Introducción a Pandas

March 11, 2024

1 Introducción a Pandas

Elaboración: Gabriel Armando Landín Alvarado

1.1 Definición de la libreria Pandas

Pandas es una libreria "open-source" para análisis de datos de Python, de acuerdo con la documentación oficial, **pandas** se encuentra bajo una licencia BSD, también conocida como licencia de Distribución de Software de Berkeley, siendo un tipo de licencia de código abierto que permite su uso, modificación y distribución de manera libre. Es una libreria que proporciona estructuras de datos para su análisis en alto rendimiento, las cuales son relativamente fáciles de manipular con el lenguaje de programación Python.

1.2 Contenido

- Estructuras de datos
  - Serie
  - DataFrame
- Carga de datos a partir de archivos externos en un DataFrame
  - csv
  - xlsx
  - txt
- Exploración de los datos
- Selección de datos
  - Selección de datos para un subconjunto de columnas
  - Selección de datos usando una condición simple
  - Selección de datos con múltiples condiciones
- Limpieza de datos
  - Registros duplicados
  - Valores perdidos
- Agrupar y agregar
  - Agrupar
  - Agregar
- Ordenar y clasificar
  - Ordenar
  - Clasificar
- Agregar filas
- Agregar columnas
- Eliminar filas y/o columnas

- Concatenar DataFrames
- Fusionar o unir DataFrames
  - Fusionar
  - Unir
- Escritura de un DataFrame en archivos externos

1.3 Estructuras de datos

1.3.1 Serie

Una Serie de pandas es una colección de elementos que poseen un índice, en un termino más técnico, es un array o arreglo de una dimensión con etiquetas con índice, éstas etiquetas hacen referencia a un eje, en python el índice inicia en 0.

- Creando una Serie de pandas

Para crear una Serie se tiene la función con los siguientes parámetros: **pandas.Series(data, index=index)**, en el siguiente ejemplo se crea una Serie a partir de una lista de números enteros, se observa que Python crea por defecto el índice iniciando en 0.

[1]:
```
# importando las libreria de pandas
import pandas as pd
list_num = [1, 2, 3, 4]
series_num = pd.Series(list_num)
# tipo de dato de series_num
print("Tipo de dato del objeto series_num: {}".format(type(series_num)))
series_num
```

```
Tipo de dato del objeto series_num: <class 'pandas.core.series.Series'>
```

[1]:
```
0    1
1    2
2    3
3    4
dtype: int64
```

1.3.2 DataFrame

Un DataFrame de pandas es una estructura de datos de dos dimensiones, este objeto es similar a una tabla de SQL pues se encuentra ordenado en filas y columnas. La estructura DataFrame es la más comunmente utilizada para la manipulación y gestión de datos, al igual que una Serie, python asigna un índice pór defecto, el cual puede ser modificado de acuerdo con el argumento *index=index*.

- Creando un DataFrame de pandas Para crear un DataFrame de pandas se tiene la función **pandas.DataFrame(data, index=index, columns=[column(s)])**; en los siguientes ejemplos se construyen dos DataFrames a partir de una lista y de un diccionario:

[2]:
```
# DataFrame a partir de una lista
list_num = [1, 2, 3, 4]
```

```
df_num = pd.DataFrame(list_num, columns=['numero']) # en este ejemplo no␣
 ↪pasamos un indice, pandas lo asigna automáticamente [0,1,2,3]
# tipo de dato de df_num
print("Tipo de dato de df_num: {}".format(type(df_num)))
df_num
```

```
Tipo de dato de df_num: <class 'pandas.core.frame.DataFrame'>
```

[2]:

	numero
0	1
1	2
2	3
3	4

[3]:
```
# DataFrame a partir de un diccionario
dicc_num = {'numero':[1, 2, 3, 4]}
df_dicc = pd.DataFrame(dicc_num) # en este caso no se establece el nombre de la␣
 ↪columna ya que python toma la clave del diccionario para el mismo
print("Tipo de dato de df_dicc: {}".format(type(df_dicc)))
df_dicc
```

```
Tipo de dato de df_dicc: <class 'pandas.core.frame.DataFrame'>
```

[3]:

	numero
0	1
1	2
2	3
3	4

1.4 Carga de datos a partir de archivos externos en un DataFrame

La libreria de pandas tiene varias funciones para leer y cargar diferentes tipos de archivos externos dentro de un DataFrame. A continuación, se muestran algunos ejemplos con los formatos más comúnes: CSV, Excel(xlsx) y TXT. Los archivos corresponden a datos de algunas organizaciones o compañias de varios países, en español los nombres de las columnas son: nombre, país, año de fundación, tipo de industria y número de empleados.

1.4.1 CSV

CSV es la abreviación de **Comma Separated Values**, lo más común es que cada valor este separado con una coma, este archivo puede contener o no un encabezado o columnas, la función para leer y crear un DataFrame a partir de este tipo de archivo es: **pandas.read_csv('filepath')**, esta función puede contener diversos parámetros como: *sep=''*, *header=*, *names=*, *index_col=*, entre otros, los cuales pueden ser revisados a detalle para esta función y otras más en la documentación oficial de Pandas.

A continuación, se lee un archivo con encabezados y que pandas reconoce.

[4]:
```
df = pd.read_csv('../data/organizations.csv')
df
```

[4]:
```
                      Name                       Country  Founded  \
0              Ferrell LLC             Papua New Guinea     1990
1   Mckinney, Riley and Day                      Finland     2015
2               Hester Ltd                        China     1971
3           Holder-Sellers                   Turkmenistan     2004
4              Mayer Group                     Mauritius     1991
..                     …                            …      …
95             Holmes Group                     Ethiopia     1975
96                Good Ltd                     Anguilla     1971
97       Clements-Espinoza  Falkland Islands (Malvinas)     1991
98              Mendez Inc              Kyrgyz Republic     1993
99          Watkins-Kaiser                         Togo     2009

                     Industry  Number_of_employees
0                    Plastics                3498
1   Glass / Ceramics / Concrete              4952
2                Public Safety                5287
3                   Automotive                 921
4               Transportation                7870
..                         …                   …
95                Photography                2988
96          Consumer Services                4292
97            Broadcast Media                 236
98       Education Management                 339
99         Financial Services                2785

[100 rows x 5 columns]
```

El siguiente código hace la lectura de un archivo sin encabezados, los cuales son asignandos mediante una lista que se pasas al parámetro *names=*.

[5]:
```
encabezados = ['nombre', 'pais', 'anio_de_fundacion', 'tipo_de_industria',␣
 ↪'numero_de_empleados']
df_encabezado = pd.read_csv('../data/organizations_no_header.csv',␣
 ↪names=encabezados)
df_encabezado
```

[5]:
```
                    nombre                       pais  anio_de_fundacion  \
0              Ferrell LLC             Papua New Guinea               1990
1   Mckinney, Riley and Day                      Finland               2015
2               Hester Ltd                        China               1971
3           Holder-Sellers                   Turkmenistan               2004
4              Mayer Group                     Mauritius               1991
..                     …                            …                  …
95             Holmes Group                     Ethiopia               1975
```

```
96                Good Ltd                      Anguilla              1971
97       Clements-Espinoza  Falkland Islands (Malvinas)              1991
98             Mendez Inc              Kyrgyz Republic              1993
99          Watkins-Kaiser                         Togo              2009

              tipo_de_industria  numero_de_empleados
0                       Plastics                 3498
1   Glass / Ceramics / Concrete                 4952
2                 Public Safety                 5287
3                    Automotive                  921
4                Transportation                 7870
..                           …                    …
95                 Photography                 2988
96            Consumer Services                 4292
97              Broadcast Media                  236
98          Education Management                  339
99            Financial Services                 2785

[100 rows x 5 columns]
```

1.4.2 XLSX (Excel)

Para leer datos desde un archivo Microsoft Excel existe la función **pandas.read_excel(, sheet_name=)**, en la cual se pasa el argumento de ubicación del archivo y el nombre de la hoja con los datos, cabe mencionar que también se debe instalar la libreria *openpyxl* para la lectura de este tipo de archivos.

[6]:
```
df_excel = pd.read_excel('../data/organizations.xlsx', sheet_name='Hoja_1')
df_excel
```

[6]:
```
                     Name                      Country  Founded  \
0              Ferrell LLC            Papua New Guinea     1990
1   Mckinney, Riley and Day                     Finland     2015
2               Hester Ltd                       China     1971
3           Holder-Sellers                  Turkmenistan     2004
4             Mayer Group                     Mauritius     1991
..                      …                           …        …
95            Holmes Group                     Ethiopia     1975
96              Good Ltd                      Anguilla     1971
97     Clements-Espinoza  Falkland Islands (Malvinas)     1991
98           Mendez Inc              Kyrgyz Republic     1993
99        Watkins-Kaiser                         Togo     2009

                     Industry  Number_of_employees
0                     Plastics                 3498
1   Glass / Ceramics / Concrete                4952
2                 Public Safety                5287
3                    Automotive                 921
```

```
4                Transportation                 7870
..                          …                  …
95                  Photography                 2988
96             Consumer Services                4292
97               Broadcast Media                 236
98          Education Management                 339
99            Financial Services                2785

[100 rows x 5 columns]
```

1.4.3 TXT

Para concluir esta sección se leerá un archivo con extensión .txt o archivo de texto plano, cuya separación entre los datos es el **pipe (|)**, para la lectura también se hace uso de la función **pandas.read_csv()**, pues ambos son archivos del mismo tipo, la diferencia principal es la indicación en el parámetro de separación: **sep='|'**.

