir la variable FECHA_MATRICULA_RENOVACION en 3 columnas, utilizando el separador que tiene dicha variables que es el “/” . Estas separaciones del Data Frame las vamos a guardar en 3 nuevas variables ‘DIA’, ‘MES’ y ‘AÑO’.

```
[5] ▶ M4

# Se separa día, mes y año para facilitar el análisis
df[['DIA', 'MES', 'ANO']] =
df.FECHA_MATRICULA_RENOVACION.str.split('/', n=3,
expand=True)
#Se convierte la variable FECHA_MATRICULA_RENOVACION_HORA en
datetime para facilitar manipulación de los datos
df.FECHA_MATRICULA_RENOVACION_HORA = pd.to_datetime
(df.FECHA_MATRICULA_RENOVACION_HORA)
```

6. Se convierte la FECHA_MATRICULA en datetime para facilitar la manipulación de los datos.

```
[6] ▶ M4

#Se convierte la variable FECHA_MATRICULA en datetime para
facilitar manipulación de los datos
df.FECHA_MATRICULA = pd.to_datetime(df.FECHA_MATRICULA)
```

La explicación para la línea anterior es la siguiente:

Se puede utilizar la siguiente plantilla para convertir cadenas a fecha y hora en Pandas Data Frame. Si no se especifica el formato no genera error. También se puede utilizar df. en lugar de df []

```
df ['DataFrame Column'] = pd.to_datetime (df ['DataFrame
Column'], formato = especifique su formato)
```

7. Se debe organizar el DataFrame según la fecha del movimiento registrado en FECHA_MATRICULA_RENOVACION_HORA. Esto facilitara las particiones que se harán a la base de datos posteriormente.

Luego imprimimos la información de las variables y vemos las dos primeras filas del Data Frame, para asegurarnos de todos los cambios hechos previamente. Cambiamos algunos tipos de datos por date.time, y organizamos el

Data Frame de menor a mayor por la variable
FECHA_MATRICULA_RENOVACION_HORA

```
[7] ▶ M4  
#Se organiza el DataFrame según la fecha del movimiento  
registrado. Esto facilitará las particiones que se harán de  
la base de datos  
df=df.sort_values('FECHA_MATRICULA_RENOVACION_HORA')  
print(df.info())  
df.head(2)
```

8. Siguiendo las indicaciones de las reuniones sostenidas entre Sistemas, Registro y EE se deben eliminar las filas de las empresas que registren en el ESTADO_MAT_REN la letra R ya que estos movimientos fueron anulados por alguna acción legal.

En la mayoría de los casos se anulan las renovaciones por motivo de cancelación. Pero, aquellas renovaciones hechas antes el 31 de marzo de cada año ya que la ley permite que las empresas cancelen su registro sin pagar la renovación, solo si se realiza el proceso antes del 31 de marzo de cada año.

En esa línea se muestra como quitar esas empresas con la letra R sin embargo se aconseja hacerlo al final del código. Preguntar por qué.

En esta línea revisamos cuantos registros hay por cada ESTADO_MAT_REN.

```
[8] ▶ M4  
df.groupby('ESTADO_MAT_REN').nunique()
```

	MATRICULA	NIT	MATRICULA_RENOVACION
ESTADO_MAT_REN			
A	91424	91181	2
D	814	814	2
R	410	410	1

9. En esta línea se obtienen los duplicados, **sin tener en cuenta** la primera observación, revisando todas las columnas. Es decir, estamos revisando que no hayan registros exactamente la misma información para todas las columnas. Ejemplo:

FELIPE 24 MASCULINO

FELIPE 24 MASCULINO

También se imprime el número de registros duplicados con todas las columnas iguales y se saca un conteo de dichos duplicados teniendo en cuenta la variable MATRICULA.

```
[9] ▶ M4
# Para obtener los duplicados EXCEPTO la primera observación
# revisando todas las columnas
duplicados = df[df.duplicated()]
print('El número de registros duplicados con todas las
columnas iguales son:')
print (duplicados.MATRICULA.count())

El número de registros duplicados con todas las columnas iguales son:
0
```

10. En la línea anterior no encontramos duplicados exactos revisando por cada fila, cada variable que hay en las columnas para cada registro. Sin embargo, si revisamos únicamente la columna de MATRICULA si se encuentra duplicados.

Entonces, en esta línea estamos buscando obtener el número de duplicados EXCEPTO la última observación revisando únicamente la columna MATRICULA. Es decir, se tomarán todas las matrículas duplicadas, exceptuando la que registró la última modificación (recuerde que previamente la BD se organizó según FECHA_MATRICULA_RENOVACION_HORA).

