

**UNIVERSIDAD NACIONAL DE SAN AGUSTÍN DE AREQUIPA**  
**ESCUELA PROFESIONAL DE CIENCIA DE LA COMPUTACIÓN**

**CIENCIA DE DATOS**



**Data Wrangling**

**PRESENTADO POR:**

**John Edson Sanchez Chilo**

**AREQUIPA – PERÚ**  
**2025**

## Data Wrangling

### Descripción del dataset

#### Contexto

En este trabajo se analizarán dos fuentes de datos, ambas pertenecientes a Nueva Gales del Sur (NSW) en Australia. Los datos fueron obtenidos directamente de su página web <https://data.nsw.gov.au/>

El objetivo principal fue analizar la transmisión del virus a nivel comunitario en conjunción con factores sociodemográficos, para entender mejor los patrones de propagación, riesgos asociados y las intervenciones gubernamentales. La idea era explorar cómo los datos epidemiológicos, geo-localización, intervenciones y características sociales interactúan en la dinámica de la pandemia.

#### 1. Datos de casos de COVID-19 en NSW

El archivo contiene notificaciones de casos de COVID-19 en el estado de Nueva Gales del Sur en Australia, durante un período de aproximadamente dos años, desde el 25 de enero del 2020 al 07 de febrero de 2022. Cada registro representa una notificación sobre un nuevo caso en una fecha y localidad específica de NSW, ingresando el Local Health District (LHD) como región y adicionalmente el Local Government Area (LGA) como subdivisión.

Es importante aclarar que Australia se divide en estados, cada estado se subdivide en distritos de salud local (LHD), y cada distrito se divide en áreas de gobierno local (LGA). Los datos usados solo pertenecen al estado de Nueva Gales del Sur (NSW)

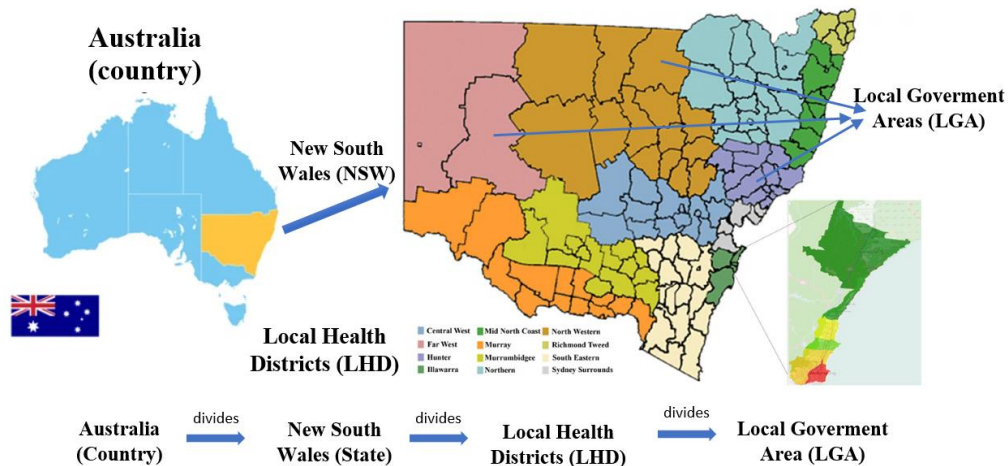


Figura 1. Mapa de subdivisiones de Australia

	notification_date	postcode	lhd_2010_code	lhd_2010_name	lga_code19	lga_name19
0	2020-01-25	2134.0	X700	Sydney	11300	Burwood (A)
1	2020-01-25	2121.0	X760	Northern Sydney	16260	Parramatta (C)
2	2020-01-25	2071.0	X760	Northern Sydney	14500	Ku-ring-gai (A)

Figura 2. Cada registro se lee como, “Caso de Covid-19 el día 2020-01-25 en el distrito de Sydney (x700), con código postal 2134, en el área de Burwood con código 11300”.

#### Definición de columnas

notification\_date: Fecha específica en la que se notificó sobre un caso de COVID-19

postcode: es el código postal que se utiliza para identificar el Local Health District

lhd\_2010\_code: Código utilizado para identificar al Local Health District (LHD)

lhd\_2010\_name: Nombre del Local Health District

lga\_code19: Código de identificación del Local Government Area (LGA)

lga\_name19: Nombre del LGA correspondiente

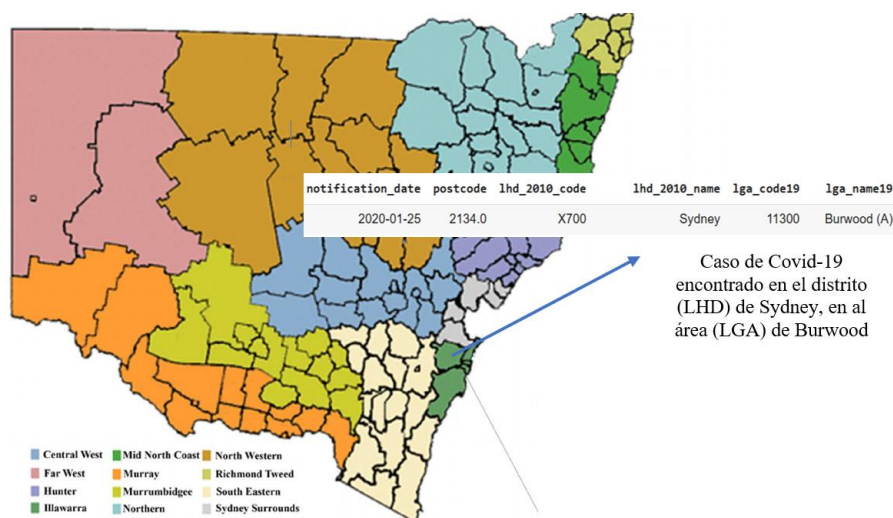


Figura 3. Representación de casos por Covid-19 por cada registro

Columna	notification_date	postcode	lga_code19	lhd_2010_code
Descripción	Representa la fecha en la que se encontro un caso de COVID-19	Codigo postal de cada Local Health District	Código del Local Government Area donde se detecto el caso	Código de 2010 que identifica el Local Health District
Tipo	Fecha	Número	Número	Texto
Formato/Medida	AA-MM-DD	Entero	Entero	Código Alfanumérico
Tipo de dato	Cuantitativo Discreto	Cualitativo Nominal	Cualitativo Nominal	Cualitativo Nominal
Mínimo	25/01/2020	2000	10050	X700
Máximo	7/02/2022	2990	18710	X999
Valores repetidos	Contiene una gran cantidad de fechas repetidas, especialmente en los ultimos meses	Aunque es numérico, no tiene valor matemático directo. Es un código geográfico.	Contiene una gran cantidad de fechas repetidas, especialmente en los ultimos meses	Contiene una gran cantidad de fechas repetidas, especialmente en los ultimos meses
Valores unicos	No existen, ya que todos se repiten	No existen, ya que todos se repiten	No existen, ya que todos se repiten	No existen, ya que todos se repiten
Rango de valores	25/01/2020 a 7/02/2022	Son 764 valores distintos	Son 128 valores distintos	Son 15 valores distintos

Tabla 1. Notificaciones de Covid-19 en NSW

## 2. Datos del censo de NSW

Datos sociodemográficos y económicos a nivel de Local Government Areas (LGAs), extraídos del censo de 2016, con la intención de analizar características comunitarias y posiblemente correlacionarlas con los patrones de contagio. La versión del censo de 2020 aún no estaba disponible en el momento de la investigación, por lo que se usaron los datos del censo de 2016.

	LGA_code	LGA_Name	LGA_Name_abbr	MedianAge	MedianMortgage	MedianPersonIncome	MedianRent	MedianFamilyIncome
0	LGA10050	Albury(C)	Albury	39	1421	642	231	1532
1	LGA10130	ArmidaleRegional(A)	Armidale.R.	36	1393	561	250	1465
2	LGA10250	Ballina(A)	Ballina	48	1733	601	340	1426
3	LGA10300	Balranald(A)	Balranald	41	950	624	150	1438
4	LGA10470	BathurstRegional(A)	Bathurst.R.	37	1670	646	280	1632

Figura 4. Tabla de datos sociodemográficos por LGA

De esta tabla se tiene un mayor número de columnas, por lo cual solo se colocará una descripción de los campos más resaltantes.

Columnas:

LGA\_code: Código único del Local Government Area

LGA\_code, LGA\_Name\_abbr: Nombre del LGA

MedianAge: Edad media. Tiene valor numérico y se puede promediar.

MedianMortgage: Representa la media de hipoteca a pagar

MedianRent: Representa la media de renta a pagar

MedianPersonIncome: Media de ingreso por familia

OneMethodbyBus: N° de personas que tienen como Modo único de transporte al Bus

LowIncome%: Porcentaje de personas de bajos ingresos y su proporción.

Otras columnas más.

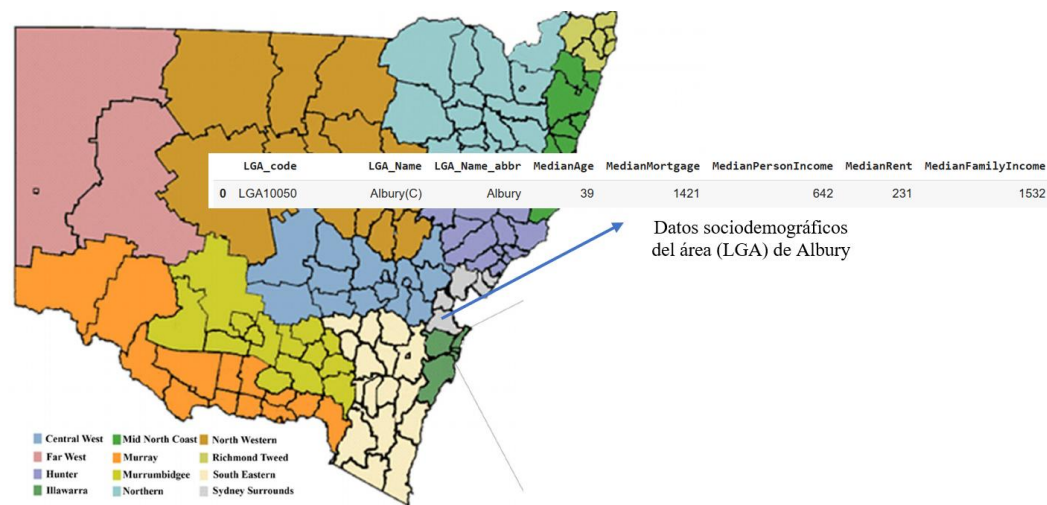


Figura 5. Representación de casos por Covid-19 por cada registro

Columna	LGA_code	LGA_code, LGA_Name_abbr	MedianAge	MedianMortgage	MedianRent	MedianPersonIncome	OneMethodbyBus	LowIncome%
Descripción	Código dñico del Local Government Area	Nombre del LGA	Edad media. Tiene valor numérico y se puede promediar.	Representa la media de hipoteca a pagar	Representa la media de renta a pagar	Media de ingreso por familia	N° de personas que tienen como Modo único de transporte al	Porcentaje de personas de bajos ingresos y su proporción.
Tipo	Número	Texto	Número	Número Entero	Número Entero	Número Entero	Número Entero	Número Decimal
Formato/Medida	Entero	Nombre	Entero	Dólar Australiano	Dólar Australiano	Dólar Australiano	Entero	Porcentaje
Tipo de dato	Cualitativo Nominal	Cualitativo Nominal	Cuantitativo discreto	Cuantitativo Discreto	Cuantitativo Discreto	Cuantitativo Discreto	Cuantitativo Discreto	Cuantitativo Continuo
Mínimo	10050	No tiene	32	433	90	439	0	0.184522
Máximo	18710	No tiene	54	3200	650	1386	1386	0.522246
Valores repetidos	Todos los valores son únicos	Todos los valores son únicos	40, 42, 36, 44, 45, ...	1300, 1083, 1733, 1517, ...	150, 200, 180, 250, ...	600, 538, 439, 524, ...	3, 8, 7, 10, ...	No hay valores repetidos
Valores unicos	Todos los valores son únicos	Todos los valores son únicos	54, 50, 52, 53	1421, 1393, 1670, 950, ...	245, 90, 105, 100, ...	719, 634, 688, 620, ...	54, 52, 127, ...	0.365752, 0.412482, ...
Rango de valores	Son 128 valores distintos	Son 128 valores distintos	32-54	433-3200	90-650	433-3200	433-3200	0.184522-0.522246