[7]:
```
df_txt = pd.read_csv('../data/organizations.txt', sep="|")
df_txt
```

[7]:
```
                    Name             Country  Founded  \
0             Ferrell-LLC    Papua New Guinea     1990
1  Mckinney-Riley and Day             Finland     2015
2             Hester-Ltd               China     1971
3         Holder-Sellers        Turkmenistan     2004
4            Mayer-Group           Mauritius     1991
5         Henry-Thompson             Bahamas     1992
6         Hansen-Everett            Pakistan     2018
7          Mcintosh-Mora             Islands     1970
8               Carr-Inc              Kuwait     1996
9      Kidd-Group-Bouvet  Island-(Bouvetoya)     2001

                     Industry  Number_of_employees
0                     Plastics                3498
1      Glass-Ceramics-Concrete                4952
2                Public Safety                5287
3                   Automotive                 921
4               Transportation                7870
5  Primary-Secondary-Education                4914
6          Publishing-Industry                7832
7                Import-Export                4389
8                     Plastics                8167
9  Primary-Secondary-Education                7473
```

1.5 Exploración de los datos

Después de leer los datos lo común es hacer una exploración de los mismos contenidos en el DataFrame, a continuación, se ejemplifican algunas de las funciones esenciales para esta tarea.

- **DataFrame.shape**, retorna una tupla que contiene el número de filas y columnas.

[8]:
```
df_shape = df.shape
print("Número de filas y columnas: ", df_shape)
```

```
Número de filas y columnas:  (100, 5)
```

- **DataFrame.head(n)**, función que retorna las primeras n filas pasadas como parámetro, por defecto muestras las 5 primeras.

[9]:
```
# mostrar los primeros 6 registros
df.head(6)
```

[9]:

	Name	Country	Founded	Industry	Number_of_employees
0	Ferrell LLC	Papua New Guinea	1990	Plastics	3498
1	Mckinney, Riley and Day	Finland	2015	Glass / Ceramics / Concrete	4952
2	Hester Ltd	China	1971	Public Safety	5287
3	Holder-Sellers	Turkmenistan	2004	Automotive	921
4	Mayer Group	Mauritius	1991	Transportation	7870
5	Henry-Thompson	Bahamas	1992	Primary / Secondary Education	4914

- **DataFrame.tail(n)**, función que retorna las últimas n filas, como en *head()*, si no se le pasa el valor de n mostrará los últimos 5 registros.

[10]:
```
# últimas 6 filas
df.tail(6)
```

[10]:

	Name	Country	Founded	Industry	Number_of_employees
94	Best, Wade and Shepard	Zimbabwe	1991	Gambling / Casinos	4873
95	Holmes Group	Ethiopia	1975	Photography	2988
96	Good Ltd	Anguilla	1971	Consumer Services	4292
97	Clements-Espinoza	Falkland Islands (Malvinas)	1991	Broadcast Media	236
98	Mendez Inc	Kyrgyz Republic	1993	Education Management	339
99	Watkins-Kaiser	Togo	2009		

```
99    Financial Services                2785
```

- **DataFrame.info()** función que imprime la general información del DataFrame como son: rango de índice, las columnas, los valores nulos, los tipos de datos, entre otros.

[11]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 100 entries, 0 to 99
Data columns (total 5 columns):
 #   Column               Non-Null Count  Dtype 
---  ------               --------------  ----- 
 0   Name                 100 non-null    object
 1   Country              100 non-null    object
 2   Founded              100 non-null    int64 
 3   Industry             100 non-null    object
 4   Number_of_employees  100 non-null    int64 
dtypes: int64(2), object(3)
memory usage: 4.0+ KB
```

- **DataFrame.describe()**, retorna por defecto las estadísticas descriptivas de las columnas númericas, si queremos incluir a las variables categóricas o no númericas, se pasa el parámetro *include='all'*.

[12]:
```
df.describe()
```

[12]:

	Founded	Number_of_employees
count	100.000000	100.000000
mean	1995.410000	4964.860000
std	15.744228	2850.859799
min	1970.000000	236.000000
25%	1983.500000	2741.250000
50%	1995.000000	4941.500000
75%	2010.250000	7558.000000
max	2021.000000	9995.000000

[13]:
```
df.describe(include='all')
```

[13]:

	Name	Country	Founded	Industry	Number_of_employees
count	100	100	100.000000	100	100.000000
unique	100	84	NaN	72	NaN
top	Ferrell LLC	Benin	NaN	Plastics	NaN
freq	1	3	NaN	4	NaN
mean	NaN	NaN	1995.410000	NaN	4964.860000
std	NaN	NaN	15.744228	NaN	2850.859799
min	NaN	NaN	1970.000000	NaN	236.000000
25%	NaN	NaN	1983.500000	NaN	2741.250000
50%	NaN	NaN	1995.000000	NaN	4941.500000
75%	NaN	NaN	2010.250000	NaN	7558.000000

```
max              NaN      NaN  2021.000000        NaN          9995.000000
```

Para aplicar la función anterior a una sola columna númerica se pueden aplicar las siguientes opciones:

[14]:
```
df.Number_of_employees.describe()
```

[14]:
```
count     100.000000
mean     4964.860000
std      2850.859799
min       236.000000
25%      2741.250000
50%      4941.500000
75%      7558.000000
max      9995.000000
Name: Number_of_employees, dtype: float64
```

[15]:
```
df['Number_of_employees'].describe()
```

[15]:
```
count     100.000000
mean     4964.860000
std      2850.859799
min       236.000000
25%      2741.250000
50%      4941.500000
75%      7558.000000
max      9995.000000
Name: Number_of_employees, dtype: float64
```

- **DataFrame.dtypes** esta función devuelve los tipos de datos del DataFrame:

[16]:
```
df.dtypes
```

[16]:
```
Name                 object
Country              object
Founded               int64
Industry             object
Number_of_employees   int64
dtype: object
```

- **DataFrame.columns**, esta función retorna los nombres de las columnas del DataFrame:

[17]:
```
print('Nombres de las columnas en el DataFrame:', df.columns)
```

```
Nombres de las columnas en el DataFrame: Index(['Name', 'Country', 'Founded',
'Industry', 'Number_of_employees'], dtype='object')
```

1.6 Selección de datos

1.6.1 Selección de datos para un subconjunto de columnas

Para seleccionar un subconjunto de columnas usamos la siguiente anotación **DataFrame[[col1, col2,...]]**, esto retorna un DataFrame con las columnas seleccionadas.

[18]:
```
print("Selección de datos de las columnas nombre y país:")
nuevo_df = df[['Name', 'Country']]
nuevo_df.head()
```

```
Selección de datos de las columnas nombre y país:
```

[18]:
```
                      Name           Country
0              Ferrell LLC  Papua New Guinea
1  Mckinney, Riley and Day           Finland
2               Hester Ltd             China
3           Holder-Sellers      Turkmenistan
4              Mayer Group         Mauritius
```

1.6.2 Selección de datos usando una condición simple

Es similar a realizar una selección en una tabla de SQL con la clausula **where**, en este caso seleccionaremos los registros donde el país sea igual a Suecia (Sweden).

[19]:
```
suecia_df = df[ df.Country == 'Sweden']
suecia_df
```

[19]:
```
                      Name Country  Founded                     Industry  \
14            Pacheco-Spears  Sweden     1984                     Maritime
43                Prince PLC  Sweden     2016  Individual / Family Services
84  George, Russo and Guerra  Sweden     1989             Military Industry

    Number_of_employees
14                  769
43                 7645
84                 2880
```

Otro ejemplo sería la seleccion de las organizaciones que tienen un número de empleados mayor a 9,000:

[20]:
```
org_num_emp_mayor_9mil = df[ df.Number_of_employees > 9000][['Name']] #␣
 ↪columna(s) a seleccionar
org_num_emp_mayor_9mil
```

[20]:
```
                      Name
9                Gaines Inc
11              Crane-Clarke
13               Glover-Pope
37               Cuevas-Moss
```