En el *print* se incluyó el `sep = '\n'` para realizar un salto de línea en la salida.

```
[10] ▶ M4
# Para obtener los duplicados EXCEPTO la última observación
# revisando la columna Matriculados. Es decir, se tomarán
# todas las matrículas duplicadas excepto la que registró la
# última modificación
duplicadosMAT = df[df.duplicated(['MATRICULA'], keep='last')]
print("El número de Matriculas duplicadas es:",
duplicadosMAT.MATRICULA.count(), sep='\n')

El número de Matriculas duplicadas es:
19690
```

11. En este punto debemos obtener los duplicados EXCEPTO la última observación revisando a la vez las columnas MATRICULA y MATRICULA_RENOVACIÓN.

Es decir, se tomarán todas las matrículas duplicadas tanto por su número de matrícula como por lo que hayan hecho ese año (matrícula o renovación). Es decir, si se presentaron dos renovaciones o dos matrículas se tomarán en la sub-base creada.

Siempre es importante aclarar que esto se debe hacer EXCEPTUANDO aquella empresa que registró la última modificación (keep = 'last')¹.

```
[11] ▶ MI
# Para obtener los duplicados EXCEPTO la última observación
# revisando en simultaneo las columnas Matriculados y
# MATRICULA_RENOVACION. Es decir, se tomarán todas las
# matrículas duplicadas tanto con su número de matrícula como
# lo que hayan hecho en ese año, es decir, si presentó dos
# renovaciones o dos matrículas se tomarán en la subbase
# creada. Es necesario acalarar que lo anterior se hace
# exceptuando la que registró la ultima modificación. Esto
# puede ocurrir debido a que los tramites de renovación de
# alguna empresa estuveiron mal y se tardaron más del plazo
# para entregar sus papeles así que se registra una nueva
# fecha de renovación sin eliminar la primera.
duplicadosMAT_REN = df[df.duplicated(['MATRICULA',
'MATRICULA_RENOVACION'], keep='last')]
print("El número de filas duplicadas analizando dos columnas
es:", duplicadosMAT_REN.MATRICULA.count(), sep='\n')

El número de filas duplicadas analizando dos columnas es:
645
```

12. Lo anterior nos indica que debemos trabajar con bases de datos separadas controlando por la variable MATRICULA_RENOVACION. Debido a que nos interesa trabajar con el último movimiento de las empresas por matrícula y con el último movimiento por renovación.

Antes de realizar la limpieza, en esta línea revisamos cuantas matrículas y cuantas renovaciones hay sin eliminar los duplicados. Para saber finalmente cuantos registros se van a limpiar.

¹ Esta limpieza se debe hacer porque puede suceder que una persona haya hecho los trámites de renovación pero algún papel tuvo un error. La Cámara le da un tiempo de 15 días al empresario para que traiga el papel correcto. Sin embargo, si la persona trae el papel el día 15+n con n!=0 el sistema toma la renovación del empresario como nueva. Es decir, añade una fila con el estado renovación, y la anterior (donde el papel estaba malo) no la elimina.

```
[12] ▶ M4
# Antes de partir la base de datos. Se hace un conteo para
# saber cuantas Matriculas y Renovaciones se tienen en la base
# sin eliminar duplicados
df.MATRICULA_RENOVACION.value_counts()

Renovacion    92604
Matricula     19047
Name: MATRICULA_RENOVACION, dtype: int64
```

13. Como ya se sabe que se debe partir la base de datos en dos para hacer la limpieza correspondiente. Se requieren tratar las empresas Matriculadas y Renovadas diferenciadas por su fecha de registro (siguiendo esa lógica del último movimiento).

Para esta línea se deben utilizar:

- `df.loc`: atributo en Pandas para acceder a un grupo de filas y columnas por etiqueta o una matriz booleana en el Data Frame dado.

Sintaxis: `DataFrame.loc` o `df.loc`

Parámetro: Ninguno

Devoluciones: scalar, series, Dataframe

- `isin()`: método en Pandas que se utiliza para filtrar marcos de datos. El método ayuda a seleccionar filas con un valor particular (o múltiple) en una columna en particular.

Sintaxis: `DataFrame.isin (valores)`

Parámetros

valores: iterable, series, list, tuple, dataframe o diccionario para verificar en el llamador series / Data frame.