Tabla 2. Datos sociodemográficos por LGA de NSW

# Data Wrangling

## ✓ Analisis Exploratorio de Datos

## ✓ Análisis del comportamiento de los datos

### 1. Manipulación y transformación de datos del archivo

Se procedera con el análisis del archivo que contiene los datos a utilizar sobre las casos COVID por LGA y los datos sociodemográficos de cada LGA

- Formato del archivo: Casos Covid-19 El archivo se encuentra en formato CSV, es un formato ligero de intercambio de datos.

```
#casos_NSW.csv
notification_date,postcode,lhd_2010_code,lhd_2010_name,lga_code19,lga_name19
2020-01-25,2134,X700,Sydney,11300,Burwood (A)
2020-01-25,2121,X760,Northern Sydney,16260,Parramatta (C)
2020-01-25,2071,X760,Northern Sydney,14500,Ku-ring-gai (A)
```

- Formato del archivo: Datos sociodemográficos por LGA El archivo se encuentra en formato CSV, es un formato ligero de intercambio de datos, en el cual se separa la información por comas.

```
#20200221-Udate-LGA-NSW.csv
LGA_code,LGA_Name,LGA_Name_abbr,MedianAge,MedianMortgage,MedianPersonIncome,MedianRent, ...
LGA10050,Albury(C),Albury,39,1421,642,231,1532,0.8,1185,2.3,24662,26411,305.9,51076,45032,4635,2 ...
```

### • Encoding del archivo

Se revisa el encoding del archivo para saber en que formato se leerá posteriormente el archivo, en este caso el resultado del encoding es ascii, por lo cual encoding como UTF-8 también pueden ser usados como encoding para este archivo.

```
import chardet
with open("cases_NSW.csv", "rb") as f:
    result = chardet.detect(f.read(10000)) # lee los primeros 10,000 bytes
    encoding = result['encoding']
    print("Encoding detectado:", encoding)

with open("20200221-Udate-LGA-NSW.csv", "rb") as f:
    result = chardet.detect(f.read(10000)) # lee los primeros 10,000 bytes
    encoding = result['encoding']
    print("Encoding detectado:", encoding)
```

→ Encoding detectado: UTF-8-SIG  
Encoding detectado: ascii

### • Tamaño del archivo

El archivo sobre casos tiene un peso de 61.76MB, con lo cual se puede decir que es un archivo de tamaño mediano, y por lo tanto se trabajará con el de manera directa, es decir no hay necesidad de trabajarlo por partes.

El segundo archivo por otra parte, pesa solo 0.07MB, con lo cual se puede decir que es un archivo pequeño y se trabajará con el de forma directa

```
import os
file_size_mb = os.path.getsize("cases_NSW.csv") / (1024 * 1024)
print(f"Tamaño del archivo: {file_size_mb:.2f} MB")

file_size_mb = os.path.getsize("20200221-Udate-LGA-NSW.csv") / (1024 * 1024)
print(f"Tamaño del archivo: {file_size_mb:.2f} MB")
```

→ Tamaño del archivo: 61.76 MB  
Tamaño del archivo: 0.07 MB

### • Transformación de datos

En los archivos CSV, se presentan dos tipos de datos a nivel granular, el primero son los casos de COVID por semana vistos en cada LGA, y el segundo son los datos particulares y sociodemográficos de cada LGA Entonces se procedera a manejar dos tablas separadas que representaran nivel granular los datos sociodemográficos de cada LGA y los casos de cada LGA.

```
#Tabla 1: Casos COVID-19 de cada LGA
import pandas as pd
cases_df = pd.read_csv('cases_NSW.csv')
cases_df.head()
```

	notification_date	postcode	lhd_2010_code	lhd_2010_name	lga_code19	lga_name19
0	2020-01-25	2134.0	X700	Sydney	11300	Burwood (A)
1	2020-01-25	2121.0	X760	Northern Sydney	16260	Parramatta (C)
2	2020-01-25	2071.0	X760	Northern Sydney	14500	Ku-ring-gai (A)
3	2020-01-27	2033.0	X720	South Eastern Sydney	16550	Randwick (C)
4	2020-03-01	2077.0	X760	Northern Sydney	14000	Hornsby (A)

```
#Tabla 2: Casos por semana de cada LGA
socio_df = pd.read_csv('20200221-Uplate-LGA-NSW.csv')
socio_df.head()
```

	LGA_code	LGA_Name	LGA_Name_abbr	MedianAge	MedianMortgage	MedianPersonIncome	MedianRent	MedianFamilyIncome	Average
0	LGA10050	Albury(C)	Albury	39	1421	642	231	1532	
1	LGA10130	ArmidaleRegional(A)	Armidale.R.	36	1393	561	250	1465	
2	LGA10250	Ballina(A)	Ballina	48	1733	601	340	1426	
3	LGA10300	Balranald(A)	Balranald	41	950	624	150	1438	
4	LGA10470	BathurstRegional(A)	Bathurst.R.	37	1670	646	280	1632	

5 rows × 120 columns

## ✓ Preguntas:

### Un registro es una entidad, describa que representa un registro

En la primera tabla cada registro representa una notificación de un caso de COVID-19 en una localidad específica.

En la segunda tabla cada registro representa un LGA y sus diferentes datos sociodemográficos

### ¿Cuántos registros hay?

En la primera tabla de casos existen un total de 973412 registros.

En la segunda tabla de LGA's existen un total de 129 registros, esto debido a que aunque en NSW solo se considera 128 LGA's, NSW también contiene zonas no incorporadas que aunque están dentro de su jurisdicción no pertenecen a ningún LGA.

```
print("Tamaño de 'cases_df' (COVID):", cases_df.shape)
print("Tamaño de 'socio_df' (Sociodemográficos):", socio_df.shape)
```

```
Tamaño de 'cases_df' (COVID): (973412, 6)
Tamaño de 'socio_df' (Sociodemográficos): (129, 120)
```

### ¿Son demasiado pocos?

No, son una cantidad adecuada para poder hacer el análisis de los datos de forma correcta

### ¿Son muchos y no tenemos Capacidad (CPU+RAM) suficiente para procesarlo?

No, en el caso de la primera tabla de casos, esta ocupa un total de 316.9MB de memoria RAM por lo cual está dentro de las capacidades de Google Colab, y en el caso de la segunda tabla esta ocupa solo un 0.14MB de RAM, lo cual es bastante bajo, y por lo tanto se puede trabajar con ambas tablas sin complicaciones.

```
print(cases_df.memory_usage(deep=True))
print("Total de memoria de Casos:", cases_df.memory_usage(deep=True).sum() / 1024**2, "MB")
```

```
print(socio_df.memory_usage(deep=True))
print("Total de memoria de LGA's:", socio_df.memory_usage(deep=True).sum() / 1024**2, "MB")
```

```
Index      132
notification_date  65218604
postcode      7787296
lhd_2010_code  58970839
lhd_2010_name  70435990
lga_code19    59926019
lga_name19    69964888
dtype: int64
Total de memoria de Casos: 316.9095687866211 MB
```



```

Index          132
LGA_code       8385
LGA_Name       9040
LGA_Name_abbr  8451
MedianAge      1032
...
LowIncomePersons 1032
LowIncome%       1032
LonePerson       1032
LonePerson%      1032
PercentofPublicTransportation 1032
Length: 121, dtype: int64
Total de memoria de LGA's: 0.14667415618896484 MB

```

### ¿Hay datos duplicados?

Si en la primera tabla de casos, existen bastantes registros repetidos, lo cual es algo normal en esta tabla, ya que cada día en la misma localidad se puede detectar más de un caso de COVID-19.

No en la segunda tabla, ya que cada registro representa un LGA diferente, e incluso haciendo una búsqueda de repeticiones por código de LGA o nombre de LGA, no se encuentran repeticiones, lo cual es correcto.

```

duplicados_cases = cases_df.duplicated().sum()
duplicados_socio = socio_df.duplicated().sum()

print(f"Filas duplicadas en cases_df: {duplicados_cases}")
print(f"Filas duplicadas en socio_df: {duplicados_socio}")

```

➡ Filas duplicadas en cases\_df: 920552  
Filas duplicadas en socio\_df: 0

```
duplicados_cases_df = cases_df[cases_df.duplicated()]
```

```

# Mostrar las primeras 5 filas duplicadas
print("Filas duplicadas en 'cases_df':")
duplicados_cases_df.head()

```

➡ Filas duplicadas en 'cases\_df':

	notification_date	postcode	lhd_2010_code	lhd_2010_name	lga_code19	lga_name19
17	2020-03-04	2113.0	X760	Northern Sydney	16700	Ryde (C)
18	2020-03-04	2113.0	X760	Northern Sydney	16700	Ryde (C)
20	2020-03-04	2113.0	X760	Northern Sydney	16700	Ryde (C)
30	2020-03-06	2119.0	X760	Northern Sydney	14000	Hornsby (A)
40	2020-03-08	2116.0	X740	Western Sydney	16260	Parramatta (C)

```
conteo_LGA = socio_df['LGA_code'].value_counts()
```

```

# Filtrar los que aparecen más de una vez
duplicados_LGA = conteo_LGA[conteo_LGA > 1]
print("Valores de 'LGA_code' duplicados y su frecuencia:")
print(duplicados_LGA)
socio_dup_rows = socio_df[socio_df['LGA_code'].isin(duplicados_LGA.index)]
socio_dup_rows.head()

```

➡ Valores de 'LGA\_code' duplicados y su frecuencia:  
Series([], Name: count, dtype: int64)

LGA_code	LGA_Name	LGA_Name_abbr	MedianAge	MedianMortgage	MedianPersonIncome	MedianRent	MedianFamilyIncome	AverageBedroom
0 rows × 120 columns								

## 2. Tipo de Datos

Se procederá a revisar cada una de las columnas de ambas tablas para identificar sus tipo de datos, así mismo también se hará un análisis para encontrar valores máximos, mínimos, repetidos, nulo, etc.