```
40                  Arroyo Inc
47  Wallace, Madden and Morris
49                Gonzales Ltd
61                  Soto Group
63                   Riley Ltd
```

[21]:
```
org_y_num_emp_mayor_9mil = df[ df.Number_of_employees > 9000][['Name',␣
↪'Number_of_employees']] # columna(s) a seleccionar
org_y_num_emp_mayor_9mil
```

[21]:
```
                         Name  Number_of_employees
9                  Gaines Inc                 9698
11               Crane-Clarke                 9011
13                Glover-Pope                 9079
37                Cuevas-Moss                 9995
40                  Arroyo Inc                9067
47  Wallace, Madden and Morris                9443
49                Gonzales Ltd                9069
61                  Soto Group                9097
63                   Riley Ltd                9315
```

1.6.3 Selección de datos con múltiples condiciones

Algunas ocasiones basta solo seleccionar con una condición simple, sin embargo, en el trabajo del día a día lo común es realizar selecciones con más de una condición, para este ejemplo, se necesita seleccionar las organizaciones de la industria de los plasticos con más de 6,000 empleados:

- **Condición 1 y Condición 2**
- **Condición 1:** Industry igual a Plastics
- **Condición 2:** Number_of_employees mayor a 6,000

En este caso, ambas condiciones se deben cumplir, para esto usamos el condicional *and* (*&*).

[22]:
```
df_ind_plast_emp_mayor_6mil = df[ (df.Industry == 'Plastics') & (df.
↪Number_of_employees > 6000)]
df_ind_plast_emp_mayor_6mil
```

[22]:
```
          Name           Country  Founded  Industry  Number_of_employees
8     Carr Inc            Kuwait     1996  Plastics                 8167
57  Cherry PLC  Marshall Islands     1980  Plastics                 8245
```

El siguiente ejemplo selecciona datos cuando se cumple una condicción o la otra, es decir, no necesariamente deben cumplirse ambas como en el caso anterior.

- **Conmdición 1 o Condición 2**

Lo anterior se logra mediante el condicional *Or* (|) :

[23]:
```
df_ind_plast_or_emp_mayor_6mil = df[ (df.Industry == 'Plastics') | (df.
↪Number_of_employees > 6000)]
df_ind_plast_or_emp_mayor_6mil.head(10) # se reduce solo a diez registros para␣
↪ser más legible ya que son 42 totales
```

[23]:

	Name	Country	Founded	\
0	Ferrell LLC	Papua New Guinea	1990	
4	Mayer Group	Mauritius	1991	
6	Hansen-Everett	Pakistan	2018	
8	Carr Inc	Kuwait	1996	
9	Gaines Inc	Uzbekistan	1997	
10	Kidd Group	Bouvet Island (Bouvetoya)	2001	
11	Crane-Clarke	Denmark	2014	
13	Glover-Pope	United Arab Emirates	2013	
15	Hodge-Ayers	Honduras	1990	
16	Bowers, Guerra and Krause	Uganda	1972	

	Industry	Number_of_employees
0	Plastics	3498
4	Transportation	7870
6	Publishing Industry	7832
8	Plastics	8167
9	Outsourcing / Offshoring	9698
10	Primary / Secondary Education	7473
11	Food / Beverages	9011
13	Medical Practice	9079
15	Facilities Services	8508
16	Primary / Secondary Education	6986

1.7 Limpieza de datos

Para ofrecer resultados precisos y consistentes resulta fundamental poseer datos libres de impurezas, es decir, no tener valores duplicados, valores perdidos, datos irrelevantes o redundantes, entre otros. Para solventar este problema es necesario manipular y corregir estas inconsistencias. En pandas se tienen varios métodos y/o funciones para limpieza, a continuación se presentan algunas esenciales para esta tarea.

1.7.1 Registros duplicados

Los valores o registros duplicados en los datos es un problema prevalente, ya que consumen mucho espacio de almacenamiento y en memoria, lo que resulta en una inexactitud de conocimiento u objetivos del negocio. Lo mejor, es remover estos registros, pero para esto, es importante identificarlos y analizar los diferentes caminos para hacerlo. En pandas para el tratamioento del problema de registros duplicados cuenta con las funciones **DataFrame.duplicated()** y **DataFrame.drop_duplicates()**.

Se consideran datos duplicados cuando en un registro tienen el mismo valor en todas las columnas, para este ejercicio nos apoyaremos del siguiente archivo:

[24]:
```
# carga de archivo csv con duplicados
df_dup = pd.read_csv('../data/organizations_duplicados.csv')
df_dup
```

[24]:

	Id_org	Name	Country	Founded	Industry	Number_of_employees
0	100	Ferrell LLC	Papua New Guinea	1990	Plastics	3498
1	101	Mckinney, Riley and Day	Finland	2015	Glass / Ceramics / Concrete	4952
2	101	Mckinney, Riley and Day	Finland	2015	Glass / Ceramics / Concrete	4952
3	102	Hester Ltd	China	1971	Public Safety	5287
4	103	Holder-Sellers	Turkmenistan	2004	Automotive	921
5	104	Mayer Group	Mauritius	1991	Transportation	7870
6	104	Mayer Group	Mauritius	1991	Transportation	7870
7	106	Henry-Thompson	Bahamas	1992	Plastics	7870
8	107	Henry-Thompson	Bahamas	1992	Primary / Secondary Education	4914
9	108	Hansen-Everett	Pakistan	2018	Publishing Industry	7832
10	108	Hansen-Everett	Pakistan	2018	Publishing Industry	7832
11	109	Mcintosh-Mora	Heard Island and McDonald Islands	1970	Import / Export	4389

Observamos que las filas con los **Id_org = [101, 104, 108]** son registros duplicados. Es muy importante analizar si un registro en verdad es duplicado y, sí es necesario eliminarlo, veamos los registros con **Id_org = [106, 107]**, se observa que hay coincidencias de valores en algunas columnas, pero esto no significa que necesariamente sea un registro duplicado, al final, será criterio del analista eliminar o no registros.

Para conocer el número de registros duplicados podemos escribir el siguiente código haciendo uso de las funciones **duplicated()** y **sum()**.

[25]:
```
num_reg_dup = df_dup.duplicated().sum()
print("Número total de registros duplicados:", num_reg_dup)
```

```
Número total de registros duplicados: 3
```

Para esta tarea también podemos ayudarnos de la columna *Id_org*, ya que por su misma naturaleza se entiende que no debe duplicarse ningún valor.

[26]:
```
# se asignan los valores duplicados al objeto declarado
num_reg_dup_id_org = df_dup[ df_dup.duplicated('Id_org')]
num_reg_dup_id_org
```

[26]:

	Id_org	Name	Country	Founded	\
2	101	Mckinney, Riley and Day	Finland	2015	
6	104	Mayer Group	Mauritius	1991	
10	108	Hansen-Everett	Pakistan	2018	

	Industry	Number_of_employees
2	Glass / Ceramics / Concrete	4952
6	Transportation	7870
10	Publishing Industry	7832

Para eliminar los registros duplicados, pandas proporciona la función **drop_duplicates()**, si pasamos una única columna como argumento elimina solamente los valores duplicados en ella, pero si pasamos una lista de columnas eliminará los valores duplicados en todas.

El código siguiente muestra la eliminación de registros duplicados para una columna en especifico (Id_org) y el sigueinte para varias columnas pasadas como una lista ([Name, Country, Founded]), así como la suma de registros duplicados después de ejecutar el código correspondiente.

[27]:
```
df_dup_drop_id = df_dup.drop_duplicates('Id_org')
df_dup_drop_id
```

[27]:

	Id_org	Name	Country	\
0	100	Ferrell LLC	Papua New Guinea	
1	101	Mckinney, Riley and Day	Finland	
3	102	Hester Ltd	China	
4	103	Holder-Sellers	Turkmenistan	
5	104	Mayer Group	Mauritius	
7	106	Henry-Thompson	Bahamas	
8	107	Henry-Thompson	Bahamas	
9	108	Hansen-Everett	Pakistan	
11	109	Mcintosh-Mora	Heard Island and McDonald Islands	

	Founded	Industry	Number_of_employees
0	1990	Plastics	3498
1	2015	Glass / Ceramics / Concrete	4952
3	1971	Public Safety	5287
4	2004	Automotive	921
5	1991	Transportation	7870
7	1992	Plastics	7870
8	1992	Primary / Secondary Education	4914
9	2018	Publishing Industry	7832
11	1970	Import / Export	4389

[28]:
```
num_reg_df_dup_drop_id = df_dup_drop_id.duplicated().sum()
print("Número de registros duplicados:", num_reg_df_dup_drop_id)
```

```
Número de registros duplicados: 0
```

[29]:
```
df_dup_drop_colum = df_dup.drop_duplicates(['Name', 'Country', 'Founded'])
df_dup_drop_colum
```

[29]:

	Id_org	Name	Country	Founded	Industry	Number_of_employees
0	100	Ferrell LLC	Papua New Guinea	1990	Plastics	3498
1	101	Mckinney, Riley and Day	Finland	2015	Glass / Ceramics / Concrete	4952
3	102	Hester Ltd	China	1971	Public Safety	5287
4	103	Holder-Sellers	Turkmenistan	2004	Automotive	921
5	104	Mayer Group	Mauritius	1991	Transportation	7870
7	106	Henry-Thompson	Bahamas	1992	Plastics	7870
9	108	Hansen-Everett	Pakistan	2018	Publishing Industry	7832
11	109	Mcintosh-Mora	Heard Island and McDonald Islands	1970	Import / Export	4389

[30]:
```
num_reg_df_dup_drop_column = df_dup_drop_colum.duplicated().sum()
print("Número de registros duplicados:", num_reg_df_dup_drop_column)
```

```
Número de registros duplicados: 0
```

1.7.2 Valores perdidos

Manipular valores perdidos es una tarea crucial, en ocasiones el enfoque más directo es eliminar los registros con los valores faltantes. Sin embargo, esto no funciona en todos los casos, pues podemos perder hechos importantes de los datos. Otra manera de manejar esta situación es imputar los valores perdidos, sin embargo, no existe un método único para encontrar éstos valores. Realizar la imputación dependerá de varios factores, como son el tipo de problema, la necesidad del negocio, afectaciones a los resultados esperados, entre otros más.

En pandas se tiene una lista importante de funciones para manipular los datos faltantes o perdidos, algunos de ellas se ven a continuación.

- **Eliminar filas con valores perdidos**

Se tiene la función **pandas.dropna()** para eliminar los registros con valores perdidos, esta función recibe como parámetros *how=all* y *how=any*, para el primer caso, se requiere que todas las columnas de la fila presenten valores perdidos (NaN-Not a Number), para el segundo basta que

alguna de las columnas exista un valor faltante para que sea eliminada la fila. A parte de los parámetros anteriores, también puede recibir el parámetro *subset=[col1, col2,...]*, el cual recibe una lista específica de columnas donde se desea aplicar la eliminación.

Para mostrar los ejemplos de esta función se usará el archivo "organizations_mv.csv".

[31]:
```
df_na = pd.read_csv('../data/organizations_mv.csv')
df_na
```

[31]:

	Name	Country	Founded	Industry	Number_of_employees
0	Ferrell LLC	Papua New Guinea	1990.0	Plastics	3498.0
1	Mckinney, Riley and Day	Finland	2015.0	Glass / Ceramics / Concrete	NaN
2	NaN	Japan	2015.0	Glass / Ceramics / Concrete	4952.0
3	Hester Ltd	China	1971.0	Public Safety	NaN
4	Holder-Sellers	Turkmenistan	2004.0	Automotive	921.0
5	NaN	NaN	NaN	NaN	NaN
6	Mayer Group	Mauritius	1991.0	Transportation	7870.0
7	NaN	NaN	NaN	NaN	NaN
8	Henry-Thompson	Bahamas	1992.0	Primary / Secondary Education	4914.0
9	NaN	NaN	2018.0	NaN	0.0
10	Mcintosh-Mora	Heard Island and McDonald Islands	1970.0	Import / Export	4389.0

[32]:
```
# borrado de registros donde todas sus columnas presenten valores perdidos␣
↪(índices 5 y 7)
df_na.dropna(how='all')
```

[32]:

	Name	Country	Founded
0	Ferrell LLC	Papua New Guinea	1990.0
1	Mckinney, Riley and Day	Finland	2015.0
2	NaN	Japan	2015.0
3	Hester Ltd	China	1971.0
4	Holder-Sellers	Turkmenistan	2004.0
6	Mayer Group	Mauritius	1991.0
8	Henry-Thompson	Bahamas	1992.0
9	NaN	NaN	2018.0