Tipo de retorno: Data Frame of Booleano of Dimension

Ahora bien se crean los dos Data Frames para manipular Matrículas y Renovaciones por separado. (ojo aquí empezamos a trabajar con dos Data Frames, tenerlos bien identificados).


```
[13] ▶ ML

# Ahora es necesario partir la base de datos en dos para
# hacer la correspondiente limpieza. Se requieren tratar las
# empresas Matriculadas y Renovadas diferenciadas por su fecha
# de registro
df_matricula = df.loc[df['MATRICULA_RENOVACION'].isin(
['Matricula'])]
df_renovacion = df.loc[df['MATRICULA_RENOVACION'].isin(
['Renovacion'])]
print('El número de registros en la BD para Matriculas es:',
df_matricula.MATRICULA_RENOVACION.value_counts(), sep='\n')
print('El número de registros en la BD para Renovaciones:',
df_renovacion.MATRICULA_RENOVACION.value_counts(), sep='\n')

El número de registros en la BD para Matriculas es:
Matricula    19047
Name: MATRICULA_RENOVACION, dtype: int64
El número de registros en la BD para Renovaciones:
Renovacion    92604
Name: MATRICULA_RENOVACION, dtype: int64
```

14. En este punto deber ir el código después de limpiar el tema de renovaciones más viejas que las matrículas. Es decir, empresas de otras cámaras de comercio que pasan sus registros a la CCC y en el sistema aparecen con fechas de matrícula mayores que los movimientos de renovación (ejemplo MAO: la empresa no puede cumplir años sin nacer).

Se deben eliminar las RENOVACIONES con fechas anteriores a las de las MATRICULAS porque esto es un indicador que de la empresa hizo un traslado de domicilio y trajo todo su historial a la CCC.

También se debe convertir la variable **FECHA_MATRICULA_RENOVACION** en datetime para facilitar la manipulación de los datos. (ojo aquí estamos trabajando con el data frame de renovación)

```
[14] ▶ ML

# Aquí DEBE IR EL CODIGO DESPUES DE LIMPIAR EL TEMA DE
# RENOVACIONES MAS VIEJAS QUE LAS MATRICULAS. SE DEBEN
# ELIMINAR LAS RENOVACIONES CON FECHAS ANTERIORES A LAS DE LAS
# MATRICULAS porque esto es un indicador que la empresa hizo
# un traslado de domicilio y se trajo todo su historial a la
# CCC
# Se convierte la variable FECHA_MATRICULA_RENOVACION_HORA
# en datetime para facilitar manipulación de los datos
df_renovacion.FECHA_MATRICULA_RENOVACION = pd.to_datetime(
df_renovacion.FECHA_MATRICULA_RENOVACION)
df_renovacion = df_renovacion.loc[df_renovacion
['FECHA_MATRICULA']<df_renovacion
['FECHA_MATRICULA_RENOVACION']]

C:\Users\lmonten\Anaconda3\lib\site-packages\pandas\core\generic.py:5
168: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/panda
s-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
self[name] = value
```

15. Se deben organizar las BD según la fecha en que se hizo el movimiento correspondiente a MATRICULA o RENOVACIÓN para la limpieza posterior.

Es importante aclarar que el ordenamiento es DESCENDENTE (es el que está por defecto. El ascending por defecto es false). Es decir, comienza desde YYYY/01/01 00:00 hasta YYYY/12/31 23:59 (de enero a diciembre).

```
[15] ▶ MI
# Se organizan las bases según la fecha en la cual se hizo
# el movimiento correspondiente a Matricul o Renovación para
# posterior limpieza. Vale aclarar que esta el ordenamiento es
# descendente, es decir, se comienza desde YYYY/01/01 00:00
# hasta YYYY/12/31 23:59 (de enero a diciembre)
df_matricula=df_matricula.sort_values
('FECHA_MATRICULA_RENOVACION_HORA')
df_renovacion=df_renovacion.sort_values
('FECHA_MATRICULA_RENOVACION_HORA')
df_renovacion.head(2)
```

16. Ya teniendo el paso anterior listo, se debe proceder a eliminar los duplicados en las dos BD. Recuerde que la idea es dejar el **último movimiento registrado** por eso configuramos el orden en el paso anterior.

Esto se debe gracias a que se identificó que algunas empresas registran algún movimiento y por algún motivo este es devuelto. En la mayoría de los casos al corrección sugerida es hecha en el plazo para mantener la fecha inicial del movimiento. Sin embargo, cuando la empresa deja pasar este plazo máximo se debe generar un nuevo movimiento, lo cual genera duplicados en la información.

```
[16] ▶ MI
# Se procede a eliminar los duplicados en las dos bases y la
# idea es dejar el último movimiento registrado. Esto se debe
# gracias a que se identificó que algunas empresas registran
# algun movimiento y por algun motivo este es devuelto, en la
# mayoría de los casos la corrección sugerida es hecha en el
# plazo para mantener la fecha inicial del movimiento, sin
# embargo, cuando la empresa deja pasar este plazo máximo se
# debe generar un nuevo movimiento lo cual genera duplicados
# en la información
df2_matricula=df_matricula.sort_values
('FECHA_MATRICULA_RENOVACION_HORA').drop_duplicates
('MATRICULA', keep='last')
df2_renovacion=df_renovacion.sort_values
('FECHA_MATRICULA_RENOVACION_HORA').drop_duplicates
('MATRICULA', keep='last')
```