### ¿Cuales son los tipos de datos de cada columnas?

Tabla de Casos de COVID-19:

- notification\_date: Fecha
- postcode: Número entero positivo
- lhd\_2010\_code: texto
- lhd\_2010\_name: texto
- lga\_code19: Número entero positivo



- lga\_name19: texto

Tabla de LGA's

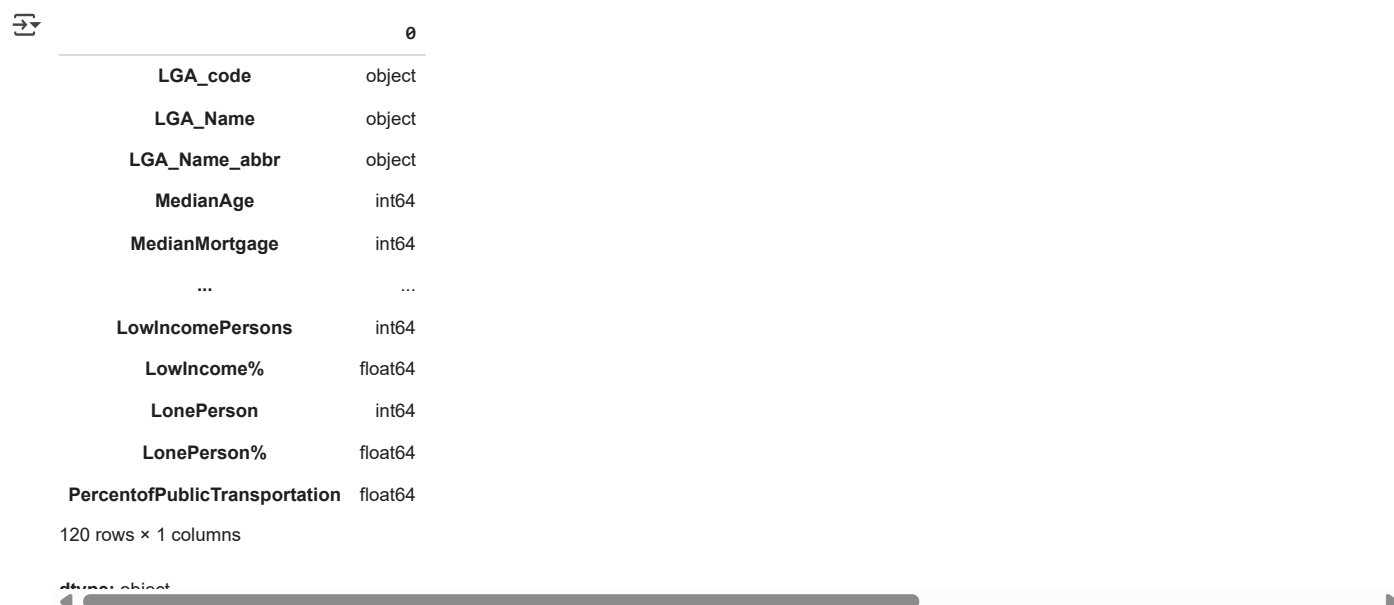
- LGA\_code: Número entero positivo
- LGA\_Name: texto
- MedianAge: Número entero positivo
- MedianMortgage: Número entero positivo

cases\_df.dtypes



	0
notification_date	object
postcode	float64
lhd_2010_code	object
lhd_2010_name	object
lga_code19	object
lga_name19	object

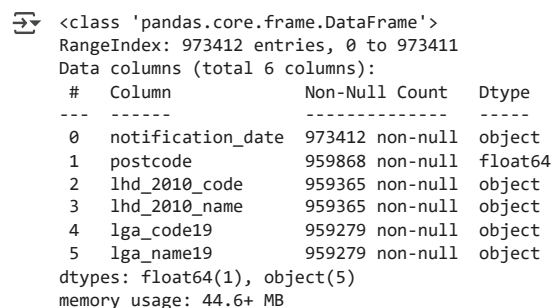
socio\_df.dtypes



	0
LGA_code	object
LGA_Name	object
LGA_Name_abbr	object
MedianAge	int64
MedianMortgage	int64
...	...
LowIncomePersons	int64
LowIncome%	float64
LonePerson	int64
LonePerson%	float64
PercentofPublicTransportation	float64

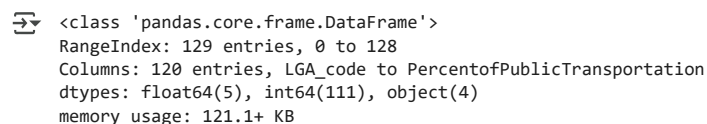
120 rows × 1 columns

cases\_df.info()



```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 973412 entries, 0 to 973411
Data columns (total 6 columns):
#   Column                Non-Null Count  Dtype
---  -
0   notification_date      973412 non-null object
1   postcode               959868 non-null float64
2   lhd_2010_code          959365 non-null object
3   lhd_2010_name          959365 non-null object
4   lga_code19             959279 non-null object
5   lga_name19             959279 non-null object
dtypes: float64(1), object(5)
memory usage: 44.6+ MB
```

socio\_df.info()



```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 129 entries, 0 to 128
Columns: 120 entries, LGA_code to PercentofPublicTransportation
dtypes: float64(5), int64(111), object(4)
memory usage: 121.1+ KB
```

¿Entre qué rangos están los datos de cada columna?, valores únicos, min, max?

cases\_df.describe()



postcode

count	959868.000000
mean	2308.971416
std	237.491267
min	2000.000000
25%	2144.000000
50%	2212.000000
75%	2502.000000
max	2990.000000

```
unique_postcodes = cases_df['postcode'].nunique()
print(unique_postcodes)
```



764

```
unique_postcodes = cases_df['lga_code19'].nunique()
print(unique_postcodes)
```



130

```
socio_df.describe()
```



	MedianAge	MedianMortgage	MedianPersonIncome	MedianRent	MedianFamilyIncome	AverageBedroom	MedianHouseholdIncome	Average...
count	129.000000	129.000000	129.000000	129.000000	129.000000	129.000000	129.000000	
mean	41.589147	1605.457364	647.015504	287.651163	1640.457364	0.81938	1343.341085	
std	5.197073	581.211773	175.203408	136.960798	537.030522	0.10464	445.674813	
min	32.000000	433.000000	439.000000	90.000000	923.000000	0.70000	767.000000	
25%	37.000000	1148.000000	540.000000	180.000000	1326.000000	0.80000	1042.000000	
50%	42.000000	1495.000000	600.000000	250.000000	1465.000000	0.80000	1196.000000	
75%	45.000000	2000.000000	682.000000	350.000000	1778.000000	0.80000	1495.000000	
max	54.000000	3200.000000	1386.000000	650.000000	3671.000000	1.20000	2687.000000	

8 rows × 116 columns

```
socio_df['OneMethodbyBus'].head()
```



OneMethodbyBus

0	126
1	49
2	97
3	3
4	51

```
repeated_MedianAges = socio_df['MedianAge'].value_counts()
repeated_MedianAges = repeated_MedianAges[repeated_MedianAges > 1]
print("Valores repetidos y sus cantidades:")
print(repeated_MedianAges)
```

```
unique_MedianAges = socio_df['MedianAge'].value_counts()
unique_MedianAges = unique_MedianAges[unique_MedianAges == 1]
print("Valores únicos (que aparecen solo una vez):")
print(unique_MedianAges)
```



```
Valores repetidos y sus cantidades:
MedianAge
40    11
42    10
36    10
44     9
45     9
37     9
```

```

43      8
49      7
41      6
39      6
48      6
47      6
38      5
34      5
35      4
46      4
33      4
51      3
32      3
Name: count, dtype: int64
Valores únicos (que aparecen solo una vez):
MedianAge
54      1
50      1
52      1
53      1
Name: count, dtype: int64

```

```

repeated_MedianMortgages = socio_df['MedianMortgage'].value_counts()
repeated_MedianMortgages = repeated_MedianMortgages[repeated_MedianMortgages > 1]
print("Valores repetidos y sus cantidades:")
print(repeated_MedianMortgages)

```

```

unique_MedianMortgages = socio_df['MedianMortgage'].value_counts()
unique_MedianMortgages = unique_MedianMortgages[unique_MedianMortgages == 1]
print("Valores únicos (que aparecen solo una vez):")
print(unique_MedianMortgages)

```

```

↔ Valores repetidos y sus cantidades:
MedianMortgage
1300      12
1083       8
1733       7
1517       6
2000       5
2167       4
2600       4
1200       4
1148       3
1842       3
2500       3
1400       3
3000       3
867        2
900        2
1950       2
1500       2
1000       2
2200       2
1100       2
Name: count, dtype: int64
Valores únicos (que aparecen solo una vez):
MedianMortgage
1421       1
1393       1
1670       1
950        1
953        1
2220       1
1750       1
1603       1
1662       1
575        1
1800       1
1213       1
1473       1
433        1
1091       1
991        1
1343       1
2150       1
2400       1
982        1
2080       1
981        1
1495       1
3033       1
1387       1
1430       1
1153       1
1324       1
1690       1
1395       1
1007       1

```

```

1768 1
1000 1

repeated_MedianRents = socio_df['MedianRent'].value_counts()
repeated_MedianRents = repeated_MedianRents[repeated_MedianRents > 1]
print("Valores repetidos y sus cantidades:")
print(repeated_MedianRents)

unique_MedianRents = socio_df['MedianRent'].value_counts()
unique_MedianRents = unique_MedianRents[unique_MedianRents == 1]
print("Valores únicos (que aparecen solo una vez):")
print(unique_MedianRents)

```

```

↗ Valores repetidos y sus cantidades:
MedianRent
150 8
200 8
180 7
250 6
160 5
350 5
280 5
220 4
190 4
260 4
340 4
565 3
185 3
320 3
240 2
450 2
255 2
460 2
400 2
380 2
305 2
300 2
170 2
270 2
370 2
650 2
210 2
500 2
Name: count, dtype: int64
Valores únicos (que aparecen solo una vez):
MedianRent
245 1
90 1
105 1
100 1
570 1
231 1
140 1
480 1
395 1
360 1
490 1
520 1
230 1
195 1
560 1
120 1
575 1
440 1
310 1
550 1
470 1
175 1
562 1
265 1
148 1

```

```

repeated_MedianPersonIncomes = socio_df['MedianPersonIncome'].value_counts()
repeated_MedianPersonIncomes = repeated_MedianPersonIncomes[repeated_MedianPersonIncomes > 1]
print("Valores repetidos y sus cantidades:")
print(repeated_MedianPersonIncomes)

unique_MedianPersonIncomes = socio_df['MedianPersonIncome'].value_counts()
unique_MedianPersonIncomes = unique_MedianPersonIncomes[unique_MedianPersonIncomes == 1]
print("Valores únicos (que aparecen solo una vez):")
print(unique_MedianPersonIncomes)

```

```

↗ Valores repetidos y sus cantidades:
MedianPersonIncome
600 3
538 2
439 2
524 2

```

```

632    2
460    2
552    2
540    2
509    2
571    2
640    2
577    2
738    2
584    2
609    2
728    2
Name: count, dtype: int64
Valores únicos (que aparecen solo una vez):
MedianPersonIncome
719    1
634    1
688    1
620    1
672    1
..
946    1
645    1
1365   1
869    1
717    1
Name: count, Length: 96, dtype: int64

```

```

repeated_OneMethodbyBuss = socio_df['OneMethodbyBus'].value_counts()
repeated_OneMethodbyBuss = repeated_OneMethodbyBuss[repeated_OneMethodbyBuss > 1]
print("Valores repetidos y sus cantidades:")
print(repeated_OneMethodbyBuss)

unique_OneMethodbyBuss = socio_df['OneMethodbyBus'].value_counts()
unique_OneMethodbyBuss = unique_OneMethodbyBuss[unique_OneMethodbyBuss == 1]
print("Valores únicos (que aparecen solo una vez):")
print(unique_OneMethodbyBuss)

```

```

↔ Valores repetidos y sus cantidades:
OneMethodbyBus
3      10
8       6
7       5
10      4
13      3
9       3
17      3
0       3
18      3
69      2
19      2
16      2
6       2
Name: count, dtype: int64
Valores únicos (que aparecen solo una vez):
OneMethodbyBus
54      1
52      1
127     1
14      1
61      1
..
4123    1
96      1
67      1
1585    1
2300    1
Name: count, Length: 81, dtype: int64

```

```

repeated_LowIncomes = socio_df['LowIncome%'].value_counts()
repeated_LowIncomes = repeated_LowIncomes[repeated_LowIncomes > 1]
print("Valores repetidos y sus cantidades:")
print(repeated_LowIncomes)

```

```

unique_LowIncomes = socio_df['LowIncome%'].value_counts()
unique_LowIncomes = unique_LowIncomes[unique_LowIncomes == 1]
print("Valores únicos (que aparecen solo una vez):")
print(unique_LowIncomes)

```

```

↔ Valores repetidos y sus cantidades:
Series([], Name: count, dtype: int64)
Valores únicos (que aparecen solo una vez):
LowIncome%
0.365752    1
0.412482    1
0.381297    1
0.345919    1

```

```
0.359634    1
..
0.341728    1
0.419020    1
0.204746    1
0.286802    1
0.306099    1
Name: count, Length: 129, dtype: int64
```

### ¿Todos los datos estan en su formato adecuado?

Si todos los datos excepto por la columna "post\_code" de la tabla de casos, esto debido a que en el caso de valores nulos de los que no se sabe su localidad, en lugar de colocar un valor vacío (que representa nulo), escribe literalmente el "None"

```
2021-12-29, None, , , ,
```

### ¿Los datos tienen diferentes unidades de medida?

Si, cada uno de los datos especialmente en la data sociodemografica tiene distintas unidades de medida, por ejemplo en el caso de ingresos esta en dolares australianos, en el caso de conteos de personas esta en un valor entero de personas, en el caso de algunos valores particulares, como habitaciones por persona y demas, estan en su respectiva medida.

