

I M T 2 2 0 0

I N T R O D U C C I Ó N A C I E N C I A D E D A T O S

2 0 2 2 - 2

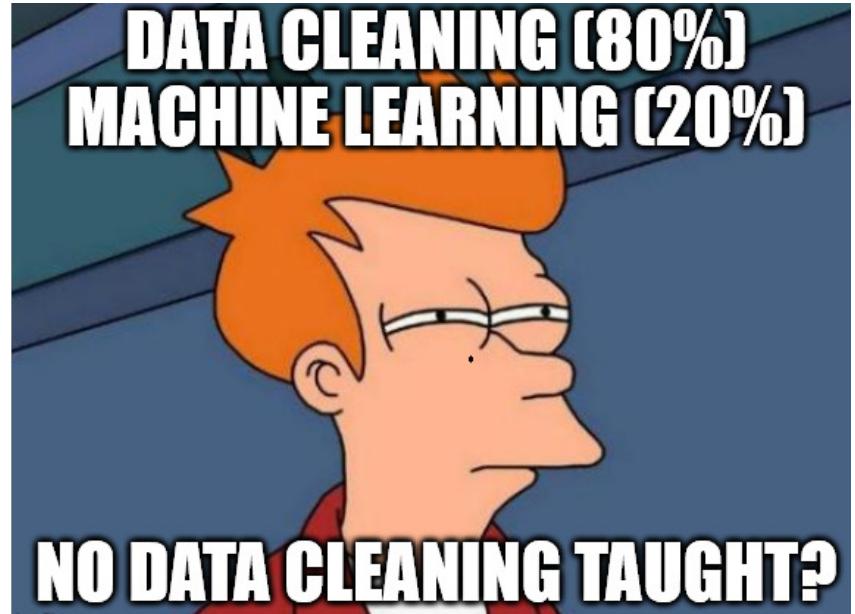
# LIMPIEZA DE DATOS

CLASE 10

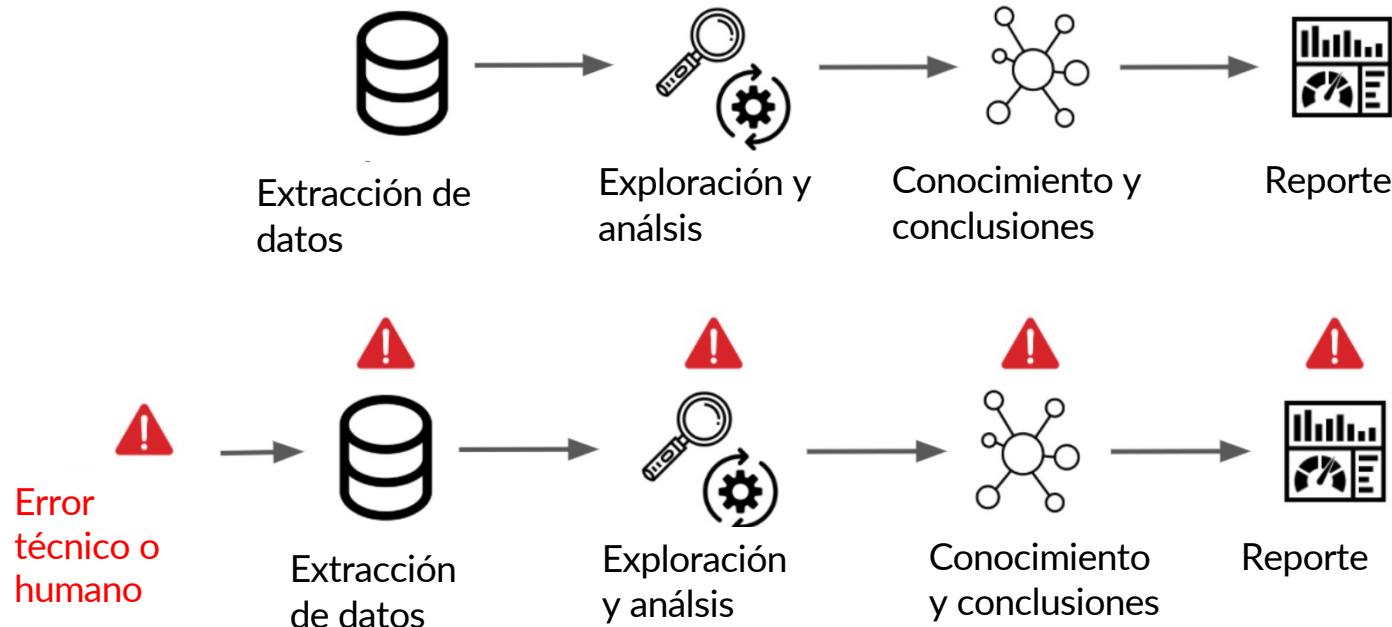
## LIMPIEZA DE DATOS

Es el proceso de identificar aquella parte