```
10           Mcintosh-Mora  Heard Island and McDonald Islands   1970.0

                        Industry  Number_of_employees
0                        Plastics               3498.0
1     Glass / Ceramics / Concrete                  NaN
2     Glass / Ceramics / Concrete               4952.0
3                  Public Safety                  NaN
4                     Automotive                921.0
6                 Transportation               7870.0
8   Primary / Secondary Education               4914.0
9                             NaN                  0.0
10               Import / Export               4389.0
```

[33]:
```
# borrado de registros donde alguna de sus columnas presenten valores perdidos␣
↪(índices 1, 2, 3, 5, 7 y 9)
df_na.dropna(how='any')
```

[33]:
```
             Name                          Country  Founded  \
0      Ferrell LLC                Papua New Guinea   1990.0
4   Holder-Sellers                     Turkmenistan   2004.0
6      Mayer Group                        Mauritius   1991.0
8   Henry-Thompson                          Bahamas   1992.0
10   Mcintosh-Mora  Heard Island and McDonald Islands   1970.0

                        Industry  Number_of_employees
0                        Plastics               3498.0
4                     Automotive                921.0
6                 Transportation               7870.0
8   Primary / Secondary Education               4914.0
10               Import / Export               4389.0
```

[34]:
```
# borrado de registros donde las columnas especifícadas presentan valores␣
↪perdidos (índices 2, 5, 7 y 9)
df_na.dropna(subset=['Name', 'Industry'])
```

[34]:
```
                     Name                          Country  Founded  \
0              Ferrell LLC                Papua New Guinea   1990.0
1   Mckinney, Riley and Day                         Finland   2015.0
3               Hester Ltd                           China   1971.0
4            Holder-Sellers                     Turkmenistan   2004.0
6               Mayer Group                        Mauritius   1991.0
8            Henry-Thompson                          Bahamas   1992.0
10            Mcintosh-Mora  Heard Island and McDonald Islands   1970.0

                        Industry  Number_of_employees
0                        Plastics               3498.0
1     Glass / Ceramics / Concrete                  NaN
```

```
3                    Public Safety                    NaN
4                       Automotive                  921.0
6                   Transportation                 7870.0
8   Primary / Secondary Education                  4914.0
10                  Import / Export                4389.0
```

- **Llenado o imputación de valores perdidos**

Como se comentó no siempre lo mejor es eliminar los registros con valores perdidos, ya que en algunas ocasiones surge la necesidad de inferir los valores para imputarlos. Para realizar esta acción pandas posee las funciones **fillna()** y **replace()**, las cuales pueden ser usadas con varios parámetros dependiendo las necesidades; a continuación, se craerá un DataFrame a partir de un diccionario para ejemplificar los métodos mencionados.

[35]:
```
import numpy as np # libreria especializada en el cálculo numérico
# np.nan significa insertar valores NaN (Not a Number) o valores perdidos
org_dict = {
    'Name': ['Abraham', 'James', 'NA', 'Victoria', 'Hinako'],
    'Department': ['Adversiting', np.nan, 'Accounting', '', 'Accounting'],
    'Salary_USD': [7920, 8500, np.nan, 7450, 0],
    'City': ['Madrid', 'Chicago', np.nan, 'Cali', 'Tokio']
    }

df_imp = pd.DataFrame(org_dict)
df_imp
```

[35]:
```
       Name   Department  Salary_USD     City
0   Abraham  Adversiting      7920.0   Madrid
1     James          NaN      8500.0  Chicago
2        NA   Accounting         NaN      NaN
3  Victoria                   7450.0     Cali
4    Hinako   Accounting         0.0    Tokio
```

Nota: En un DataFrame los valores NaN (Not a Number) significan que son valores perdidos, sin embargo, en pandas los valores de cero, espacios en blanco y NA no son considerados valores faltantes.

- **DataFrame.fillna()**

En seguida se hará uso del método **fillna()**, el cual recibe como parámetro el valor con el que se imputa el o los valores perdidos, en este caso un string o cadena de texto *'Valor Faltante'*.

[36]:
```
df_imp_fillna = df_imp.fillna('Valor Faltante')
df_imp_fillna
```

[36]:
```
       Name      Department      Salary_USD           City
0   Abraham     Adversiting          7920.0         Madrid
1     James  Valor Faltante          8500.0        Chicago
2        NA      Accounting  Valor Faltante  Valor Faltante
3  Victoria                          7450.0           Cali
```

```
4    Hinako      Accounting             0.0           Tokio
```

El método **ffill()** (forward fill) hace un llenado hacia adelante, es decir, el valor perdido se sustituye con el valor previo de la misma columna.

[37]:
```
df_imp_fillna_ffill = df_imp.ffill()
df_imp_fillna_ffill
```

[37]:

	Name	Department	Salary_USD	City
0	Abraham	Adversiting	7920.0	Madrid
1	James	Adversiting	8500.0	Chicago
2	NA	Accounting	8500.0	Chicago
3	Victoria		7450.0	Cali
4	Hinako	Accounting	0.0	Tokio

El método **bfill()** (backward fill) hace una imputación hacia atrás, es decir, el valor perdido se sustituye con el valor de adelante de la misma columna.

[38]:
```
df_imp_fillna_bfill = df_imp.bfill()
df_imp_fillna_bfill
```

[38]:

	Name	Department	Salary_USD	City
0	Abraham	Adversiting	7920.0	Madrid
1	James	Accounting	8500.0	Chicago
2	NA	Accounting	7450.0	Cali
3	Victoria		7450.0	Cali
4	Hinako	Accounting	0.0	Tokio

Regresando con la función **fillna()**, ahora se sustituirán los valores perdidos con algún valor estadístico de la propia columna númerica, como son la media (*mean()*), mediana (*median()*) y moda (*mode()*). Para ejemplificar estas funciones imputaremos el valor perdido de la columna *Salary_USD*.

[39]:
```
df_imp.describe().T # .T transpone los datos
```

[39]:

	count	mean	std	min	25%	50%	75%	max
Salary_USD	4.0	5967.5	4001.444531	0.0	5587.5	7685.0	8065.0	8500.0

[40]:
```
salary_mean = df_imp['Salary_USD'].fillna(df_imp['Salary_USD'].mean())
df_imp_fillna_salary_mean = pd.DataFrame(salary_mean)
df_imp_fillna_salary_mean
```

[40]:

	Salary_USD
0	7920.0
1	8500.0
2	5967.5
3	7450.0
4	0.0

[41]:
```
salary_median = df_imp['Salary_USD'].fillna(df_imp['Salary_USD'].median())
df_imp_fillna_salary_median = pd.DataFrame(salary_median)
df_imp_fillna_salary_median
```

[41]:

	Salary_USD
0	7920.0
1	8500.0
2	7685.0
3	7450.0
4	0.0

En el siguiente ejemplo se imputarán valores perdidos con la moda, pero será para una columna categórica, es importante pasar el parámetro que indica que la imputación será por filas [0].

[42]:
```
department_mode = df_imp['Department'].fillna(df_imp.Department.mode()[0]) #␣
 ↪cero hace referencia a los valores por fila
df_imp_fillna_department = pd.DataFrame(department_mode)
df_imp_fillna_department
```

[42]:

	Department
0	Adversiting
1	Accounting
2	Accounting
3	
4	Accounting

- **DataFrame.replace()**

Como se observó con las funciones **fillna()**, **ffill()** y **bfill()** sólo los valores NaN son imputados, pero no llena valores NA y espacios en blanco, para esta situación se puede usar la función **replace()**, a la cual se pasa como parámetros el valor actual y el valor nuevo, obteniendo un remplazo de estos valores.

A continuación, veremos algunos aplicaciones de la función **replace()**.

[43]:
```
# seguiremos haciendo uso del DataFrame df_imp
df_imp
```

[43]:

	Name	Department	Salary_USD	City
0	Abraham	Adversiting	7920.0	Madrid
1	James	NaN	8500.0	Chicago
2	NA	Accounting	NaN	NaN
3	Victoria		7450.0	Cali
4	Hinako	Accounting	0.0	Tokio

[44]:
```
# se define un diccionario con los valores a reemplazar, la clave o key es el␣
 ↪valor actual y value el valor de reemplazo
df_replace = df_imp.replace({'NA':'Not applicable', 0:5500, "":'Blank', np.nan:
 ↪'Not a Number'})
```

```
df_replace
```

[44]:

	Name	Department	Salary_USD	City
0	Abraham	Adversiting	7920.0	Madrid
1	James	Not a Number	8500.0	Chicago
2	Not applicable	Accounting	Not a Number	Not a Number
3	Victoria	Blank	7450.0	Cali
4	Hinako	Accounting	5500.0	Tokio

[45]:
```
# para una columna especifíca
salary_replace = pd.Series(df_imp['Salary_USD'].replace(
                                                        {0:5000, # valor cero␣
 ↪es reemplazado por 5,000
                                                         np.nan:
 ↪df_imp['Salary_USD'].mean() # valor NaN es reemplazado por la media
                                                        })
                           )
salary_replace
```

[45]:
```
0    7920.0
1    8500.0
2    5967.5
3    7450.0
4    5000.0
Name: Salary_USD, dtype: float64
```

1.8 Agrupar y agregar

Cuando se necesita obtener un resumen de los datos se puede hacer uso de las funciones de agrupación y agregación. Para agrupar pandas posee la función **groupby()** y para la agregación tiene varias funciones, como son: **min()**, **max()**, **count()**, **sum()**, entre otras.

1.8.1 Agrupar

Para conocer el uso de la función **groupby()** se hará uso del archivo correspondiente al Censo Poblacional 2020 del Instituto Nacional de Estadística y Geografía (INEGI) para la Ciudad de México, cabe mencionar que este solo contiene una parte del total de todas las columnas.

A la función **groupby()** se le pasará el argumento 'NOM_MUN', el cual corresponde al nombre de las alcaldías, para fines prácticos se hará referencia a éstas como municipios que es como se les conoce a estas delimitaciones territoriales en la mayor parte del país. El objeto o variable que contiene la agrupación define a los nombres de los municipios como keys o llaves.

[46]:
```
df_censo_cdmx = pd.read_csv('../data/censo_2020_cdmx_pob.csv')
df_censo_cdmx
```

[46]:

	ENTIDAD	NOM_ENT	MUN	NOM_MUN	LOC	\
0	9	Ciudad de México	2	Azcapotzalco	1	
1	9	Ciudad de México	3	Coyoacán	1	