### ¿Cuáles son los datos categóricos, ¿hay necesidad de convertirlos en numéricos?

Ninguna de las dos tablas cuenta con datos categóricos, por lo cual no hay necesidad de convertir nada.

## 3. Granularidad

### ¿Qué representa un registro? Describe qué representa cada fila

En la primera tabla, cada registro hace referencia a una notificación de un caso de COVID en una localidad(LGA) y fecha específica.

En la segunda tabla cada registro representa un LGA y sus factores sociodemográficos.

### Si es una data etiquetada, ¿como interpretas la información de las clases?

Ninguna de las dos tablas es etiquetada

### ¿Hay niveles de granularidad de los datos? Por ejemplo, datos a nivel país, región, ciudad. Años, meses, días, horas, minutos, etc

Si, en la primera tabla de casos, se sabe que todos pertenecen al país de Australia en la región de Nueva Gales del Sur, y la data específica que LHD (Local Health District) e internamente desde este específica que LGA(Local Government Area), aparte de también mencionar la fecha del caso de COVID-19.

## 4. Limpieza

### ¿Están todas las filas completas o tenemos campos con valores nulos?

En el caso de la primera tabla, si hay valores NULOS, hay algunos registros, de las cuales solo tenemos la fecha en la que se dió, mas no tenemos el resto de valores que determinan la localidad.

Se hizo una revisión del porcentaje de valores nulos, y se hablo que era de 1.54%, por lo cual no es una cantidad significativa o que afecte a los datos, así que se tomó la decisión de eliminarlos del dataset.

En el caso de la segunda tabla, no había ningún valor NULO, por lo cual no había necesidad de ello.

```
total_filas = len(cases_df)
```

```
# Contar cuántas filas tienen al menos un valor nulo
filas_con_nulos = cases_df.isnull().any(axis=1).sum()
```

```
# Calcular porcentaje
porcentaje_nulos = (filas_con_nulos / total_filas) * 100
```

```
# Mostrar resultados
print(f"Total de filas: {total_filas}")
print(f"Filas con al menos un valor nulo: {filas_con_nulos}")
print(f"Porcentaje de filas con nulos: {porcentaje_nulos:.2f}%")
```

```
Total de filas: 973412
Filas con al menos un valor nulo: 15034
Porcentaje de filas con nulos: 1.54%
```

```
cases_df_clean = cases_df.dropna()
```

```
# Verificar cuántas filas quedaron después de limpiar
print(f"Número de filas después de limpiar: {len(cases_df_clean)}")
```

```
Número de filas después de limpiar: 958378
```

```
total_filas = len(socio_df)

# Contar cuántas filas tienen al menos un valor nulo
filas_con_nulos = socio_df.isnull().any(axis=1).sum()

# Calcular porcentaje
porcentaje_nulos = (filas_con_nulos / total_filas) * 100

# Mostrar resultados
print(f"Total de filas: {total_filas}")
print(f"Filas con al menos un valor nulo: {filas_con_nulos}")
print(f"Porcentaje de filas con nulos: {porcentaje_nulos:.2f}%")
```

```
↗ Total de filas: 129
  Filas con al menos un valor nulo: 0
  Porcentaje de filas con nulos: 0.00%
```

### ¿Siguen alguna distribución?

Al aplicar describe() y analizar la columna de fechas en la tabla de casos COVID, se observa una distribución temporal clara. Inicialmente, hay pocos casos registrados y a medida que pasan los días, la cantidad de casos aumenta, lo cual es característico de un brote epidémico. Esto muestra una distribución acumulativa a lo largo del tiempo.

En la tabla socioeconómica (socio\_df), al aplicar describe() en columnas como MedianPersonIncome y AverageHouseSize, se observa que los valores varían ampliamente, lo que sugiere que la distribución no es uniforme. Estos campos podrían seguir distribuciones sesgadas o tener outliers, lo que se confirmaría mejor con histogramas.

```
cases_df_clean.describe()
```

```
↗
```

	postcode
count	958378.000000
mean	2308.555611
std	237.188603
min	2000.000000
25%	2144.000000
50%	2211.000000
75%	2502.000000
max	2880.000000

```
socio_df.describe()
```

```
↗
```

	MedianAge	MedianMortgage	MedianPersonIncome	MedianRent	MedianFamilyIncome	AverageBedroom	MedianHouseholdIncome	AverageHouseSize
count	129.000000	129.000000	129.000000	129.000000	129.000000	129.000000	129.000000	129.000000
mean	41.589147	1605.457364	647.015504	287.651163	1640.457364	0.81938	1343.341085	1.511111
std	5.197073	581.211773	175.203408	136.960798	537.030522	0.10464	445.674813	0.357778
min	32.000000	433.000000	439.000000	90.000000	923.000000	0.70000	767.000000	0.500000
25%	37.000000	1148.000000	540.000000	180.000000	1326.000000	0.80000	1042.000000	0.750000
50%	42.000000	1495.000000	600.000000	250.000000	1465.000000	0.80000	1196.000000	0.750000
75%	45.000000	2000.000000	682.000000	350.000000	1778.000000	0.80000	1495.000000	0.750000
max	54.000000	3200.000000	1386.000000	650.000000	3671.000000	1.20000	2687.000000	2.000000

8 rows × 116 columns

### Medidas de tendencia central: media aritmética, geométrica, armónica, mediana, moda, desviación estándar.

Tabla 1: Casos COVID

Esta tabla tiene principalmente columnas de tipo texto, pero algunas como postcode pueden ser numéricas. Sin embargo:

- notification\_date es una columna de tipo fecha, por lo cual solo se le puede aplicar la moda
- postcode es un código categórico, no una cantidad con sentido aritmético → solo se puede aplicar la moda (para saber cuál es el más frecuente).
- Otras columnas como lga\_name19, lhd\_2010\_code, etc., son también categóricas → moda únicamente.
- No hay columnas numéricas reales como cantidad de casos, edades o ingresos.



Tabla 2: Datos Socioeconómicos Esta sí tiene campos numéricos reales, como ingresos, tamaño de casa, dormitorios, etc., por lo que se pueden aplicar todas las medidas de tendencia. Sin embargo ya que son bastantes campos, solo se ha elegido algunos como campos representativos.

Selección de columnas representativas por categoría Demográficos (edades, población, sexo):

MedianAge

Male, Female, Population

TotalMale(allages), TotalFemale(allages)

Ingresos (individuales, familiares, por hogar):

MedianPersonIncome

MedianFamilyIncome

MedianHouseholdIncome

(Se omiten los rangos semanales, ya que son muchos y difíciles de analizar individualmente sin más contexto)

Vivienda:

MedianMortgage

MedianRent

AverageBedroom

AverageHouseSize

Área:

Area (km², sirve para ver densidad poblacional si se cruza con población)

Otros relevantes:

LowIncomePersons

LowIncome%

LonePerson

LonePerson%

PercentofPublicTransportation

```
from scipy import stats
```

```
# Moda del postcode (el código postal más frecuente)
moda_notification_date = cases_df_clean['notification_date'].mode().iloc[0]
```

```
# Moda del postcode (el código postal más frecuente)
moda_postcode = cases_df_clean['postcode'].mode().iloc[0]
```

```
# Moda de las LGA (región con más casos reportados)
moda_lga = cases_df_clean['lga_name19'].mode().iloc[0]
```

```
print("Moda de la fecha (más frecuente):", moda_notification_date)
print("Moda del Postcode (más frecuente):", moda_postcode)
print("Moda del LGA (más frecuente):", moda_lga)
```

```
↳ Moda de la fecha (más frecuente): 2022-01-06
Moda del Postcode (más frecuente): 2170.0
Moda del LGA (más frecuente): Canterbury-Bankstown (A)
```

```
import numpy as np
import pandas as pd
from scipy import stats
```

```
# Columnas seleccionadas
cols = [
    'MedianAge', 'Male', 'Female', 'Population',
    'TotalMale(allages)', 'TotalFemale(allages)',
    'MedianPersonIncome', 'MedianFamilyIncome', 'MedianHouseholdIncome',
    'MedianMortgage', 'MedianRent', 'AverageBedroom', 'AverageHouseSize',
    'Area', 'LowIncomePersons', 'LowIncome%', 'LonePerson', 'LonePerson%',
    'PercentofPublicTransportation'
]
```

```
# Asegurarse de que los valores sean numéricos (puede haber strings por comas, espacios, etc.)
for col in cols:
    socio_df[col] = pd.to_numeric(socio_df[col], errors='coerce')
```

```
# Calcular y mostrar estadísticas
```

```

for col in cols:
    values = socio_df[col].dropna()

    print(f"\n📊 {col}")
    print(f"Media aritmética: {np.mean(values):.2f}")
    print(f"Mediana: {np.median(values):.2f}")
    print(f"Moda: {stats.mode(values, keepdims=True)[0][0]:.2f}")
    print(f"Desviación estándar: {np.std(values):.2f}")

    if (values > 0).all():
        print(f"Media geométrica: {stats.gmean(values):.2f}")
        try:
            print(f"Media armónica: {stats.hmean(values):.2f}")
        except:
            print("Media armónica: No válida (valores extremos)")
    else:
        print("Media geométrica/armónica: No aplicables (valores ≤ 0)")

```



#### 📊 MedianAge

Media aritmética: 41.59  
 Mediana: 42.00  
 Moda: 40.00  
 Desviación estándar: 5.18  
 Media geométrica: 41.27  
 Media armónica: 40.94

#### 📊 Male

Media aritmética: 28295.56  
 Mediana: 11318.00  
 Moda: 569.00  
 Desviación estándar: 37171.70  
 Media geométrica: 12046.07  
 Media armónica: 5216.60

#### 📊 Female

Media aritmética: 29161.16  
 Mediana: 11288.00  
 Moda: 492.00  
 Desviación estándar: 38087.94  
 Media geométrica: 12215.55  
 Media armónica: 5081.42

#### 📊 Population

Media aritmética: 57456.82  
 Mediana: 22807.00  
 Moda: 1056.00  
 Desviación estándar: 75241.74  
 Media geométrica: 24270.36  
 Media armónica: 10306.20

#### 📊 TotalMale(allages)

Media aritmética: 25918.02  
 Mediana: 10290.00  
 Moda: 466.00  
 Desviación estándar: 34597.44  
 Media geométrica: 10709.27  
 Media armónica: 4492.96

#### 📊 TotalFemale(allages)

Media aritmética: 26869.44  
 Mediana: 10251.00  
 Moda: 389.00  
 Desviación estándar: 35590.36  
 Media geométrica: 10954.30  
 Media armónica: 4405.09

#### 📊 MedianPersonIncome

Media aritmética: 647.02  
 Mediana: 600.00  
 Moda: 600.00  
 Desviación estándar: 174.52  
 Media geométrica: 628.47  
 Media armónica: 613.51

#### 📊 MedianFamilyIncome

### Correlación y covarianza: permite entender la relación entre dos variables aleatorias

#### Tabla 1 de Casos

- No hay fuertes correlaciones útiles entre las variables numéricas excepto la esperada entre year y month.
- Podrías considerar que la mayoría de estas variables son independientes entre sí desde el punto de vista lineal.