17. Debido a que ya eliminamos duplicados, contamos el número de registros que nos quedaron

```
[17] ▶ MI
# Una vez eliminados los duplicados se cuenta el número de
registros
print('El número de registros en la BD para Matriculas
después de eliminar duplicados es:',
df2_matricula.MATRICULA_RENOVACION.value_counts(), sep='\n')
print('El número de registros en la BD para Renovaciones
después de eliminar duplicados es:',
df2_renovacion.MATRICULA_RENOVACION.value_counts(), sep='\n')
```

18. Como ya sabemos que corregimos todos los problemas que tiene la BD en relación con los problemas de duplicados. Procedemos a unir las dos BD que creamos previamente.

Ahora bien, el `ascending= True` lo utilizamos porque necesitamos que el orden por `MATRICULA_RENOVACIÓN` no sea DESCENDENTE. Sabemos por defecto el `ascending=false`, es decir, va bajando de enero a diciembre. Yo lo quiero de diciembre a enero, le digo que seas ASCENDENTE entonces por eso le damos `True`.

Es así porque necesitamos que las matrículas se queden primero y luego queden las renovaciones. (MIGUEL PERO CÓMO SI ESTA VARIABLE NO TIENE FECHA)

```
[18] ▶ MI
# Se procede a unir las bases de datos para la posterior
limpieza en conjunto
df2 = pd.concat([df2_matricula, df2_renovacion])
df2 = df2.sort_values('MATRICULA_RENOVACION', ascending=True)
print('El número de registros en la BD unida es:',
df2.MATRICULA_RENOVACION.value_counts(), sep='\n')
El número de registros en la BD unida es:
Renovacion    91948
Matricula      19046
Name: MATRICULA_RENOVACION, dtype: int64
```

19. Ya teniendo nuestra BD armada (`df2`) procedemos a crear una nueva BD que tendrá o problemas en términos de duplicados. Porque recuerde que nos interesa dejar el primer registro que tengamos en la BD, pues por cada matrícula se genera una renovación (este es el problema que siempre conocíamos, anteriormente limpiamos nuevos inconvenientes, por eso le decimos `KEEP= first`).

```
[19] ▶ MI
# Para eliminar duplicados se debe tener en cuenta que en
# Matricula nos interesa dejar el primer registro que tengamos
# en la base pues por cada matricula se genera una renovación
df3 = df2.sort_values('MATRICULA_RENOVACION', ascending=True)
df3.drop_duplicates('MATRICULA', keep='first')
print('El número de registros en la BD unida sin duplicados
es:', df3.MATRICULA_RENOVACION.value_counts(), sep='\n')

El número de registros en la BD unida sin duplicados es:
Renovación    72911
Matricula      19846
Name: MATRICULA_RENOVACION, dtype: int64
```

20. Nos volvemos a asegurar por última vez que no tengamos duplicados por MATRICULA:

```
[20] ▶ MI
duplicadosMAT3 = df3[df3.duplicated(['MATRICULA'],
keep='last')]
print("El número de Matriculas duplicadas es:",
duplicadosMAT3.MATRICULA.count(), sep='\n')

El número de Matriculas duplicadas es:
0

Desde aquí se inicia la limpieza de otras variables para facilitar
los posteriores análisis
```

Hasta este punto la BD está limpia. Sin embargo se hace limpieza adicional de otras variables para facilitar análisis posteriores (se corrigen barrios, comunas, ciudades, etc).

21. En esta línea listamos por ciudad cuantas matrículas hay.

Dataframe.groupby() se usa para separar los datos en grupos basado en diferentes criterios. Los objetos de pandas pueden ser divididos en cualquiera de sus ejes. (df3.groupby(variable a agrupar))

Al analizar datos, muchas veces el usuario desea ver valores únicos en columnas específicas. Pandas nunique () se usa para obtener valores únicos. [columna].nunique()

pandas.Series.unique

Series.[`unique\(dropna=True\)`](#)

Return number of unique elements in the object.

Excludes NA values by default.

Parameters:

dropna: boolean, default True
Don't include NaN in the count.

Returns:

nunique: int

```
[21] df3.groupby('CIUDAD')['MATRICULA'].nunique()

CIUDAD
Cali      83372
Dagua     756
Jamundi   3723
La Cumbre 273
Palmira    1
Vijes     197
Yumbo     3632
Name: MATRICULA, dtype: int64
```

22. Como sabemos que en las variables *Comuna* y *Ciudad* hay diversos problemas relacionados con digitación errónea (*Comuna* o por ejemplo). También queremos reemplazar las comunas con nombres de municipios por el nombre del municipio, por ejemplo, cambiar *comuna Dagua* por *Dagua*.