de la **data incorrecta, incompleta, imprecisa, irrelevante o faltante**, y luego **modificar, reemplazar o eliminar** según corresponda.

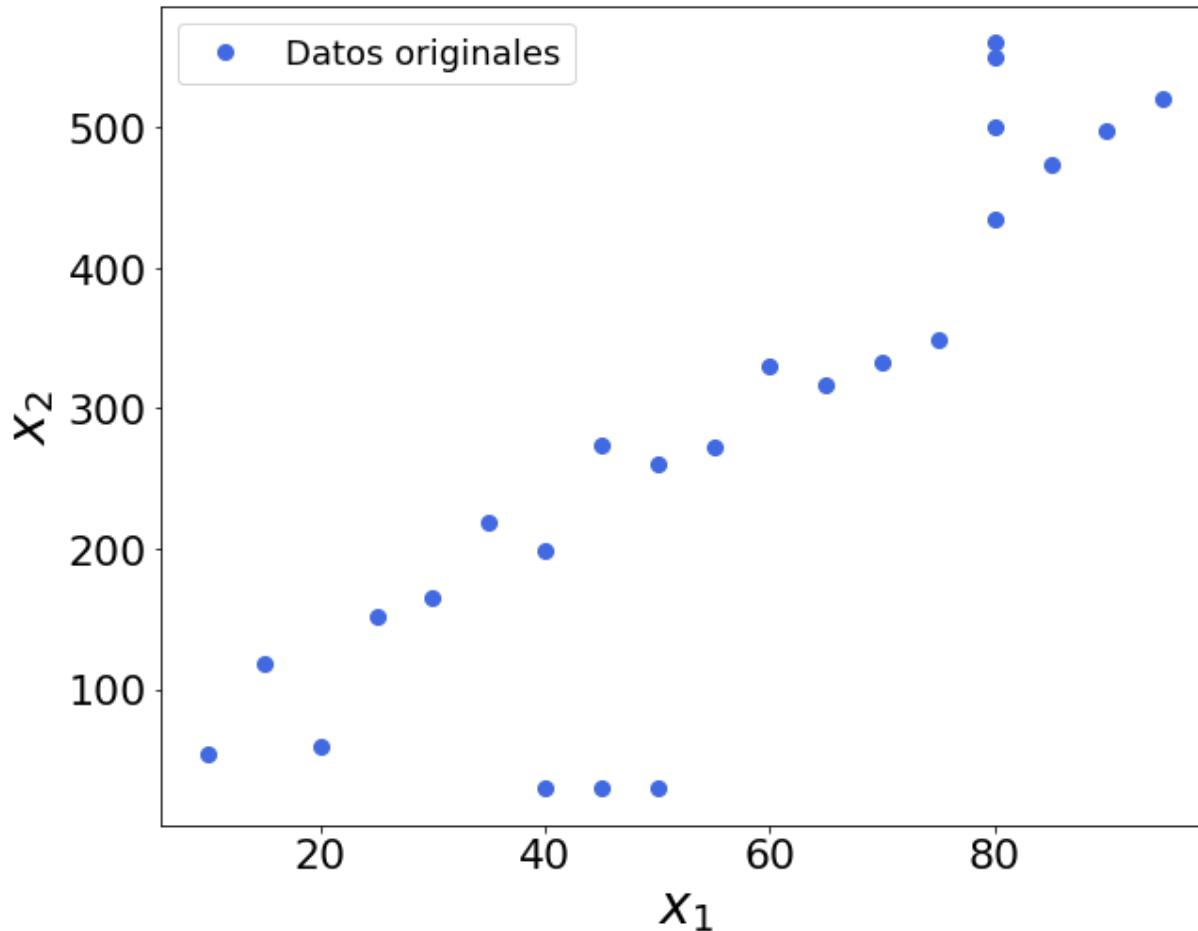
Es una de las tareas esenciales para el buen análisis y modelamiento de los datos.