#### Tabla 2 sobre datos sociodemográficos

MedianAge tiene correlación negativa con muchas variables: lo cual sugiere que zonas con población más joven tienden a:

- Tener mayores ingresos familiares.
- Tener hogares más grandes (más personas por hogar).
- Tener más población total.

AverageHouseSize vs Population: 0.55

- Lugares con más población tienden a tener casas con más personas en promedio.

Population vs Area: -0.31

- Zonas más pobladas suelen tener menor área → podrían ser zonas urbanas densas.

```
cases_df_clean['notification_date'] = pd.to_datetime(cases_df_clean['notification_date'])
```

```
# Crear nuevas columnas numéricas derivadas de la fecha
```

```
cases_df_clean['year'] = cases_df_clean['notification_date'].dt.year
```

```
cases_df_clean['month'] = cases_df_clean['notification_date'].dt.month
```

```
cases_df_clean['day'] = cases_df_clean['notification_date'].dt.day
```

```
cases_df_clean['weekday'] = cases_df_clean['notification_date'].dt.weekday # lunes=0
```

```
# Convertir posibles columnas numéricas
```

```
cases_df_clean['postcode'] = pd.to_numeric(cases_df_clean['postcode'], errors='coerce')
```

```
cases_df_clean['lga_code19'] = pd.to_numeric(cases_df_clean['lga_code19'], errors='coerce')
```

```
# Seleccionar solo columnas numéricas
```

```
numeric_cases_df_clean = cases_df_clean[['year', 'month', 'day', 'weekday', 'postcode', 'lga_code19']]
```

```
# Correlación
```

```
print("\n📊 Matriz de correlación (tabla 1):")
```

```
print(numeric_cases_df_clean.corr())
```

```
# Covarianza
```

```
print("\n📈 Matriz de covarianza (tabla 1):")
```

```
print(numeric_cases_df_clean.cov())
```

```
<ipython-input-16-1833c6268d81>:1: 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/pandas-docs/stable/user\\_guide/indexing.html#returning-a-view-vers](https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-vers)

```
cases_df_clean['notification_date'] = pd.to_datetime(cases_df_clean['notification_date'])
```

```
<ipython-input-16-1833c6268d81>:4: 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/pandas-docs/stable/user\\_guide/indexing.html#returning-a-view-vers](https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-vers)

```
cases_df_clean['year'] = cases_df_clean['notification_date'].dt.year
```

```
<ipython-input-16-1833c6268d81>:5: 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/pandas-docs/stable/user\\_guide/indexing.html#returning-a-view-vers](https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-vers)

```
cases_df_clean['month'] = cases_df_clean['notification_date'].dt.month
```

```
<ipython-input-16-1833c6268d81>:6: 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/pandas-docs/stable/user\\_guide/indexing.html#returning-a-view-vers](https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-vers)

```
cases_df_clean['day'] = cases_df_clean['notification_date'].dt.day
```

```
<ipython-input-16-1833c6268d81>:7: 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/pandas-docs/stable/user\\_guide/indexing.html#returning-a-view-vers](https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-vers)

```
cases_df_clean['weekday'] = cases_df_clean['notification_date'].dt.weekday # lunes=0
```

```
<ipython-input-16-1833c6268d81>:10: 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/pandas-docs/stable/user\\_guide/indexing.html#returning-a-view-vers](https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-vers)

```
cases_df_clean['postcode'] = pd.to_numeric(cases_df_clean['postcode'], errors='coerce')
```

```
<ipython-input-16-1833c6268d81>:11: 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/pandas-docs/stable/user\\_guide/indexing.html#returning-a-view-vers](https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-vers)

```
cases_df_clean['lga_code19'] = pd.to_numeric(cases_df_clean['lga_code19'], errors='coerce')
```

```
📊 Matriz de correlación (tabla 1):
```

	year	month	day	weekday	postcode	lga_code19
year	1.000000	-0.949682	-0.393600	-0.005587	0.078109	0.028149
month	-0.949682	1.000000	0.411642	0.002762	-0.080273	-0.015548
day	-0.393600	0.411642	1.000000	0.009944	0.013074	0.011974
weekday	-0.005587	0.002762	0.009944	1.000000	-0.005074	0.004663
postcode	0.078109	-0.080273	0.013074	-0.005074	1.000000	-0.165045
lga_code19	0.028149	-0.015548	0.011974	0.004663	-0.165045	1.000000

Matriz de covarianza (tabla 1):

	year	month	day	weekday	postcode
year	0.182860	-1.666483	-1.574717	-0.004605	7.922401
month	-1.666483	16.839417	15.804142	0.021848	-78.132049

```
socio_df_numeric = socio_df.apply(pd.to_numeric, errors='coerce')
```

# Opcional: puedes seleccionar un subconjunto si hay muchas columnas

```
subset_cols = ['MedianAge', 'MedianPersonIncome', 'MedianFamilyIncome', 'AverageHouseSize', 'Population', 'Area']
subset = socio_df_numeric[subset_cols]
```

# Correlación

```
print("\n Matriz de correlación (tabla 2):")
print(subset.corr())
```

# Covarianza

```
print("\n Matriz de covarianza (tabla 2):")
print(subset.cov())
```



Matriz de correlación (tabla 2):

	MedianAge	MedianPersonIncome	MedianFamilyIncome
MedianAge	1.000000	-0.437652	-0.476469
MedianPersonIncome	-0.437652	1.000000	0.956668
MedianFamilyIncome	-0.476469	0.956668	1.000000
AverageHouseSize	-0.642882	0.142680	0.298170
Population	-0.447274	0.205362	0.317997
Area	0.022078	-0.136626	-0.232931

	AverageHouseSize	Population	Area
MedianAge	-0.642882	-0.447274	0.022078
MedianPersonIncome	0.142680	0.205362	-0.136626
MedianFamilyIncome	0.298170	0.317997	-0.232931
AverageHouseSize	1.000000	0.547527	-0.161238
Population	0.547527	1.000000	-0.306627
Area	-0.161238	-0.306627	1.000000

Matriz de covarianza (tabla 2):

	MedianAge	MedianPersonIncome	MedianFamilyIncome
MedianAge	27.009569	-3.985014e+02	-1.329818e+03
MedianPersonIncome	-398.501393	3.069623e+04	9.001252e+04
MedianFamilyIncome	-1329.818435	9.001252e+04	2.884018e+05
AverageHouseSize	-0.858412	6.422620e+00	4.114033e+01
Population	-175582.620700	2.717762e+06	1.289942e+07
Area	1308.019728	-2.753674e+05	-1.438444e+06

	AverageHouseSize	Population	Area
MedianAge	-0.858412	-1.755826e+05	1.308020e+03
MedianPersonIncome	6.422620	2.717762e+06	-2.753674e+05
MedianFamilyIncome	41.140334	1.289942e+07	-1.438444e+06
AverageHouseSize	0.066010	1.062575e+04	-4.746475e+02
Population	10625.749637	5.705549e+09	-2.661030e+08
Area	-474.647484	-2.661030e+08	1.307952e+08

## ✓ Análisis de outliers

¿Cuáles son los Outliers? (unos pocos datos aislados que difieren drásticamente del resto y “contaminan” ó desvían las distribuciones)

Tabla 1 de Casos de COVID-19

year: Tiene un 21.97% de valores outlier, y todos los valores son 2020 y 2021. Esto sugiere que el resto de la tabla contiene datos de 2020, los cuales a pesar de ser unos pocos meses, al presentar mayor cantidad de casos, contiene casi el 80% de datos restantes.

Tabla 2 de Datos Sociodemográficos

- Variables económicas (como MedianPersonIncome, MedianRent, MedianFamilyIncome) presentan entre 2% y 10% de valores atípicos. Esto puede indicar:
- Ciudades con ingresos o rentas muy altos o bajos.
- Posibles errores de captura si los valores son inesperadamente extremos.
- AverageBedroom: 44.19% de valores outliers, principalmente con valores como 0.7, 0.9, 1.1, 1.2, etc. Posiblemente estas son zonas con unidades pequeñas (tipo estudio), lo cual es inusual comparado con el resto.
- Demográficos (Male, Female, Population, edades): Muchos valores considerados outliers, alrededor del 10–15%, lo cual indica:
- Municipios con poblaciones atípicamente grandes o pequeñas.
- Podría tener sentido revisar si estas localidades tienen una demografía muy distinta al promedio.

```
def detectar_outliers_iqr(df):
    outliers = {}
    for col in df.select_dtypes(include='number').columns:
        Q1 = df[col].quantile(0.25)
        Q3 = df[col].quantile(0.75)
        IQR = Q3 - Q1
        lower = Q1 - 1.5 * IQR
        upper = Q3 + 1.5 * IQR
        outlier_rows = df[(df[col] < lower) | (df[col] > upper)]
        if not outlier_rows.empty:
            outliers[col] = {
                'count': outlier_rows.shape[0],
                'percent': (outlier_rows.shape[0] / df.shape[0]) * 100,
                'values': outlier_rows[col].unique()
            }
    return outliers
```

```
outliers_tabla1 = detectar_outliers_iqr(cases_df_clean)
outliers_tabla2 = detectar_outliers_iqr(socio_df)
```

```
# Mostrar resultados
print("📊 Outliers en Tabla 1:")
print(outliers_tabla1)
print("\n📊 Outliers en Tabla 2:")
print(outliers_tabla2)
```

```
📊 Outliers en Tabla 1:
{'year': {'count': 210582, 'percent': 21.97274979183579, 'values': array([2020, 2021], dtype=int32)}, 'month': {'count': 208245,