```
2          9  Ciudad de México    4  Cuajimalpa de Morelos     1
3          9  Ciudad de México    4  Cuajimalpa de Morelos    10
4          9  Ciudad de México    4  Cuajimalpa de Morelos    20
..       …                 …  …                   …   …
501        9  Ciudad de México   13             Xochimilco  9999
502        9  Ciudad de México   14          Benito Juárez     1
503        9  Ciudad de México   15             Cuauhtémoc     1
504        9  Ciudad de México   16         Miguel Hidalgo     1
505        9  Ciudad de México   17    Venustiano Carranza     1

                         NOM_LOC          LONGITUD          LATITUD  ALTITUD  \
0                    Azcapotzalco   99°11'03.698" W  19°29'02.770" N   2244.0
1                       Coyoacán   99°09'43.724" W  19°21'00.770" N   2247.0
2             Cuajimalpa de Morelos   99°17'59.252" W  19°21'26.461" N   2780.0
3                     Cruz Blanca   99°19'26.437" W  19°19'04.026" N   2982.0
4              San Lorenzo Acopilco   99°19'32.454" W  19°19'51.617" N   2936.0
..                              …                …                …      …
501  Localidades de dos viviendas              NaN              NaN      NaN
502                 Benito Juárez   99°09'28.272" W  19°22'19.172" N   2238.0
503                    Cuauhtémoc   99°09'06.781" W  19°26'29.928" N   2230.0
504                Miguel Hidalgo   99°11'26.716" W  19°24'26.168" N   2264.0
505             Venustiano Carranza   99°06'49.324" W  19°25'09.343" N   2230.0

     POBTOT  …  P_15A17  P_15A17_F  P_15A17_M  P_18A24  P_18A24_F  \
0    432205  …    16351       7969       8382    43088      21303
1    614447  …    22618      11108      11510    63817      31569
2    186693  …     8344       4128       4216    21403      11039
3       728  …       32         16         16       95         43
4     26042  …     1257        624        633     3137       1557
..      …  …      …         …         …       …        …
501      53  …        1          1          0        4          2
502  434153  …    10787       5423       5364    33294      16744
503  545884  …    18325       9006       9319    51762      25874
504  414470  …    13093       6426       6667    36545      18619
505  443704  …    17632       8809       8823    45461      22446

     P_18A24_M  P_15A49_F  P_60YMAS  P_60YMAS_F  P_60YMAS_M
0        21785     115600     78650       45565       33085
1        32248     162387    126592       73584       53008
2        10364      54258     22800       12635       10165
3           52        196        54          28          26
4         1580       7238      2658        1457        1201
..         …         …         …           …           …
501          2         12         7           3           4
502      16550     123446     87344       52615       34729
503      25888     152077     93809       55146       38663
504      17926     115822     71111       41605       29506
```

```
505       23015      117331      78964        45947        33017

[506 rows x 52 columns]
```

[47]:
```
# crear el agrupamiento
df_censo_cdmx_agrup_nom_mun = df_censo_cdmx.groupby('NOM_MUN')
print("Objeto agrupado: ", df_censo_cdmx_agrup_nom_mun)
```

```
Objeto agrupado:  <pandas.core.groupby.generic.DataFrameGroupBy object at
0x0000024FB0F9DB50>
```

[48]:
```
# obtener los valores agrupados o keys del objeto agrupado
grupos_mun = df_censo_cdmx_agrup_nom_mun.groups.keys()
print("Valores agrupados:\n", grupos_mun)
```

```
Valores agrupados:
 dict_keys(['Azcapotzalco', 'Benito Juárez', 'Coyoacán', 'Cuajimalpa de
Morelos', 'Cuauhtémoc', 'Gustavo A. Madero', 'Iztacalco', 'Iztapalapa', 'La
Magdalena Contreras', 'Miguel Hidalgo', 'Milpa Alta', 'Tlalpan', 'Tláhuac',
'Venustiano Carranza', 'Xochimilco', 'Álvaro Obregón'])
```

Si se requiere agrupar por más de una columna se puede usar la función *groupby()* con una lista de los nombres de las columnas a agrupar como argumento: **df.groupby(['col1, col2, ..., coln'])**.

1.8.2 Agregar

Después de mostrar el uso de la función de agrupar, lo siguiente es hacer uso de las funciones de agregación, la síntaxis puede verse así: **df.groupby([col1, col2...]).función_de_agregación()**. Un ejemplo del uso de esta sintaxis puede ser el obtener la población total para cada municipio. Lo primero es agrupar por la columna de municipio (NUM_MUN) y aplicar la función de agregación de suma [*sum()*] para la columna población total (POBTOT).

[49]:
```
df_censo_cdmx.groupby(['NOM_MUN'])[['POBTOT']].sum()
```

[49]:

NOM_MUN	POBTOT
Azcapotzalco	432205
Benito Juárez	434153
Coyoacán	614447
Cuajimalpa de Morelos	217686
Cuauhtémoc	545884
Gustavo A. Madero	1173351
Iztacalco	404695
Iztapalapa	1835486
La Magdalena Contreras	247622
Miguel Hidalgo	414470
Milpa Alta	152685
Tlalpan	699928
Tláhuac	392313

```
Venustiano Carranza      443704
Xochimilco               442178
Álvaro Obregón           759133
```

Existe otra forma de usar las funciones de agregación, para esto es necesario es hacer uso de otra función, siendo la siguiente: **agg([función_de_agregación1(), función_de_agregación2()...])**, las funciones de agregación se pueden pasar a la función **agg()** como una lista o un diccionario. En el siguiente ejemplo retomamos la agrupación de los municipios, las funciones de agregación aplicaran a la columna de población total, siendo: valor mínimo y máximo, suma, conteo, media y desviación estándar, para esta última se obtienen valores NaN por ser un solo ser un registro desagregado.

[50]:
```
df_censo_cdmx_agg_pob = df_censo_cdmx.groupby(['NOM_MUN'])['POBTOT'].
↪agg(['min', 'sum', 'mean', 'std', 'max', 'count'])
df_censo_cdmx_agg_pob
```

[50]:

NOM_MUN	min	sum	mean	std	\
Azcapotzalco	432205	432205	4.322050e+05	NaN	
Benito Juárez	434153	434153	4.341530e+05	NaN	
Coyoacán	614447	614447	6.144470e+05	NaN	
Cuajimalpa de Morelos	21	217686	1.145716e+04	42844.607399	
Cuauhtémoc	545884	545884	5.458840e+05	NaN	
Gustavo A. Madero	1173351	1173351	1.173351e+06	NaN	
Iztacalco	404695	404695	4.046950e+05	NaN	
Iztapalapa	1835486	1835486	1.835486e+06	NaN	
La Magdalena Contreras	10	247622	1.768729e+04	65836.241721	
Miguel Hidalgo	414470	414470	4.144700e+05	NaN	
Milpa Alta	6	152685	7.101628e+02	3113.709893	
Tlalpan	5	699928	7.290917e+03	61381.526750	
Tláhuac	13	392313	8.173188e+03	46761.749042	
Venustiano Carranza	443704	443704	4.437040e+05	NaN	
Xochimilco	6	442178	4.466444e+03	43151.722365	
Álvaro Obregón	3	759133	1.265222e+05	309851.063712	

NOM_MUN	max	count
Azcapotzalco	432205	1
Benito Juárez	434153	1
Coyoacán	614447	1
Cuajimalpa de Morelos	186693	19
Cuauhtémoc	545884	1
Gustavo A. Madero	1173351	1
Iztacalco	404695	1
Iztapalapa	1835486	1
La Magdalena Contreras	246428	14
Miguel Hidalgo	414470	1
Milpa Alta	27768	215

```
Tlalpan                  599935     96
Tláhuac                  323771     48
Venustiano Carranza      443704      1
Xochimilco               429481     99
Álvaro Obregón           759003      6
```

1.9 Ordenar y clasificar

En algunas ocasiones se necesita ordenar, así como clasificar los datos. Por ejemplo, se necesita ordenar la población de las localidades del archivo del censo de forma ascendente, o bien, clasificar los datos y generar marcadores de esta clasificación. Por supuesto, pandas provee funciones para estas tareas.

1.9.1 Ordenar

Pandas posee las funciones **sort_index()** y **sort_values()** para ordenar los datos. La función **sort_index()** ordenará los datos de acuerdo con los valores del índice, mientras que **sort_values()** ordenara con base en los valores de una columna en específico, por defecto, estas funciones ordenan de forma ascendente.

En el siguiente ejemplo se ordenan los datos usando la función **sort_index()**, los índices se asignarán con los valores 1, 4, 2, 5, 3 pasados como una lista en el parámetro **index=**, después se ordenan con la función **sort_index()**, obteniendo el orden 1, 2, 3, 4, 5.

[51]:
```
alc_pobtot_dict = {
    'id_mun': [1, 3, 8, 11, 15],
    'municipio': ['Azcapotzalco', 'Coyoacán', 'Iztapalapa', 'Milpa Alta',␣
 ↪'Xochimilco'],
    'pobtot': [432205, 614447, 1835486, 152685, 442178]
}

df_alc = pd.DataFrame(alc_pobtot_dict, index=[1, 4, 2, 5, 3])

print("DataFrame sin ordenar:\n",df_alc)
print("\n==================================\n")
df_alc_sort_idx = df_alc.sort_index()
print("DataFrame ordenado por índice:\n", df_alc_sort_idx)
```

```
DataFrame sin ordenar:
    id_mun     municipio   pobtot
1       1  Azcapotzalco   432205
4       3      Coyoacán   614447
2       8    Iztapalapa  1835486
5      11    Milpa Alta   152685
3      15    Xochimilco   442178