En el caso que en la BD encontremos ciudades diferentes a la jurisdicción de la CC se reemplazarían por Cali para facilitar el análisis. Claramente es un posible error de digitación.

ENTONCES:

Utilizamos el código *pandas.DataFrame.replace*:

**DataFrame.replace(to_replace=None,value=None,
inplace=False, limit=None, regex=False, method='pad')**

Los Valores del DataFrame son reemplazados con otros valores de forma dinámica. Esto difiere de la actualización con `.loc` o `.iloc` que requiere que especifique una ubicación para actualizar con algún valor.

Parameters: **to_replace** : *str, regex, list, dict, Series, int, float, or None*

How to find the values that will be replaced.

- numeric, str or regex:
 - numeric: numeric values equal to *to_replace* will be replaced with *value*
 - str: string exactly matching *to_replace* will be replaced with *value*
 - regex: regexs matching *to_replace* will be replaced with *value*

value : *scalar, dict, list, str, regex, default None*

Value to replace any values matching *to_replace* with. For a DataFrame a dict of values can be used to specify which value to use for each column (columns not in the dict will not be filled). Regular expressions, strings and lists or dicts of such objects are also allowed.

inplace : *bool, default False*

If True, in place. Note: this will modify any other views on this object (e.g. a column from a DataFrame). Returns the caller if this is True.

En df3 en la variable comuna reemplaze:

```
[22] ▶ ▶≡ M4

# En esta línea de código se limpian los diferentes valores
de Comuna que se tienen
df3.COMUNA.replace(to_replace=["Comuna 01", "Comuna 02",
                              "Comuna 03", "Comuna 04", "Comuna 05", "Comuna 06", "Comuna
                              07", "Comuna 08", "Comuna 09"],
                  value=["Comuna 1", "Comuna 2",
                          "Comuna 3", "Comuna 4", "Comuna 5", "Comuna 6", "Comuna 7",
                          "Comuna 8", "Comuna 9"], inplace=True)
# En esta línea se reemplazan las comunas con nombre de
municipio por el nombre del municipio
df3.COMUNA.replace(to_replace=["Comuna Dagua", "Comuna
                              Jamundi", "Comuna La Cumbre", "Comuna Vijes", "Comuna Yumbo"]
                  ,
                  value=["Dagua", "Jamundi", "La
                          Cumbre", "Vijes", "Yumbo"], inplace=True)
# En esta línea se reemplazan las ciudades diferentes a la
jurisdicción de la CC por Cali para facilitar el análisis
df3.CIUDAD.replace(to_replace=["Bogota", "Palмира",
                              "Piendamó", "Calima Darien", "Sevilla"],
                  value="Cali", inplace=True)
df3.groupby('CIUDAD')['MATRICULA'].nunique()
```

```
[23] ▶ Mu
```

```
#PN Y PJ  
dict_PNPJ = {  
    'NOM_ENTE_JURIDICO': ["Persona Natural", "Sociedad por Acciones Simplificada", "Sociedad en Comandita Simple", "Sociedad Limitada", "Empresa Unipersonal", "Sociedad Anónima", "Societades Extranjeras", "Sociedades Civiles", "Empresa Asociativa de Trabajo", "Sociedad Colectiva", "Fondos de Pensiones"],  
  
    'PN_PJ_CORR': [ "Persona Natural",      "Persona Juridica",   "Persona Juridica",   "Persona Juridica",   "Persona Juridica",  
                    "Persona Juridica",     "Persona Juridica",   "Persona Juridica",   "Persona Juridica",   "Persona Juridica",  
                    "Persona Juridica"] ,  
}
```

24. En esta línea creamos formalmente el Data Frame utilizando el diccionario de datos que creamos previamente referente a las PN y PJ. Esto con el objetivo de hacer el cruce posteriormente.

De la explicación solo estamos usando los parámetros data y columns:

pandas.DataFrame

`class pandas.DataFrame(data=None, index=None, columns=None, dtype=None, copy=False)` [\[source\]](#)

Two-dimensional, size-mutable, potentially heterogeneous tabular data.

Data structure also contains labeled axes (rows and columns). Arithmetic operations align on both row and column labels. Can be thought of as a dict-like container for Series objects. The primary pandas data structure.

Parameters: **data** : ndarray (structured or homogeneous), Iterable, dict, or DataFrame
Dict can contain Series, arrays, constants, dataclass or list-like objects. If data is a dict, column order follows insertion-order.
Changed in version 0.25.0: If data is a list of dicts, column order follows insertion-order.

index : Index or array-like
Index to use for resulting frame. Will default to RangeIndex if no indexing information part of input data and no index provided.

columns : Index or array-like
Column labels to use for resulting frame. Will default to RangeIndex (0, 1, 2, ..., n) if no column labels are provided.

dtype : dtype, default None
Data type to force. Only a single dtype is allowed. If None, infer.

copy : bool, default False
Copy data from inputs. Only affects DataFrame / 2d ndarray input.