📊 Outliers en Tabla 2:
{'MedianPersonIncome': {'count': 12, 'percent': 9.30232558139535, 'values': array([ 977,  957,  942, 1149, 1295, 1386,  916,  933
1365])}, 'MedianRent': {'count': 3, 'percent': 2.3255813953488373, 'values': array([650, 622])}, 'MedianFamilyIncome': {'c
2671, 3626}}}, 'AverageBedroom': {'count': 57, 'percent': 44.18604651162791, 'values': array([0.7, 0.9, 1.1, 1. , 1.2])},
123507, 113158,  96822, 106713, 107852, 100542]}}, 'Female': {'count': 14, 'percent': 10.852713178294573, 'values': array(
129370, 112996,  99243, 111750, 100530, 103087]}}, 'Area': {'count': 10, 'percent': 7.751937984496124, 'values': array([21
22308.4, 26256.2, 93214.2])}, 'Population': {'count': 14, 'percent': 10.852713178294573, 'values': array([336962, 346302,
252878, 226149, 196066, 218464, 208374, 203630])}, 'Citizen': {'count': 12, 'percent': 9.30232558139535, 'values': array([
159035, 169009, 196928, 176569]}}, 'MaleAge(0-14)': {'count': 14, 'percent': 10.852713178294573, 'values': array([37859, 1
20275, 20238, 20767, 16839, 18004])}, 'MaleAge(15-24)': {'count': 19, 'percent': 14.728682170542637, 'values': array([ 981
13973, 10262, 13164, 12673, 12921,  9391, 12715, 12831, 10560,
13540])}, 'MaleAge(25-34)': {'count': 19, 'percent': 14.728682170542637, 'values': array([23668, 10449, 24885, 16069, 2121
13063, 11172, 13121, 20468, 13590, 11414,  9656, 11947, 29423,
12159])}, 'MaleAge(35-44)': {'count': 18, 'percent': 13.953488372093023, 'values': array([ 9311, 24273,  9795, 21828, 1775
10674, 13406, 17227, 16926, 12511, 13821, 15954, 10736, 11935])}, 'MaleAge(45-54)': {'count': 14, 'percent': 10.8527131782
11972, 11224, 13874, 10531, 12202]}}, 'MaleAge(55-64)': {'count': 12, 'percent': 9.30232558139535, 'values': array([15459,
10007, 12443, 10803])}, 'MaleAge(65-74)': {'count': 9, 'percent': 6.976744186046512, 'values': array([ 9720, 11374, 15554,
114420, 103457,  90173, 100530,  88249,  93023])}, 'FemaleAge(0-14)': {'count': 14, 'percent': 10.852713178294573, 'values
19381, 19327, 19540, 15975, 17170]}}, 'FemaleAge(15-24)': {'count': 20, 'percent': 15.503875968992247, 'values': array([ 9
10687, 13542,  9951, 12241, 11714, 12506,  9450, 12113, 15478,
 9772, 12554])}, 'FemaleAge(25-34)': {'count': 19, 'percent': 14.728682170542637, 'values': array([25677, 11496, 25044, 17
14314, 11049, 14292, 21236, 14416, 11991, 10209, 12757, 28615,
12103])}, 'FemaleAge(35-44)': {'count': 18, 'percent': 13.953488372093023, 'values': array([10572, 24477, 10176, 22302, 19
11734, 14260, 18508, 16005, 13040, 14815, 12950, 12032, 12504])}, 'FemaleAge(45-54)': {'count': 14, 'percent': 10.85271317
12816, 11879, 14955, 11238, 12679]}}, 'FemaleAge(55-64)': {'count': 12, 'percent': 9.30232558139535, 'values': array([1642
10812, 13235, 11685])}, 'FemaleAge(65-74)': {'count': 7, 'percent': 5.426356589147287, 'values': array([10442, 11759, 1765
119690, 104947,  92784, 105068,  84618,  95723])}, 'WeeklyFamilyHouseholdsIncome(Negative)': {'count': 21, 'percent': 16.2
427,  763,  364,  464,  485,  373,  775,  357,  380,  442]}}, 'WeeklyFamilyHouseholdsIncome(1-149)': {'count': 13, 'perce
288,  306,  380]}}, 'WeeklyFamilyHouseholdsIncome(300-399)': {'count': 11, 'percent': 8.527131782945736, 'values': array(
1539]}}, 'WeeklyFamilyHouseholdsIncome(500-649)WeeklyFamilyHouseholdsIncome(650-799)': {'count': 11, 'percent': 8.52713178
3401, 3978]}}, 'WeeklyFamilyHouseholdsIncome(1250-1499)': {'count': 13, 'percent': 10.077519379844961, 'values': array([61
3327, 3831]}}, 'WeeklyFamilyHouseholdsIncome(1500-1749)': {'count': 13, 'percent': 10.077519379844961, 'values': array([55
2890, 3037]}}, 'WeeklyFamilyHouseholdsIncome(1750-1999)': {'count': 13, 'percent': 10.077519379844961, 'values': array([57
3280, 3226]}}, 'WeeklyFamilyHouseholdsIncome(2000-2499)': {'count': 13, 'percent': 10.077519379844961, 'values': array([12
7483, 7332, 4985, 6346]}}, 'WeeklyFamilyHouseholdsIncome(2500-2999)': {'count': 17, 'percent': 13.178294573643413, 'va
5711, 5174, 6352, 3858, 4180, 4734]}}, 'WeeklyFamilyHouseholdsIncome(3000-3499)': {'count': 19, 'percent': 14.728682170542
4748, 4008, 3321, 2312, 4866, 3293, 3533, 3103]}}, 'WeeklyFamilyHouseholdsIncome(3500-3999)': {'count': 16, 'percent': 12.
2211, 3689, 3211, 3060, 2052]}}, 'WeeklyFamilyHouseholdsIncome(Over4000)': {'count': 19, 'percent': 14.728682170542637, 'v
15354, 6149, 6846, 4542, 9643, 9395, 8778, 4516, 5356,
5009]}}, 'WeeklyFamilyHouseholdsIncome(PartialIncomeStated)': {'count': 12, 'percent': 9.30232558139535, 'values': array(
4864]}}, 'WeeklyFamilyHouseholdsIncome(NotStated)': {'count': 12, 'percent': 9.30232558139535, 'values': array([1163, 628
721]}}, 'WeeklyNon-FamilyHouseholdsIncome(Negative)': {'count': 17, 'percent': 13.178294573643413, 'values': array([ 618,
760, 1277,  682,  448, 3661,  627]}}, 'WeeklyNon-FamilyHouseholdsIncome(1-149)': {'count': 13, 'percent': 10.077519379844
1416]}}, 'WeeklyNon-FamilyHouseholdsIncome(300-399)': {'count': 14, 'percent': 10.852713178294573, 'values': array([18
1297, 2368, 2438]}}, 'WeeklyNon-FamilyHouseholdsIncome(400-499)': {'count': 8, 'percent': 6.2015503875969, 'values': array
1729]}}, 'WeeklyNon-FamilyHouseholdsIncome(1250-1499)': {'count': 13, 'percent': 10.077519379844961, 'values': array([1158
2848, 1166]}}, 'WeeklyNon-FamilyHouseholdsIncome(1500-1749)': {'count': 15, 'percent': 11.627906976744185, 'values': array
805, 1126, 2898,  986]}}, 'WeeklyNon-FamilyHouseholdsIncome(1750-1999)': {'count': 13, 'percent': 10.077519379844961, 'va
2505, 749]}}, 'WeeklyNon-FamilyHouseholdsIncome(2000-2499)': {'count': 19, 'percent': 14.728682170542637, 'values': array
1752, 1025, 1302, 5462, 1060,  777,  980,  973]}}, 'WeeklyNon-FamilyHouseholdsIncome(2500-2999)': {'count': 19, 'percent':
135, 457, 198, 182, 1161, 288, 159, 142]}}, 'WeeklyNon-FamilyHouseholdsIncome(3000-3499)': {'count': 19, 'percent':
854, 114,  80, 111]}}, 'WeeklyNon-FamilyHouseholdsIncome(3500-3999)': {'count': 19, 'percent': 14.728682170542637, 'values
967, 414,  540, 4086,  983,  372, 1183]}}, 'WeeklyNon-FamilyHouseholdsIncome(Over4000)': {'count': 22, 'percent': 17.054
320, 273, 463, 138, 128,  92, 1750, 535, 166, 405]}}, 'WeeklyNon-FamilyHouseholdsIncome(PartialIncomeStated)': {'
```


## Visualización

Tabla 1

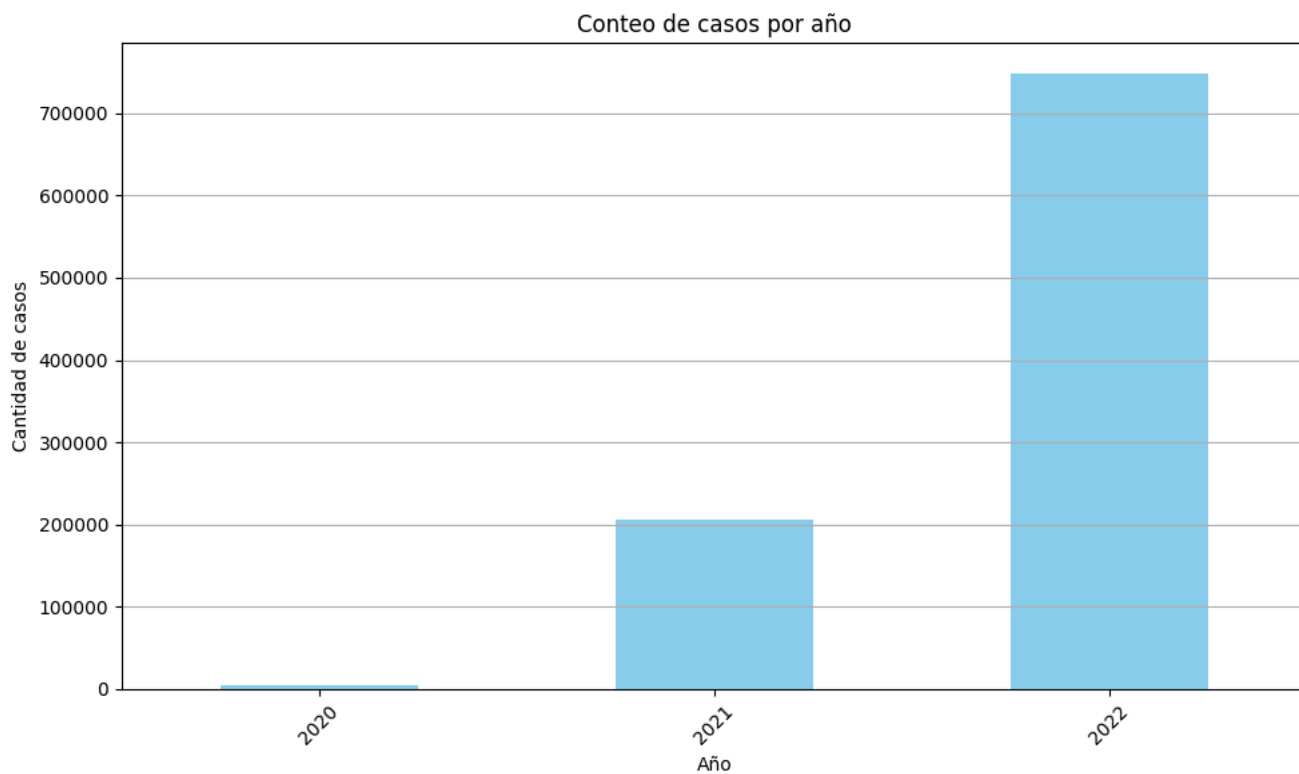
```
import matplotlib.pyplot as plt
cases_df_clean['notification_date'] = pd.to_datetime(cases_df_clean['notification_date'])

# Agrupar por año y contar
cases_by_year = cases_df_clean['notification_date'].dt.year.value_counts().sort_index()
```

```
# Plot
plt.figure(figsize=(10, 6))
cases_by_year.plot(kind='bar', color='skyblue')
plt.title('Conteo de casos por año')
plt.xlabel('Año')
plt.ylabel('Cantidad de casos')
plt.xticks(rotation=45)
plt.grid(axis='y')
plt.tight_layout()
plt.show()
```

 <ipython-input-22-1d38d94f675d>:2: 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/pandas-docs/stable/user\\_guide/indexing.html#returning-a-view-versus](https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus)  
cases\_df\_clean['notification\_date'] = pd.to\_datetime(cases\_df\_clean['notification\_date'])