==================================

DataFrame ordenado por índice:
```

```
    id_mun     municipio   pobtot
1        1  Azcapotzalco   432205
2        8    Iztapalapa  1835486
3       15    Xochimilco   442178
4        3      Coyoacán   614447
5       11    Milpa Alta   152685
```

Ahora corresponde hacer uso de la función **sort_values()**, como ya se mencionó ordena por valores de una columna, en este caso se ordena por la columna de población (pobtot), hay que recordar que ordena por defecto de forma ascendente.

[52]:
```
df_alc_sort_pobtot = df_alc.sort_values(['pobtot'])
print("DataFrame ordenado a partir de los valores de una columna:")
df_alc_sort_pobtot
```

```
DataFrame ordenado a partir de los valores de una columna:
```

[52]:
```
    id_mun     municipio   pobtot
5       11    Milpa Alta   152685
1        1  Azcapotzalco   432205
3       15    Xochimilco   442178
4        3      Coyoacán   614447
2        8    Iztapalapa  1835486
```

Si se desea ordenar de forma descendente es necesario pasar el argumento **ascending=False**, en el siguiente ejemplo se ordena por la columna "id_mun" de forma descendente.

[53]:
```
df_alc_sort_id_mun_desc = df_alc.sort_values(['id_mun'], ascending=False)
print("DataFrame ordenado por id_mun de manera descendente: ")
df_alc_sort_id_mun_desc
```

```
DataFrame ordenado por id_mun de manera descendente:
```

[53]:
```
    id_mun     municipio   pobtot
3       15    Xochimilco   442178
5       11    Milpa Alta   152685
2        8    Iztapalapa  1835486
4        3      Coyoacán   614447
1        1  Azcapotzalco   432205
```

1.9.2 Clasificar

Para realizar una clasificación se tiene la función **rank()**, esta función asigna una clasificación de valores en una columna nueva iniciando con el número 1. Asimismo, la función **rank()** posee las opciones de ordenar de forma ascendente y descendente con el mismo argumento. Para estos ejemplos se clasifica de acuerdo con la población en forma ascendente y descendente.

[54]:
```
df_alc['pobtot_rank_asc'] = df_alc['pobtot'].rank()
df_alc
```

[54]:

	id_mun	municipio	pobtot	pobtot_rank_asc
1	1	Azcapotzalco	432205	2.0
4	3	Coyoacán	614447	4.0
2	8	Iztapalapa	1835486	5.0
5	11	Milpa Alta	152685	1.0
3	15	Xochimilco	442178	3.0

[55]:
```
df_alc['pobtot_rank_desc'] = df_alc['pobtot'].rank(ascending=False)
df_alc
```

[55]:

	id_mun	municipio	pobtot	pobtot_rank_asc	pobtot_rank_desc
1	1	Azcapotzalco	432205	2.0	4.0
4	3	Coyoacán	614447	4.0	2.0
2	8	Iztapalapa	1835486	5.0	1.0
5	11	Milpa Alta	152685	1.0	5.0
3	15	Xochimilco	442178	3.0	3.0

1.10 Agregar filas

Con las funciones **.loc[]** y **len()** se puede agregar una nueva fila a un DataFrame existente. Para este ejemplo el DataFrame será el que se creó a partir de las algunas alcaldías y su población, la fila que se agrega se construye a partir de un diccionario.

[56]:
```
df_alc_org = df_alc.iloc[:,:3] # se seleccionan todas las filas y las primeras␣
 ↪3 columnas
df_alc_org
```

[56]:

	id_mun	municipio	pobtot
1	1	Azcapotzalco	432205
4	3	Coyoacán	614447
2	8	Iztapalapa	1835486
5	11	Milpa Alta	152685
3	15	Xochimilco	442178

[57]:
```
nueva_fila = {
    'id_mun':12,
    'municipio':'Tlalpan',
    'pobtot':699928
    }
print(type(nueva_fila))

df_alc_org.loc[len(df_alc_org)] = nueva_fila # localizamos el último registro␣
 ↪mediante la longitud del DataFrame y se agrega la nueva fila
df_alc_org.sort_index()
```

```
<class 'dict'>
```

[57]:

	id_mun	municipio	pobtot
1	1	Azcapotzalco	432205
2	8	Iztapalapa	1835486
3	15	Xochimilco	442178
4	3	Coyoacán	614447
5	12	Tlalpan	699928

1.11 Agregar columnas

Se puede agregar una nueva columna a un DataFrame existente de las siguientes maneras:

- Agregar una nueva columna con un valor constante

[58]:
```
df_alc_org['valor_constante'] = 'Nuevo valor'
df_alc_org
```

[58]:

	id_mun	municipio	pobtot	valor_constante
1	1	Azcapotzalco	432205	Nuevo valor
4	3	Coyoacán	614447	Nuevo valor
2	8	Iztapalapa	1835486	Nuevo valor
5	12	Tlalpan	699928	Nuevo valor
3	15	Xochimilco	442178	Nuevo valor

- Agregar una nueva columna con valores de una lista

[59]:
```
df_alc_org['pob_fem'] = [227255, 325337, 947835, 365051, 398130] # lista de la␣
↪población femenina
df_alc_org
```

[59]:

	id_mun	municipio	pobtot	valor_constante	pob_fem
1	1	Azcapotzalco	432205	Nuevo valor	227255
4	3	Coyoacán	614447	Nuevo valor	325337
2	8	Iztapalapa	1835486	Nuevo valor	947835
5	12	Tlalpan	699928	Nuevo valor	365051
3	15	Xochimilco	442178	Nuevo valor	398130

- Agregar una nueva columna mediante la aplicación de una transformación lógica

Es recurrente en el análisis de datos la necesidad de agregar una columna con nuevos valores a partir de alguna función de transformación en un DataFrame existente. Para este ejemplo se va a suponer que se tiene la necesidad de incrementar en 10% el total de la población como factor de demanda en un estudio de movilidad, para esto se hace uso de la función **apply()**.

La función **apply()** pernmite hacer uso una función personalizada y aplicar la misma en las columnas del DataFrame o en una columna en específico.

Lo primero que se hace es escribir la función para incrementar en 10% la población, posteriormente, se emplea la función **apply()** a la cual le pasa la función que asignará los nuevos valores con incremento de 10%.

[60]:
```
# función personalizada
def inc_pob_10_porc(pob_tot):
    return int(pob_tot * 1.1) # asignamos la salida en tipo entero para no␣
 ↪tener números decimales

# agregar la nueva columna con los valores incrementados en 10%
df_alc_org['pob_tot_inc_10_porc'] = df_alc_org['pobtot'].apply(inc_pob_10_porc)
df_alc_org
```

[60]:

	id_mun	municipio	pobtot	valor_constante	pob_fem	pob_tot_inc_10_porc
1	1	Azcapotzalco	432205	Nuevo valor	227255	475425
4	3	Coyoacán	614447	Nuevo valor	325337	675891
2	8	Iztapalapa	1835486	Nuevo valor	947835	2019034
5	12	Tlalpan	699928	Nuevo valor	365051	769920
3	15	Xochimilco	442178	Nuevo valor	398130	486395

También se puede hacer uso de la función **lambda()** en lugar de una función regular de Python, un ejemplo de este tipo de función para un objetivo similar es el siguiente:

[61]:
```
# agregaremos una columna con valores de incremento en 12%
df_alc_org['pob_tot_inc_12_porc'] = df_alc_org['pobtot'].apply(lambda pob:␣
 ↪int(pob * 1.12))
df_alc_org
```

[61]:

	id_mun	municipio	pobtot	valor_constante	pob_fem	\
1	1	Azcapotzalco	432205	Nuevo valor	227255	
4	3	Coyoacán	614447	Nuevo valor	325337	
2	8	Iztapalapa	1835486	Nuevo valor	947835	
5	12	Tlalpan	699928	Nuevo valor	365051	
3	15	Xochimilco	442178	Nuevo valor	398130	

	pob_tot_inc_10_porc	pob_tot_inc_12_porc
1	475425	484069
4	675891	688180
2	2019034	2055744
5	769920	783919
3	486395	495239

1.12 Eliminar filas y/o columnas

Para eliminar filas o columnas pandas provee la función **drop()**. Para su uso es necesario pasar los parámetros de índice (para un solo registro o un rango) e información del eje (axis=0 para filas y axis=1 para columnas).

El primer ejemplo consiste en eliminar la fila con índice=3 (Xochimilco)

[62]:
```
df_alc_drop_fila = df_alc_org.drop([3], axis=0)
df_alc_drop_fila
```

[62]:
```
   id_mun     municipio   pobtot valor_constante  pob_fem  \
1       1  Azcapotzalco   432205     Nuevo valor   227255
4       3      Coyoacán   614447     Nuevo valor   325337
2       8    Iztapalapa  1835486     Nuevo valor   947835
5      12       Tlalpan   699928     Nuevo valor   365051

   pob_tot_inc_10_porc  pob_tot_inc_12_porc
1               475425               484069
4               675891               688180
2              2019034              2055744
5               769920               783919
```

Para eliminar columnas se pasa el nombre o lista de nombres de las columnas a eliminar y la información del eje (axis=1) como parámetros a la función **drop()**. Sin embargo, se tiene otra opción para esto, se puede hacer uso del parámetro **inplace=True**, esto significa que la función eliminará la columna en el DataFrame existente.

[63]:
```
df_alc_org.drop(['pob_tot_inc_10_porc'], axis=1, inplace=True)
df_alc_org
```

[63]:
```
   id_mun     municipio   pobtot valor_constante  pob_fem  pob_tot_inc_12_porc
1       1  Azcapotzalco   432205     Nuevo valor   227255               484069
4       3      Coyoacán   614447     Nuevo valor   325337               688180
2       8    Iztapalapa  1835486     Nuevo valor   947835              2055744
5      12       Tlalpan   699928     Nuevo valor   365051               783919
3      15    Xochimilco   442178     Nuevo valor   398130               495239
```

1.13 Concatenar DataFrames

Para concatenar DataFrames se tiene la función **concat()**, los parámetros mínimos que se deben pasar son los DataFrames a unir y la información del eje, es importante mencionar que por defecto el eje es igual a cero (*axis=0*), es decir, si no se específica pandas intentará anexar el segundo DataFrame en la parte inferior del primero. Para el ejemplo de esta función se crearán dos DataFrames a partir de los datos del archivo del Censo, los DataFrames tendrán las columnas de identificador o clave de municipio, nombre de municipio y el total de la población femenina.

[64]:
```
df_1 = pd.DataFrame({'id_mun':[2, 3, 4, 5],
                     'nom_mun':['Azcapotzalco', 'Coyoacán', 'Cuajimalpa de␣
 ↪Morelos', 'Gustavo A. Madero'],
                     'pob_fem':[227255, 325337, 113537, 609477]}
                    )

df_2 = pd.DataFrame({'id_mun':[6, 7, 8, 9],
                     'nom_mun':['Iztacalco', 'Iztapalapa', 'La Magdalena␣
 ↪Contreras', 'Milpa Alta'],
                     'pob_fem':[212343, 947835, 129335, 78314]}
                    )
print(df_1)
```