<https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.html>

```
[24] ▶ ▶ M1
# Se crea el dataframe con el diccionario de datos referente
# a los PN y PJ para el posterior cruce de información
df_PNPJ = pd.DataFrame(dict_PNPJ, columns =
["NOM_ENTE_JURIDICO", "PN_PJ_CORR"])
```

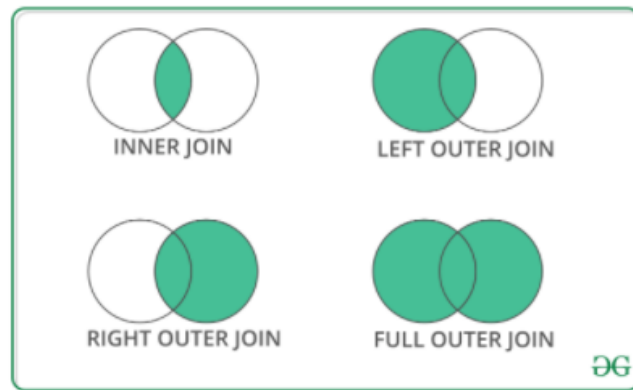
25. Pandas tiene opciones de alto rendimiento para fusionar y unir en memoria. Cuando necesitamos combinar DataFrames muy grandes, las uniones sirven como una forma poderosa de realizar operaciones con rapidez.

Las uniones solo se pueden realizar en dos DataFrames a la vez indicados como tablas *izquierda* y *derecha*.

La *clave* es la columna común en la que se unirán los dos DataFrames.

Es una buena práctica utilizar *claves* que tengan valores únicos en toda la columna para evitar la duplicación no intencionada de los valores de las filas. Pandas provee una función única: *merge()*, como el punto de entrada para todas las operaciones de unión de bases de datos estándar entre objetos DataFrame.

Hay cuatro formas básicas de manejar la combinación (interna, izquierda, derecha y externa)(inner, left, right and outer), dependiendo de las filas que deben conservar sus datos.



FUSIONAR DATAFRAME USANDO *how* COMO ARGUMENTO

Utilizamos *how* como argumento para fusionar específicamente cómo determinar qué clave se incluirá en la tabla resultante. Si una combinación de claves no aparece en las tablas de la izquierda o de la derecha, los valores de la tabla combinada será NA. A continuación, se muestra un resumen de las opciones de *how* y sus nombres equivalente en SQL:

MERGE METHOD	JOIN NAME	DESCRIPTION
left	LEFT OUTER JOIN	Use keys from left frame only
right	RIGHT OUTER JOIN	Use keys from right frame only
outer	FULL OUTER JOIN	Use union of keys from both frames
inner	INNER JOIN	Use intersection of keys from both frames

<https://www.geeksforgeeks.org/python-pandas-merging-joining-and-concatenating/>

ENTONCES:

Se unen las dos BD (df3 + df_PNPJ) para obtener la información de Persona Natural y Persona Jurídica corregidos.

```
[25] ▶ ▶≡ MI  
# Se unen las dos bases de datos para obtener la información  
# de PN y PJ corregidos  
df3 = pd.merge(df3, df_PNPJ, how='left', on=  
['NOM_ENTE_JURIDICO', 'NOM_ENTE_JURIDICO'])
```

Para agregar nuevos valores al diccionario se recomienda trabajar con el **archivo txt para hacer las pruebas**.

Finalmente, se recomienda actualizar el diccionario agregando en cada campo el valor correspondiente, es decir, para cada actualización se debe contar con:

- Barrio registrado en la BD (BARRIO_REG)
- Nombre del barrio corregido (NOM_BAR)
- Código del barrio en el shapefile del IDESC (COD_BAR)

Se agregan como *string* al final de cada aptado para facilitar el manejo de datos.

27. En esta línea se crea un diccionario para corregir el tema de los barrios. Incluimos como está escrito el barrio en la BD, luego el nombre del barrio corregido y luego el código del barrio para posteriores análisis utilizando la herramienta Qgis.

```
[27] ▶ ▶≡ MI  
#BARRIOS  
  
dict_barrios = {  
    'BARRIO_REG': ["La Legua", "Palermo", "Terror Colorado",  
    "Vista Hermosa", "Sector Patio Bonito", "Aguacatal", "Alto  
    Aguacatal", "Bajo Aguacatal-Las Colinas", "Santa Rita", "Santa  
    Teresita", "Arboledas", "Normandia", "Juanambu", "Centenario",  
    "Granada", "Versalles", "San Vicente", "Prados Del Norte (N Sn  
    Vic.)", "La Flora", "Area Libre-Parque Del Amor", "La Campina",  
    "La Paz", "La Paz (Cabecera)", "El Bosque", "Menga", "Ciudad Los  
    Alamos", "Chipichape", "Brisas De Los Alamos", "Urbanizacion La  
    Merced", "Vipasa", "Urbanizacion La Flora", "Altos De Menga",  
    "Altos De Normandia", "Altos Santa Monica", "El Filo",  
    "Golondrinas (Cabecera)", "La Maria", "San Isidro", "Sector
```