```
# Asegúrate que la columna 'fecha' sea datetime
# Asegúrate que la columna 'notification_date' sea datetime
cases_df_clean['notification_date'] = pd.to_datetime(cases_df_clean['notification_date'])
```


```
# Filtrar solo casos de 2020 y 2021
filtered_df = cases_df_clean[
    (cases_df_clean['notification_date'].dt.year == 2020) |
    (cases_df_clean['notification_date'].dt.year == 2021)
]
```

```
# Crear una columna con año-mes
filtered_df['año_mes'] = filtered_df['notification_date'].dt.to_period('M').astype(str)
```

```
# Agrupar y contar por año-mes
cases_by_year_month = filtered_df['año_mes'].value_counts().sort_index()
```

```
# Graficar
plt.figure(figsize=(14, 6))
cases_by_year_month.plot(kind='bar', color='mediumseagreen')
plt.title('Conteo de casos por año y mes (2020-2021)')
plt.xlabel('Año-Mes')
```

```
plt.ylabel('Cantidad de casos')
plt.xticks(rotation=90)
plt.grid(axis='y')
plt.tight_layout()
plt.show()
```

 <ipython-input-26-50603686acc8>:3: 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/pandas-docs/stable/user\\_guide/indexing.html#returning-a-view-versus-a-copy](https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy)

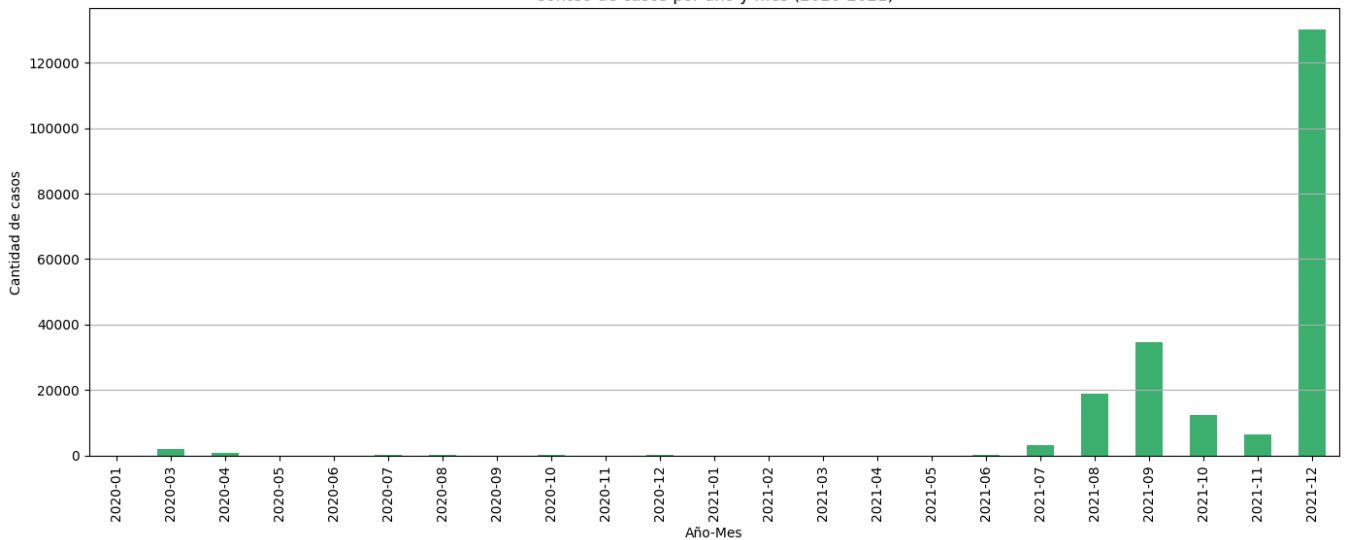
```
cases_df_clean['notification_date'] = pd.to_datetime(cases_df_clean['notification_date'])
```

<ipython-input-26-50603686acc8>:12: 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/pandas-docs/stable/user\\_guide/indexing.html#returning-a-view-versus-a-copy](https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy)

```
filtered_df['año_mes'] = filtered_df['notification_date'].dt.to_period('M').astype(str)
```

Conteo de casos por año y mes (2020-2021)



```
import matplotlib.pyplot as plt
cases_df_clean['notification_date'] = pd.to_datetime(cases_df_clean['notification_date'])

# Filtrar solo casos de 2020 y 2021
filtered_df = cases_df_clean[
    (cases_df_clean['notification_date'].dt.year == 2020) |
    (cases_df_clean['notification_date'].dt.year == 2021)
]

# Crear una columna con año-mes
filtered_df['año_mes'] = filtered_df['notification_date'].dt.to_period('M').astype(str)

# Agrupar y contar por año-mes
cases_by_year_month = filtered_df['año_mes'].value_counts().sort_index()

# Graficar
plt.figure(figsize=(14, 6))
cases_by_year_month.plot(kind='bar', color='mediumseagreen')
plt.title('Conteo de casos por año y mes (2020-2021)')
plt.xlabel('Año-Mes')
plt.ylabel('Cantidad de casos')
plt.xticks(rotation=90)
plt.grid(axis='y')
plt.tight_layout()
plt.show()
```



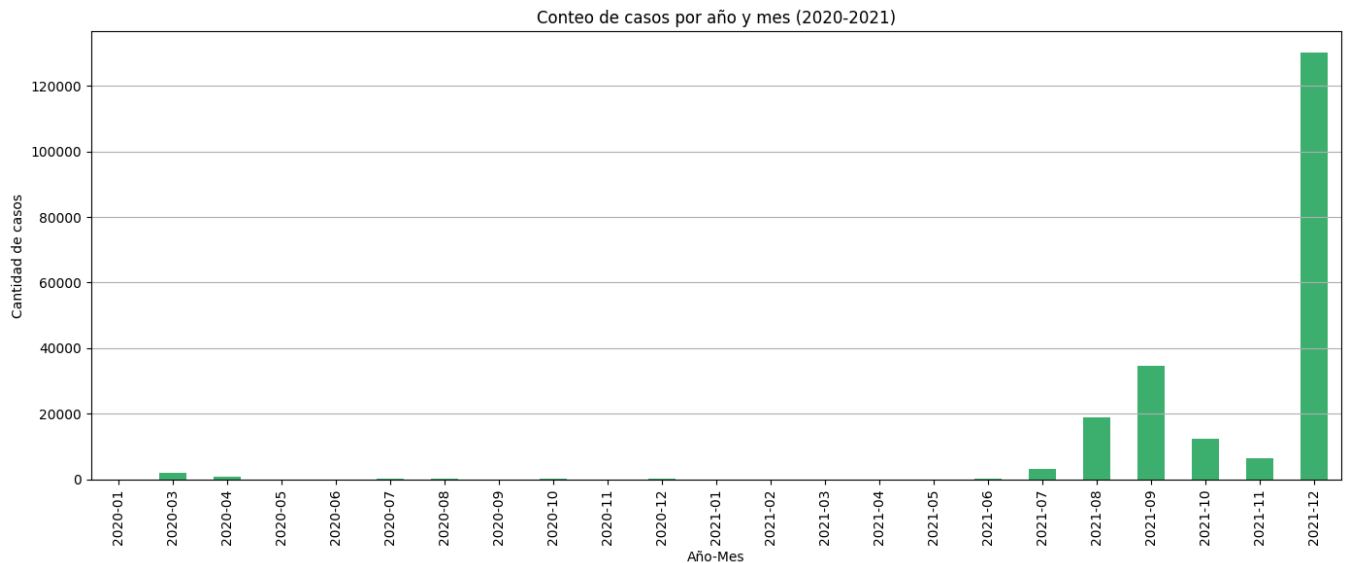
```

<ipython-input-5-85f5244b1a27>:2: 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/pandas-docs/stable/user\_guide/indexing.html#returning-a-view-versus-cases\_df\_clean\['notification\_date'\] = pd.to\_datetime\(cases\_df\_clean\['notification\_date'\]\)
<ipython-input-5-85f5244b1a27>:11: 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/pandas-docs/stable/user\_guide/indexing.html#returning-a-view-versus-filtered\_df\['año\_mes'\] = filtered\_df\['notification\_date'\].dt.to\_period\('M'\).astype\(str\)

```



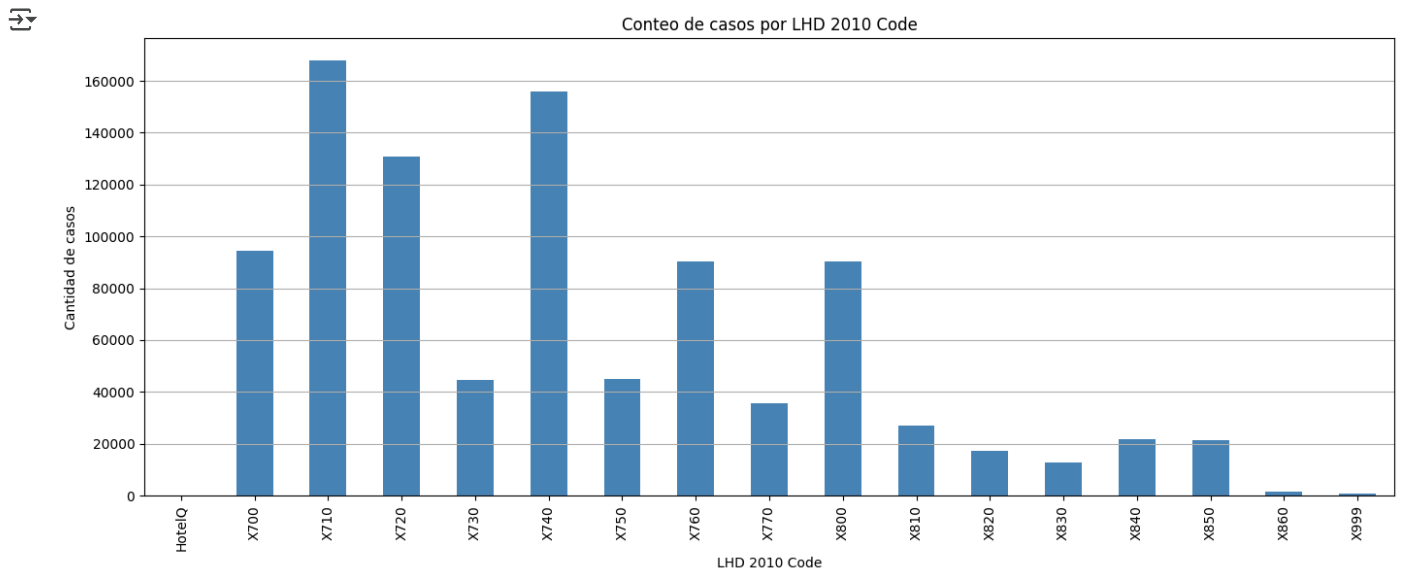
```

import matplotlib.pyplot as plt

# Agrupar y contar por 'lhd_2010_code'
cases_by_lhd = cases_df_clean['lhd_2010_code'].value_counts().sort_index()

# Graficar
plt.figure(figsize=(14, 6))
cases_by_lhd.plot(kind='bar', color='steelblue')
plt.title('Conteo de casos por LHD 2010 Code')
plt.xlabel('LHD 2010 Code')
plt.ylabel('Cantidad de casos')
plt.xticks(rotation=90)
plt.grid(axis='y')
plt.tight_layout()
plt.show()