```
print("=========================================")
print(df_2)
```

```
   id_mun                nom_mun  pob_fem
0       2           Azcapotzalco   227255
1       3               Coyoacán   325337
2       4  Cuajimalpa de Morelos   113537
3       5      Gustavo A. Madero   609477
=========================================
   id_mun                  nom_mun  pob_fem
0       6               Iztacalco   212343
1       7              Iztapalapa   947835
2       8  La Magdalena Contreras   129335
3       9              Milpa Alta    78314
```

[65]:
```
df_concat = pd.concat([df_1, df_2])
df_concat
```

[65]:
```
   id_mun                  nom_mun  pob_fem
0       2           Azcapotzalco   227255
1       3               Coyoacán   325337
2       4  Cuajimalpa de Morelos   113537
3       5      Gustavo A. Madero   609477
0       6               Iztacalco   212343
1       7              Iztapalapa   947835
2       8  La Magdalena Contreras   129335
3       9              Milpa Alta    78314
```

Se observa que efectivamente se anexo el segundo DataFrame al primero en la parte inferior al especificar el eje. Si se pone atención se observa que los valores del índice se mantienen de origen en el DataFrame de salida, para solucionar esta situación se puede hacer uso del parámetro **ignore_index=True**.

[66]:
```
df_concat_ign_idx = pd.concat([df_1, df_2], ignore_index=True)
df_concat_ign_idx
```

[66]:
```
   id_mun                  nom_mun  pob_fem
0       2           Azcapotzalco   227255
1       3               Coyoacán   325337
2       4  Cuajimalpa de Morelos   113537
3       5      Gustavo A. Madero   609477
4       6               Iztacalco   212343
5       7              Iztapalapa   947835
6       8  La Magdalena Contreras   129335
7       9              Milpa Alta    78314
```

Ahora se hará uso de la función **concat()** con el parámetro del eje igual a 1 (axis=1), es decir, se unirá por el eje de columna, para esto, se crea un DataFrame con el total de la población masculina correspondiente a los municipios del df_1.

[67]:
```
df_3 = pd.DataFrame({'id_mun':[2, 3, 4, 5],
                     'pob_fem':[204950, 289110, 104149, 563874]}
                    )
df_3
```

[67]:

	id_mun	pob_fem
0	2	204950
1	3	289110
2	4	104149
3	5	563874

[68]:
```
df_concat_axis_1 = pd.concat([df_1, df_3], axis=1)
df_concat_axis_1
```

[68]:

	id_mun	nom_mun	pob_fem	id_mun	pob_fem
0	2	Azcapotzalco	227255	2	204950
1	3	Coyoacán	325337	3	289110
2	4	Cuajimalpa de Morelos	113537	4	104149
3	5	Gustavo A. Madero	609477	5	563874

1.14 Fusionar o unir DataFrames

Pandas proporciona dos funciones muy completas para fusionar (**merge()**) o unir(**join()**) DataFrames, estas funciones tiene características similares a la función **join** de SQL. En esta sección se trabajará con ambas funciones haciendo uso de sus parámetros fundamentales.

Para iniciar con la tarea se construyen los DataFrames de municipios (mun_df) y localidades (loc_df) con los algunos datos censales.

[69]:
```
mun_df = pd.DataFrame({'id_mun':[8, 9, 4, 13, 12],
                     'nom_mun':['La Magdalena Contreras', 'Milpa Alta',␣
 ↪'Cuajimalpa de Morelos', 'Xochimilco', 'Tlalpan']}
                     )
mun_df
```

[69]:

	id_mun	nom_mun
0	8	La Magdalena Contreras
1	9	Milpa Alta
2	4	Cuajimalpa de Morelos
3	13	Xochimilco
4	12	Tlalpan

[70]:
```
loc_df = pd.DataFrame({'id_mun':[8, 9, 4, 13, 11],
                     'nom_loc':['Rancho Viejo', 'San Antonio Tecómitl', 'Cruz␣
 ↪Blanca', 'Paraje Zacapa', 'San Andrés Mixquic']}
                    )
loc_df
```

[70]:
	id_mun	nom_loc
0	8	Rancho Viejo
1	9	San Antonio Tecómitl
2	4	Cruz Blanca
3	13	Paraje Zacapa
4	11	San Andrés Mixquic

1.14.1 Fusionar

Básicamente la función **merge()** toma dos DataFrames como parámetros de entrada y retorna un DataFrame fusionado como salida. La sintaxis de la función **merge()** y algunos de los parámetros esenciales son:

pandas.merge(left, right, how='inner', on=None, left_on=None, right_on=None)

- **left:** DataFrame izquierdo.
- **right:** DataFrame derecho.
- **how:** Aquí se indica cómo se desea fusionar los DataFrames, similar a SQL: ['inner', 'left', 'right', 'outer', 'cross'], por defecto se aplica 'inner'.
- **on:** La columna de identificador o índices que deben estar presentes en ambos DataFrames.
- **left_on:** Columna de identificador o índices con el cual se debe fusionar el DataFrame de la izquierda.
- **right_on:** Columna de identificador o índices con el cual se debe fusionar el DataFrame de la derecha.

A continuación se presentan algunos ejemplos de la función **merge()** usando los parámetros antes descritos para su mejor comprensión.

- Cuando ambos DataFrames poseen el mismo nombre de la columna de identificador o índice

Se observa que los DataFrames de mun_df y loc_df poseen una columna en común (id_mun), esta posee los índices con los cuales se van a fusionar los registros que tienen en común el valor de dicha columna, el nombre de esta columna se pasa en el parámetro 'on' (**on='id_mun'**), asimismo, para este primer ejemplo en el parámetro 'how' se definirá con 'inner'(**how='inner'**), esto significa que solo se fusionarán los registros con coincidencias o intersección en ambos DataFrames.

[71]:
```
df_merge_inner = pd.merge(mun_df, loc_df, on='id_mun', how='inner')
df_merge_inner
```

[71]:
	id_mun	nom_mun	nom_loc
0	8	La Magdalena Contreras	Rancho Viejo
1	9	Milpa Alta	San Antonio Tecómitl
2	4	Cuajimalpa de Morelos	Cruz Blanca
3	13	Xochimilco	Paraje Zacapa

Para el siguiente ejemplo en el parametro 'how' se asigna el string 'left'(**how='left'**), es decir, fusionará aquellos registros donde solo exista coincidencia con los del DataFrame izquierdo. Considere que en caso de no encontrar valores coincidentes en las columnas anexadas se asignara valores NaN.

[72]:
```
df_merge_left = pd.merge(mun_df, loc_df, on='id_mun', how='left')
df_merge_left
```

[72]:

	id_mun	nom_mun	nom_loc
0	8	La Magdalena Contreras	Rancho Viejo
1	9	Milpa Alta	San Antonio Tecómitl
2	4	Cuajimalpa de Morelos	Cruz Blanca
3	13	Xochimilco	Paraje Zacapa
4	12	Tlalpan	NaN

En el siguiente se asigna 'right' en el parametro 'how' (**how='right'**), lo que significa que fusionará aquellos registros donde solo exista coincidencia con los del DataFrame derecho. De la misma manera, de no encontar valores existentes en las columnas que se devuelven en el DataFrame fusionado se asignara valores NaN.

[73]:
```
df_merge_right = pd.merge(mun_df, loc_df, on='id_mun', how='right')
df_merge_right
```

[73]:

	id_mun	nom_mun	nom_loc
0	8	La Magdalena Contreras	Rancho Viejo
1	9	Milpa Alta	San Antonio Tecómitl
2	4	Cuajimalpa de Morelos	Cruz Blanca
3	13	Xochimilco	Paraje Zacapa
4	11	NaN	San Andrés Mixquic

Ahora en el parametro 'how' se asigna 'outer'(**how='outer'**), con lo cual se obtienen absolutamente todos registros exista o no coincidencia de los valores de la columna del identificador.

[74]:
```
df_merge_outer = pd.merge(mun_df, loc_df, on='id_mun', how='outer')
df_merge_outer
```

[74]:

	id_mun	nom_mun	nom_loc
0	8	La Magdalena Contreras	Rancho Viejo
1	9	Milpa Alta	San Antonio Tecómitl
2	4	Cuajimalpa de Morelos	Cruz Blanca
3	13	Xochimilco	Paraje Zacapa
4	12	Tlalpan	NaN
5	11	NaN	San Andrés Mixquic

Con la asignación de 'cross' en el parámetro 'how' (**how='cross'**) se obtiene un producto cartesiano, en este caso no es necesario el uso del parámetro *on=*. Dicho de otra manera, el total de filas del DataFrame resultante es igual al número de registros del primer DataFrame multiplicado por el total registros del segundo DataFrame.

[75]:
```
df_merge_cross = pd.merge(mun_df, loc_df, how='cross')
df_merge_cross
```

[75]:

	id_mun_x	nom_mun	id_mun_y	nom_loc
0	8	La Magdalena Contreras	8	Rancho Viejo