28. En esta línea creamos formalmente el Data Frame utilizando el diccionario de datos que creamos previamente referente a los nombres de barrio. Esto con el objetivo de hacer el cruce posteriormente.

```
[28] ▶ ⌵ M4

# Se crea el dataframe con el diccionario de datos referente
# a los nombre de los barrios para el posterior cruce de
# información
df_barrios = pd.DataFrame(dict_barrios, columns =
['BARRIO_REG', 'NOM_BAR', 'COD_BAR'])
```

Preguntar a Miguel por la anotación que está después de esta línea.

29. Para poder tener un cruce satisfactorio debemos volver las llaves a letras mayúsculas en ambos dataframes (df3 y df_barrios).

pandas.Series.str.upper

Series.str.upper()

Convierta las cadenas de Series / Index a mayúsculas.

Equivalente a `str.upper()`.

Devoluciones: Serie o índice de objeto

```
[29] ▶ ⌵ M4

# Para unir las bases es necesario volver las llaves en
# mayuscula en ambos dataframes
df3['BARRIO'] = df3['BARRIO'].str.upper()
df_barrios['BARRIO_REG'] = df_barrios['BARRIO_REG'].str.upper()
()
```

Me genera la duda de por qué en algunos se usan los [] y en otros como en el siguiente solo puntos.

30. Como en nuestra BD df3 nuestra variable objetivo se llama BARRIO, entonces, renombramos la variable a cruzar en el DataFrame df_barrios para poder tener un cruce satisfactorio.

```
[30] ▶ ⌵ M4

# Se renombra la variable a cruzar para que tenga el mismo
# nombre en las dos bases
df_barrios.rename(columns={'BARRIO_REG':'BARRIO'},
inplace=True)
```

31. En esta línea lo que hacemos es eliminar los posibles espacios en blanco que hay en las llaves para evitar errores a la hora de cruzar las BD. Esta acción debe hacerse en ambas BD.

```
[31] ▶ ▶≡ M4  
  
# Se eliminan los espacio en blanco de las llaves para  
evitar errores al cruzar las bases de datos  
df3.BARRIO = df3.BARRIO.str.replace(' ', '')  
df_barrios.BARRIO = df_barrios.BARRIO.str.replace(' ', '')
```

32. Al eliminar los espacios en blanco se pueden generar llaves duplicadas, esto generaría filas duplicadas al realizar un left join. En ese sentido, deben eliminar las llaves duplicadas del diccionario:

```
[32] ▶ ▶≡ M4  
  
# Al eliminar los espacios se pueden generar llaves  
duplicadas, lo cual genera filas duplicadas al realizar un  
left join, por tal motivo, se eliminan las llaves duplicadas  
del diccionario  
duplicados_df_barrios = df_barrios[df_barrios.duplicated(  
    ['BARRIO'], keep='last')]  
print("El número de llaves duplicadas es:",  
    duplicados_df_barrios.BARRIO.count(), sep='\n')  
  
El número de llaves duplicadas es:  
3
```

33. Se deben eliminar las llaves duplicadas del diccionario de Barrios (revisar por qué se utiliza el sort):

```
[33] ▶ ▶≡ M4  
  
# Se eliminan las llaves duplicadas del diccionario de Barrios  
df_barrios=df_barrios.sort_values('BARRIO').drop_duplicates  
('BARRIO', keep='last')
```

34. En esta línea se unen las dos BD para obtener la información de los barrios corregidos. El `astype(str)` se utiliza para asegurarnos que el tipo de datos se mantenga en string y poder hacer operaciones de forma satisfactoria:

```
[34] ▶ ▶≡ MI

# Se unen las dos bases de datos para obtener la información
de barrios corregidos
df3.BARRIO = df3.BARRIO.astype(str)
df_barrios.BARRIO = df_barrios.BARRIO.astype(str)
df3 = pd.merge(df3, df_barrios, how='left', on=['BARRIO',
'BARRIO'])
```