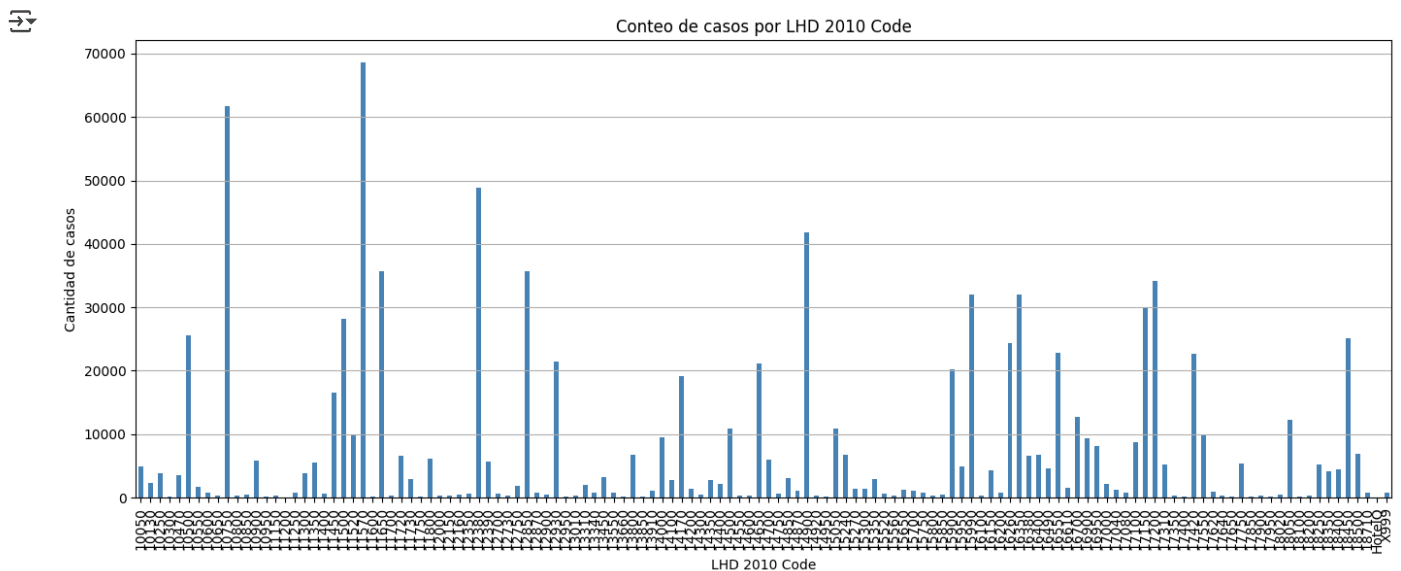
```



```
import matplotlib.pyplot as plt

# Agrupar y contar por 'lhd_2010_code'
cases_by_lhd = cases_df_clean['lga_code19'].value_counts().sort_index()

# Graficar
plt.figure(figsize=(14, 6))
cases_by_lhd.plot(kind='bar', color='steelblue')
plt.title('Conteo de casos por LHD 2010 Code')
plt.xlabel('LHD 2010 Code')
plt.ylabel('Cantidad de casos')
plt.xticks(rotation=90)
plt.grid(axis='y')
plt.tight_layout()
plt.show()
```



```
cases_df_clean['notification_date'] = pd.to_datetime(cases_df_clean['notification_date'])
```

```
# Filtrar solo los años 2020 y 2021
filtered_df = cases_df_clean[
    (cases_df_clean['notification_date'].dt.year.isin([2020, 2021]))
```

```

]

# Especificar el LGA que deseas visualizar
lga_especifico = 'Ryde (C)' # Cambia esto por el nombre que quieras analizar

# Filtrar por ese LGA
filtered_df = filtered_df[filtered_df['lga_name19'] == lga_especifico]

# Crear columna Año-Mes
filtered_df['año_mes'] = filtered_df['notification_date'].dt.to_period('M').astype(str)

# Agrupar y contar casos por mes
cases_by_month = filtered_df['año_mes'].value_counts().sort_index()

# Plot
plt.figure(figsize=(14, 6))
cases_by_month.plot(kind='bar', color='teal')
plt.title(f'Evolución de casos mensuales en {lga_especifico} (2020-2021)')
plt.xlabel('Año-Mes')
plt.ylabel('Cantidad de casos')
plt.xticks(rotation=90)
plt.grid(axis='y')
plt.tight_layout()
plt.show()

```



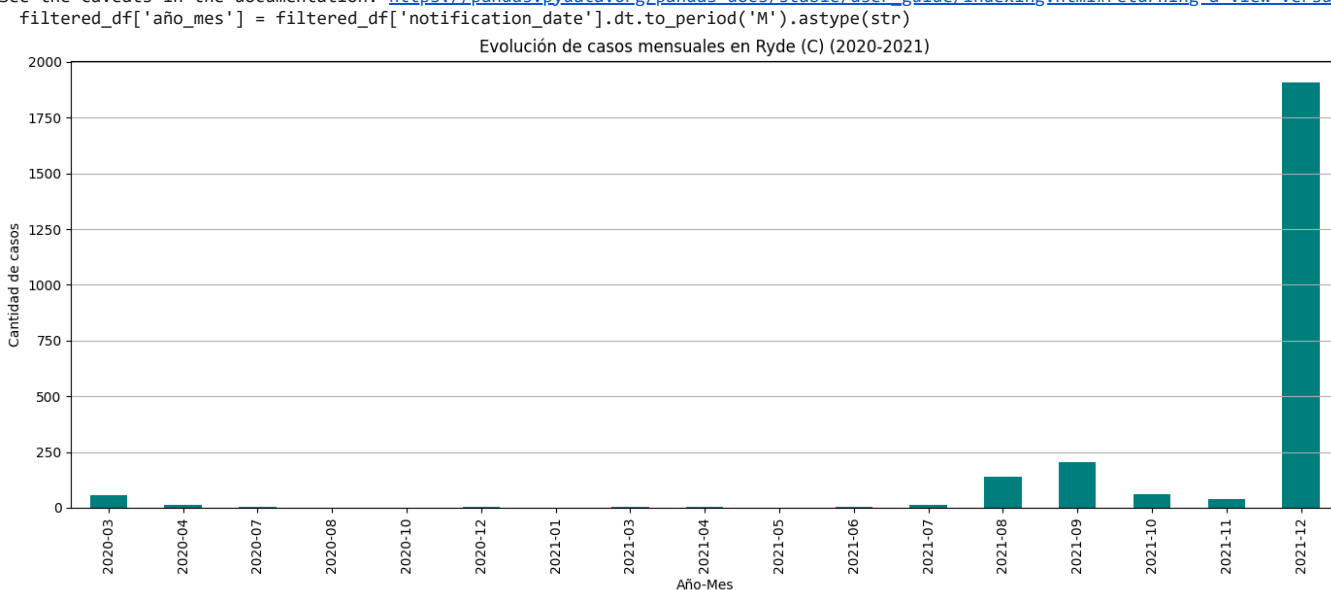
<ipython-input-10-3e5ec5464078>:1: SettingWithCopyWarning:  
A value is trying to be set on a copy of a slice from a DataFrame.  
Try using .loc[row\_indexer,col\_indexer] = value instead

See the caveats in the documentation: [https://pandas.pydata.org/pandas-docs/stable/user\\_guide/indexing.html#returning-a-view-versus-a-copy](https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy)

```
cases_df_clean['notification_date'] = pd.to_datetime(cases_df_clean['notification_date'])
```

<ipython-input-10-3e5ec5464078>:15: 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/pandas-docs/stable/user\\_guide/indexing.html#returning-a-view-versus-a-copy](https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy)



Comienza a programar o [generar](#) con IA.

### Evolución de casos en el tiempo para un LGA específico

```

import matplotlib.pyplot as plt

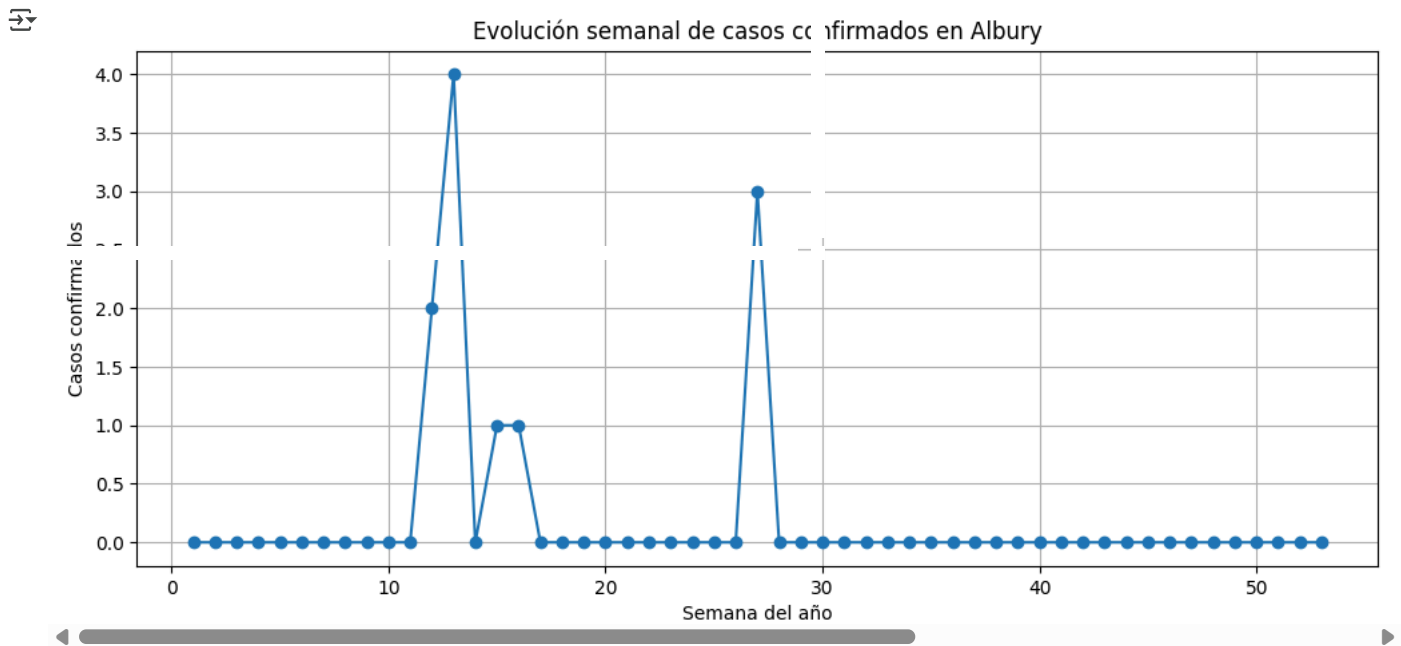
# Filtrar los datos de casos para Albury
albury_cases = df_cases[df_cases['region'] == 'Albury']

# Ordenar por semana por si acaso
albury_cases = albury_cases.sort_values('week')

# Graficar
plt.figure(figsize=(12, 5))

```

```
plt.plot(albury_cases['week'], albury_cases['cases'], marker='o', linestyle='-')
plt.title('Evolución semanal de casos confirmados en Albury')
plt.xlabel('Semana del año')
plt.ylabel('Casos confirmados')
plt.grid(True)
plt.show()
```



```
cases_df_clean = cases_df_clean[cases_df_clean['lga_code19'].str.isnumeric()]
```

```
# Paso 2: Convertir lga_code19 a entero
```

```
cases_df_clean['lga_code19'] = cases_df_clean['lga_code19'].astype(int)
```

```
# Paso 3: Procesar socio_df para extraer código numérico
```

```
socio_df['lga_code_numeric'] = socio_df['LGA_code'].str.replace('LGA', '').astype(int)
```

```
# Paso 4: Contar casos por LGA
```