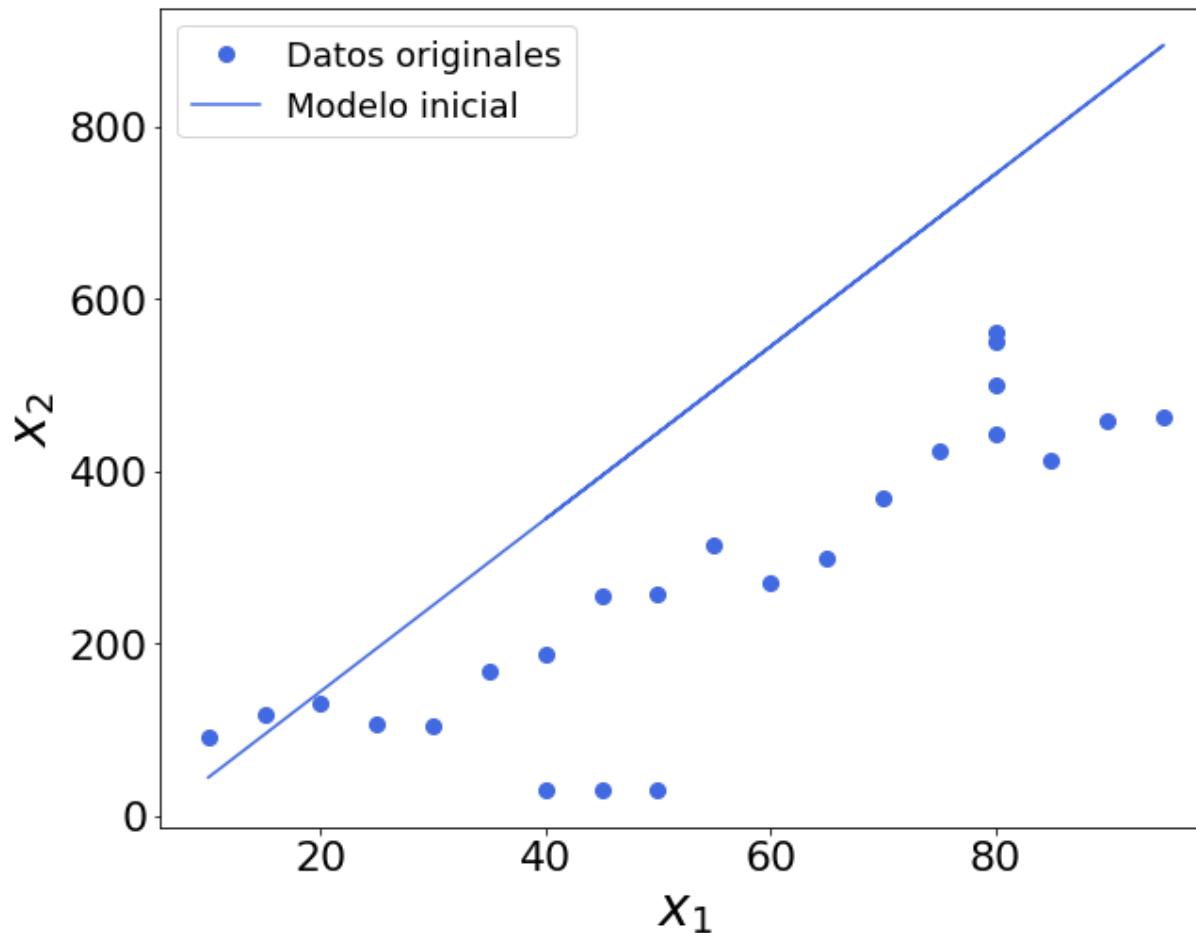
# LIMPIEZA DE DATOS



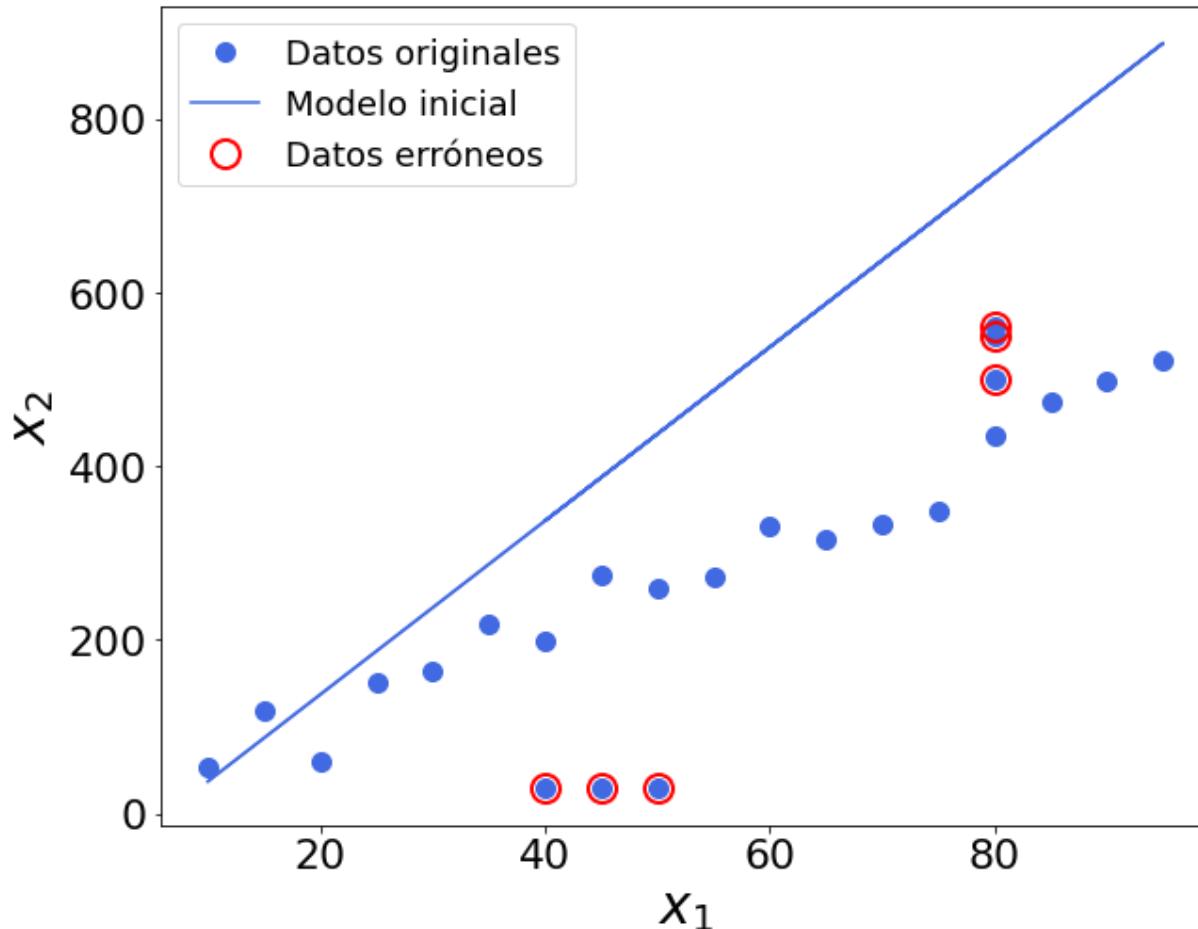
# LIMPIEZA DE DATOS: EJEMPLO



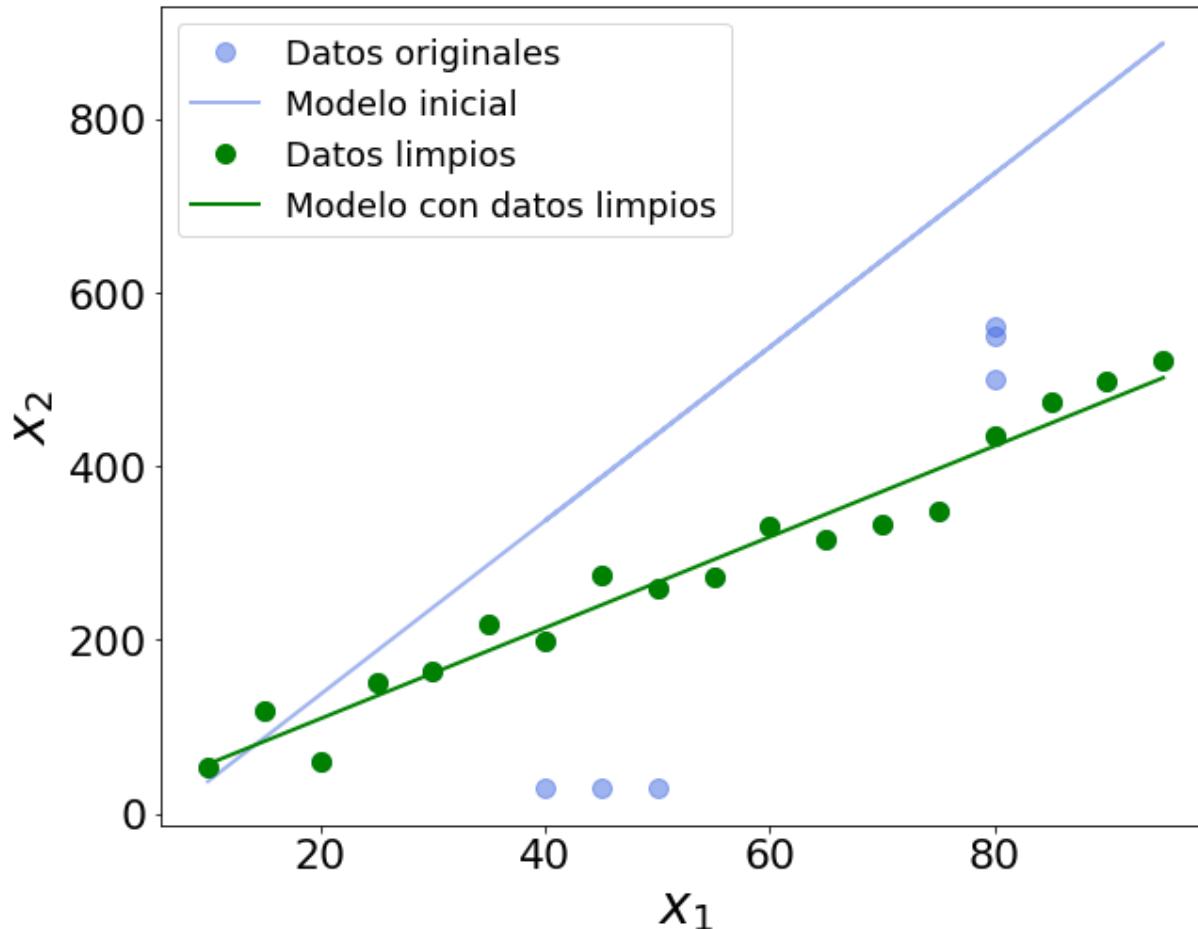
# LIMPIEZA DE DATOS: EJEMPLO



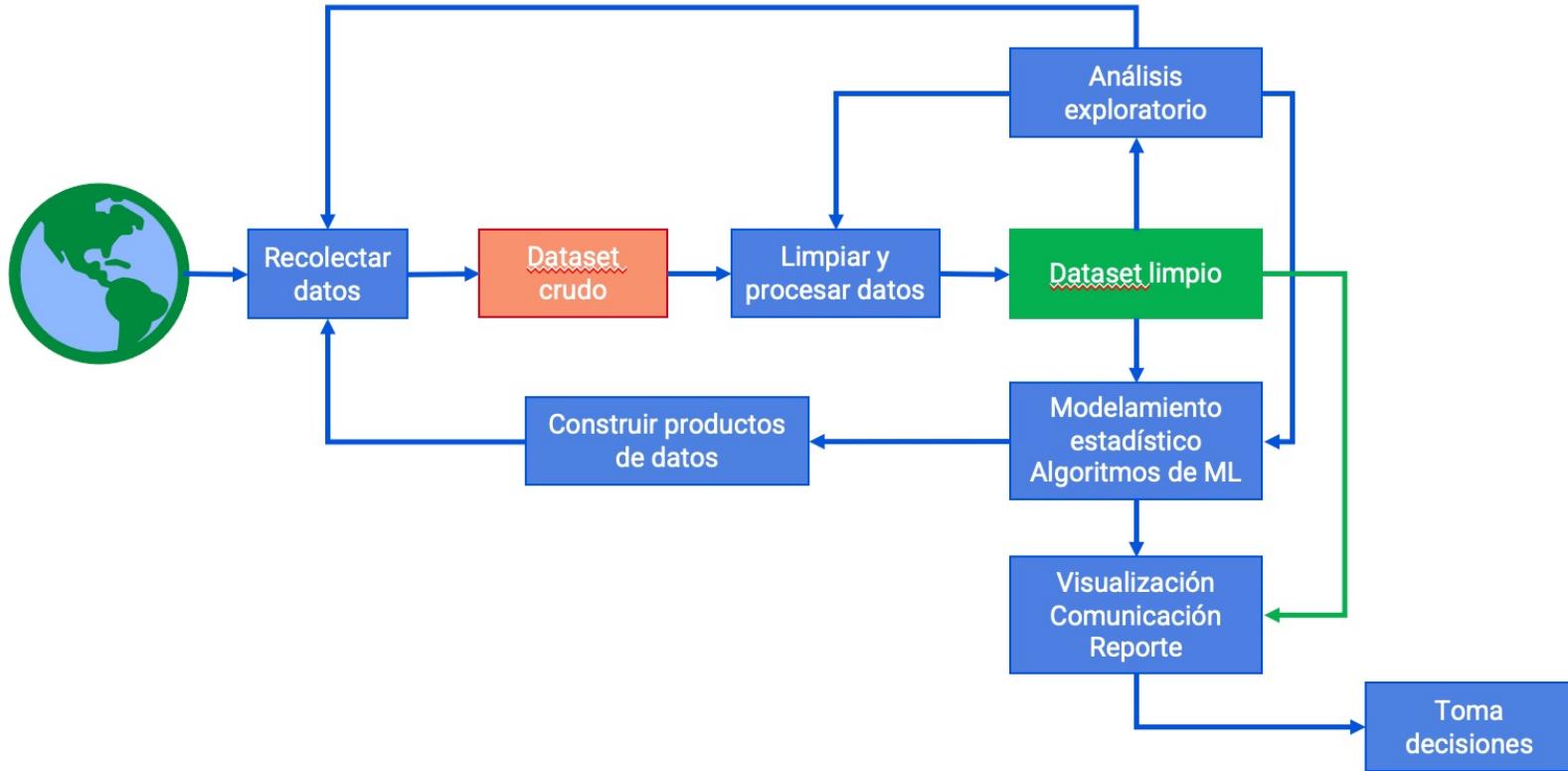
# LIMPIEZA DE DATOS: EJEMPLO



# LIMPIEZA DE DATOS: EJEMPLO



# PROCESO DE CIENCIA DE DATOS



# LIMPIEZA DE DATOS: PROBLEMAS COMUNES

			Datos duplicados		Transformación de valores					