35. En esta línea vamos a crear un diccionario tal cual como hicimos con los ENTES JURIDICOS y BARRIOS. Pero ahora vamos a corregir todos los problemas que se pueden generar en el diligenciamiento de los código CIIU.

```
[35] ▶ ▶≡ MI

#CIIU

dict_CIIU = {
    'CODIGO_CIIU': ["12100", "722000",
"851202", "111", "112", "113", "114", "119",
"121", "122", "123", "124", "125", "126", "127",
"128", "129", "130", "141", "142", "143", "144",
"145", "149", "150", "161", "162", "163", "164",
"210", "220", "240", "311", "312", "321", "322",
"510", "710", "722", "729", "811", "812", "820",
"899", "910", "990", "1011", "1012", "1020", "1030",
"1040", "1051", "1052", "1061", "1062", "1063", "1071",
"1072", "1081", "1082", "1083", "1084", "1089", "1090",
"1101", "1102", "1103", "1104", "1200", "1311", "1312",
"1313", "1391", "1392", "1393", "1394", "1399", "1410",
"1420", "1430", "1511", "1512", "1513", "1521", "1522",
"1523", "1610", "1620", "1630", "1640", "1690", "1701"]
}
```

Con la corrección se soluciona todo, grupos, divisiones, descripciones, etc. Importante revisar el código para ver la profundidad del análisis y corrección a este nivel de la información.

36. En esta línea, creamos formalmente el DataFrame con el diccionario que se creó en el paso anterior que hace referencia a la limpieza por CIIU, para poder realizar el posterior cruce de información de forma exitosa.

Básicamente le decimos al programa que cree un DataFrame llamado *df_CIIU* que se alimente a partir del diccionario *dict_CIIU* y de dicho diccionario tome las variables que aparecen en naranja.

```
[36] ▶ ▶≡ MI

# Se crea el dataframe con el diccionario de datos referente
# a los CIIU para el posterior cruce de información
df_CIIU = pd.DataFrame(dict_CIIU, columns = ["CODIGO_CIIU",
"CIIU_COR", "DESC_CIIU_COR", "SECCION_COR",
"DESC_SECCION_COR", "DIVISION", "DES_DIVI", "GRUPO",
"DES_GRU"])
```

37. En esta línea, para que el cruce se haga de forma exitosa entre las dos BD se debe establecer que el tipo de llave que se utilizará es de tipo *Str*.

```
[37] ▶ ▶≡ MI

# Para la correcta unión entre las bases se procede
# establecer el tipo llave que se utilizará "str"
df3.CODIGO_CIIU = df3.CODIGO_CIIU.astype(str)
df_CIIU.CODIGO_CIIU = df_CIIU.CODIGO_CIIU.astype(str)
```

38. Se unen las dos BD (*df3* y *df_CIIU*) para obtener la información de códigos CIIU corregidos.

```
[38] ▶ ▶≡ MI

# Se unen las dos bases de datos para obtener la información
# con código CIIU corregidos
df3 = pd.merge(df3, df_CIIU, how='left', on=['CODIGO_CIIU'],
indicator="indicator_column")
```

39. Teniendo en cuenta las indicaciones que se recibieron en las reuniones con Sistemas y Registro se deben eliminar las filas de las empresas que registren en el ESTADO_MAT_REN la letra R.

Esto debido a que dichos movimientos fueron anulados por alguna acción legal.

En la mayoría de casos se anulan las RENOVACIONES por motivo de CANCELACIÓN hechas antes del 31 de marzo de cada año. Ya que la ley permite que las empresas cancelen su registro sin pagar la renovación. Siempre y cuando se realice el proceso antes del 31 de marzo de cada año.

Entonces el uso del `.query` nos ayuda a filtrar la información según nuestro objetivo, el cual es mantener todos los registros que tengan el ESTADO_MAT_REN diferente a R.

```
[39] ▶ ⚙ MI
# Siguiendo las indicaciones de Sistemas y Registro se deben
# eliminar las filas de las empresas que registren en el
# ESTADO_MAT_REN la letra R ya que estos movimientos fueron
# anulados por alguna acción legal. En la mayoría de los casos
# se anulan las renovaciones por motivo de cancelación hechas
# antes del 31 de marzo de cada año ya que la ley permite que
# las empresas cancelen su registro sin pagar la renovación
# solo si se realiza el proceso antes del 31 de marzo de cda
# año
df4 = df3.query('ESTADO_MAT_REN!="R"')
```

Preguntar acerca del inplace por defecto que hay detrás:
<https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.query.html>

40. Finalmente, guardamos en Excel y en formato CSV el último DataFrame, el cual tiene todos los cambios realizados en términos de limpieza de información:

```
[40] ▶ ⚙ MI
# Para guardar la base de datos lista para ser tratada en
# formato Excel y CSV
df4.to_excel(r'RM_DIC2020_V2(ENE2021_EEE3).xlsx', index =
False)
#df4.to_csv(r'RM_FEB2021_V2(MAR2021_EEE3).csv', index =
False, encoding='utf-8-sig')
```