```
1          8  La Magdalena Contreras         9  San Antonio Tecómitl
2          8  La Magdalena Contreras         4           Cruz Blanca
3          8  La Magdalena Contreras        13        Paraje Zacapa
4          8  La Magdalena Contreras        11    San Andrés Mixquic
5          9              Milpa Alta         8         Rancho Viejo
6          9              Milpa Alta         9  San Antonio Tecómitl
7          9              Milpa Alta         4           Cruz Blanca
8          9              Milpa Alta        13        Paraje Zacapa
9          9              Milpa Alta        11    San Andrés Mixquic
10         4   Cuajimalpa de Morelos         8         Rancho Viejo
11         4   Cuajimalpa de Morelos         9  San Antonio Tecómitl
12         4   Cuajimalpa de Morelos         4           Cruz Blanca
13         4   Cuajimalpa de Morelos        13        Paraje Zacapa
14         4   Cuajimalpa de Morelos        11    San Andrés Mixquic
15        13              Xochimilco         8         Rancho Viejo
16        13              Xochimilco         9  San Antonio Tecómitl
17        13              Xochimilco         4           Cruz Blanca
18        13              Xochimilco        13        Paraje Zacapa
19        13              Xochimilco        11    San Andrés Mixquic
20        12                 Tlalpan         8         Rancho Viejo
21        12                 Tlalpan         9  San Antonio Tecómitl
22        12                 Tlalpan         4           Cruz Blanca
23        12                 Tlalpan        13        Paraje Zacapa
24        12                 Tlalpan        11    San Andrés Mixquic
```

- Cuando los DataFrames no poseen el mismo nombre de la columna de valores a coincidir

Cuando los DataFrames a fusionar no poseen el mismo nombre en la columna con los valores coincidentes pandas posee los parámetros **left_on=None** y **right_on=None**, con estos prámetros se señalan los nombres de las columnas a intersectar. Para este ejemplo, renombraremos la columna 'id_mun' con el nombre de 'id', asimsimo, se asigna este DataFrame a una nueva variable.

[76]:
```
loc_df_id = pd.DataFrame({'id':[8, 9, 4, 13, 11],
                          'nom_loc':['Rancho Viejo', 'San Antonio Tecómitl', 'Cruz␣
 ↪Blanca', 'Paraje Zacapa', 'San Andrés Mixquic']}
                         )
loc_df_id
```

[76]:
```
   id              nom_loc
0   8         Rancho Viejo
1   9  San Antonio Tecómitl
2   4          Cruz Blanca
3  13        Paraje Zacapa
4  11    San Andrés Mixquic
```

[77]:
```
df_merge_right_on = pd.merge(mun_df, loc_df_id, left_on='id_mun',␣
 ↪right_on='id', how='inner')
df_merge_right_on
```

[77]:
```
   id_mun                 nom_mun  id              nom_loc
0       8  La Magdalena Contreras   8         Rancho Viejo
1       9             Milpa Alta   9  San Antonio Tecómitl
2       4   Cuajimalpa de Morelos   4           Cruz Blanca
3      13             Xochimilco  13        Paraje Zacapa
```

1.14.2 Unir

La función **join()** es usada para unir columnas de otro DataFrame, es similar a la función **merge()**, sin embargo, pero la función **join()** realiza la unión con base en los índices del DataFrame. La sintaxis así como parámetros más significativos de esta función son:

DataFrame.join(other, on=None, how='left', lsuffix='', rsuffix='', validate='str')

- **other:** Un DataFrame, serie o lista que contenga la combinación de estos.
- **on:** Columna de identificador que debe estar presentes en ambos DataFrames.
- **how:** Forma de unión de los DataFrames ['inner', 'left', 'right', 'outer', 'cross'], por defecto se aplica 'inner'.
- **lsuffix:** Sufijo para los nombres de las columnas superpuestas en el DataFrame izquierdo.
- **rsuffix:** Sufijo para los nombres de las columnas superpuestas en el DataFrame derecho.
- **validate:** Si se específica, valida que la unión sea del tipo específicado, es decir, comprueba la cardinalidad (1:1, 1:m, m:1, m:m).

Para conocer el uso de la función **join()** se hará uso de los DataFrames que se han utilizado, se asignaran los datos a nuevas variables con índices de una lista mediante el parámetro **index=**.

[78]:
```
mun_df_idx = pd.DataFrame({'id_mun':[8, 9, 4, 13, 12],
                    'nom_mun':['La Magdalena Contreras', 'Milpa Alta',␣
 ↪'Cuajimalpa de Morelos', 'Xochimilco', 'Tlalpan']},
                           index=[1, 2, 3, 4, 5]
                    )
mun_df_idx
```

[78]:
```
   id_mun                 nom_mun
1       8  La Magdalena Contreras
2       9             Milpa Alta
3       4   Cuajimalpa de Morelos
4      13             Xochimilco
5      12                Tlalpan
```

[79]:
```
loc_df_idx = pd.DataFrame({'id_mun':[8, 9, 4, 13, 11],
                    'nom_loc':['Rancho Viejo', 'San Antonio Tecómitl', 'Cruz␣
 ↪Blanca', 'Paraje Zacapa', 'San Andrés Mixquic']},
                           index=[0, 2, 3, 5, 6]
                   )
loc_df_idx
```

[79]:
```
   id_mun              nom_loc
0       8         Rancho Viejo
```

```
2       9  San Antonio Tecómitl
3       4           Cruz Blanca
5      13         Paraje Zacapa
6      11    San Andrés Mixquic
```

En el siguiente código se realiza la unión de los DataFrames anteriores con base en los índices asignados, el DataFrame de municipios (mun_df_idx) tomará lo posición de derecha y el de localidades (loc_df_idx) la izquierda, para identificar las columnas de coincidencia procedentes de cada DataFrame se hace uso de los parámetros **lsuffix='‘** y **rsuffix=’’**, a los cuales se pasa una cadena de texto, asimismo, se pasa al parámetro **how=’’** con el instring 'inner', para obtener una superposición solo de intersección de valores coincidentes de los índices. El resultado es un DataFrame solo con los índices de valores 2, 3 y 5, mismos que estan contenidos en ambos DataFrames y con el sufijo correspondiente en cada columna.

[80]:
```
joined_df_idx = mun_df_idx.join(loc_df_idx, lsuffix='_suf_izq',␣
 ↪rsuffix='_suf_der', how='inner')
joined_df_idx
```

[80]:
```
   id_mun_suf_izq               nom_mun  id_mun_suf_der               nom_loc
2               9            Milpa Alta               9  San Antonio Tecómitl
3               4  Cuajimalpa de Morelos               4           Cruz Blanca
5              12               Tlalpan              13         Paraje Zacapa
```

Para el siguinte ejemplo, los índices se definirán sobre una columna existente de los DataFrames, se observa que existe una columna en común (‘id_mun’) en ambos, para asignarla como índice se usará la función siguiente:

DataFrame.set_index()

Después de hacer esto se realiza la unión de los DataFrames.

[81]:
```
mun_df_idx_id_mun = mun_df.set_index('id_mun')
mun_df_idx_id_mun
```

[81]:
```
                       nom_mun
id_mun
8       La Magdalena Contreras
9                   Milpa Alta
4        Cuajimalpa de Morelos
13                  Xochimilco
12                     Tlalpan
```

[82]:
```
loc_df_idx_id_mun = loc_df.set_index('id_mun')
loc_df_idx_id_mun
```

[82]:
```
                     nom_loc
id_mun
8               Rancho Viejo
9       San Antonio Tecómitl
```

```
4                Cruz Blanca
13             Paraje Zacapa
11        San Andrés Mixquic
```

[83]:
```
joined_df_idx_id_mun = mun_df_idx_id_mun.join(loc_df_idx_id_mun, how='inner')
joined_df_idx_id_mun
```

[83]:
```
                        nom_mun               nom_loc
id_mun
8       La Magdalena Contreras          Rancho Viejo
9                   Milpa Alta  San Antonio Tecómitl
4        Cuajimalpa de Morelos           Cruz Blanca
13                  Xochimilco         Paraje Zacapa
```

Como se observa no existe una columna repetida en el DataFrame de salida, por lo que no es necesario usar los parámetros **lsuffix** y **rsuffix**.

1.15 Escritura de un DataFrame en archivos externos

Finalmente, después de hacer un pequeño recorrido por algunas funciones útiles de pandas para el análisis de datos, toca conocer como escribir los DataFrames obtenidos en archivos externos, tales como: CSV, Excel (xlsx), TXT y JSON, siendo estos unos de los más utilizados. Para realizar lo mencionadó se usará el DataFrame 'mun_df'.

[84]:
```
mun_df
```

[84]:
```
   id_mun                 nom_mun
0       8  La Magdalena Contreras
1       9              Milpa Alta
2       4   Cuajimalpa de Morelos
3      13              Xochimilco
4      12                 Tlalpan
```

- Escribir en un archivo CSV

Para escribir en un archivo CSV, pandas posee la función **to_csv()**, a la cual se le pasa la ruta y nombre con extensión donde se guardará el archivo, algunos otros prámetros son el tipo de separador (**sep='), la codificación (encoding="**) por defecto es utf-8, entre otros más.

[85]:
```
mun_df.to_csv('../data/municipios.csv')
```

Por defecto pandas genera automáticamente la columna de índices, para cambiar esto, es necesario pasar el parámetro 'index' igual a 'false' (**index=False**).

[86]:
```
mun_df.to_csv('../data/municipios_no_indice.csv', index=False)
```

- Escribir en un archivo TXT

Para escribir un archivo TXT, de la misma manera se usa la función **to_csv()**, pero es necesario cambiar la extensión, para este ejemplo se usará el parámetro de separación (**sep='**) asignado pipe ('|') como separador.

[87]:
```
mun_df.to_csv('../data/municipios_sep_pipe.txt', sep='|', index=False)
```

- Escribir en un archivo Excel

Para escribir en un archivo Excel(xlsx) se usa la función **to_excel()**, se pasa el parámetro de no generación de índices y, con el parámetro **sheet_name=''** se asigna el nombre a la hoja. Algunos otros parámetros importantes son: **startrow=int** y **startcol=int**, éstos definen a partir de qué fila y columna respectivamente inician los datos a escribir.

[88]:
```
mun_df.to_excel('../data/municipios_excel.xlsx', index=False,␣
↪sheet_name='first')
```

- Escribir en un archivo JSON

Para realizar la escritura en un archivo json pandas posee la función **to_json()**.

[89]:
```
mun_df.to_json('../data/municipios.json')
```