	alias	title	name	price	rating	review_count	distance	coordinates_latitude	coordinates_longitude	location_address1
0	wine_bars	Wine Bars	Barrica 94	\$\$	4.5	72	1390.18	-33.4343	-70.6352	Bellavista 052
1	chilean	Chilean	Barrica 94	\$\$	4.5	72	1390.18	-33.4343	-70.6352	Bellavista 052
2	cocktailbars	Cocktail Bars	Barrica 94	\$\$	4.5	72	1390.18	-33.4343	-70.6352	Bellavista 052
3	chilean	Chilean	Mestizo		4.5	52	6024.2	-33.3941	-70.6	Av. del Bicentenario 4050
4	international	International	Mestizo		4.5	52	6024.2	-33.3941	-70.6	Av. del Bicentenario 4050
...	...	...	...	...	...	...	...	...	...	...
1433	pizza	Pizza	Kalafate	\$	3.5	3	3104.5	-33.4207	-70.607	Paseo Mardoqueo Fernández 19
1434	fooddeliveryservices	Food Delivery Services	Kalafate	\$	3.5	3	3104.5	-33.4207	-70.607	Paseo Mardoqueo Fernández 19
1435	thai	Thai	W.O.K World Oriented Kitchen	Nan	4	1	3192.12	-33.4691	-70.642	Centro Comercial Costanera Center.
1436	wok	Wok	W.O.K World Oriented Kitchen	Nan	4	1	3192.12	-33.4691	-70.642	Centro Comercial Costanera Center.
1437	italian	Italian	La Casa Nostra	\$\$	4	1	1623.28	-33.4393	-70.6422	Victoria Subercaseaux 209

Datos faltantes

Nulos

# LIMPIEZA DE DATOS: PROBLEMAS COMUNES

## Datos innecesarios

	Unnamed: 0	COM-MAN	COM	MAN	DIR	HEIGHT
29663	29663	15103-877	PROVIDENCIA	877	LARRAIN GANDARILLAS 131	7
29664	29664	15103-877	PROVIDENCIA	877	SEMINARIO 128	7
29665	29665	15103-879	PROVIDENCIA	879	ARZOBISPO LARRAIN G 122	8
29666	29666	15103-879	PROVIDENCIA	879	CONDELL 679	5
29667	29667	15103-881	PROVIDENCIA	881	GENERAL BUSTAMANTE 273	7
29668	29668	15103-883	PROVIDENCIA	883	AV GRAL BUSTAMANTE 176	4
29669	29669	15103-89	PROVIDENCIA	89	SANTA MARIA 0206	12
29670	29670	15103-896	PROVIDENCIA	896	MARIN 0366 0388	8
29671	29671	15103-902	PROVIDENCIA	902	BENJAMIN VICUNA MACKENNA 346	5
29672	29672	15103-904	PROVIDENCIA	904	SEMINARIO 343 BL A	6
29673	29673	15103-904	PROVIDENCIA	904	SEMINARIO 343 BL B	6
29674	29674	15103-906	PROVIDENCIA	906	MARIN 0113	4
29675	29675	15103-91	PROVIDENCIA	91	C WALKER 092	7
29676	29676	15103-91	PROVIDENCIA	91	STA MARIA 0316	7
29677	29677	15103-91	PROVIDENCIA	91	STA MARIA 0326	7
29678	29678	15103-91	PROVIDENCIA	91	STA MARIA 0346	7
29679	29679	15103-910	PROVIDENCIA	910	SANTA VICTORIA 0274 A	4
29680	29680	15103-910	PROVIDENCIA	910	SANTA VICTORIA 0274 B	4
29681	29681	15103-910	PROVIDENCIA	910	SANTA VICTORIA 0274 C	4
29682	29682	15103-910	PROVIDENCIA	910	SEMINARIO 388	4
29683	29683	15103-919	PROVIDENCIA	919	BENJAMIN VICUNA MACKENNA 532	4
29684	29684	15103-921	PROVIDENCIA	921	C ANTUNEZ 1869	10
29685	29685	15103-922	PROVIDENCIA	922	C ANTUNEZ 1835	10
29686	29686	15103-923	PROVIDENCIA	923	C ANTUNEZ 1831	8
29687	29687	15103-924	PROVIDENCIA	924	SEMINARIO 508	7
29688	29688	15103-926	PROVIDENCIA	926	CONDELL 1415	14
29689	29689	15103-926	PROVIDENCIA	926	MALAQUIAS CONCHA 0310	5
29690	29690	15103-928	PROVIDENCIA	928	BARCELONA 2018	5
29691	29691	15103-928	PROVIDENCIA	928	GUARDIA VIEJA 181	14
29692	29692	15103-928	PROVIDENCIA	928	GUARDIA VIEJA 255	18
29693	29693	15103-928	PROVIDENCIA	928	PEDRO DE ALDIVIA 100	17
29694	29694	15103-929	PROVIDENCIA	929	BENJAMIN VICUNA MACKENNA 592	4
29695	29695	15103-930	PROVIDENCIA	930	D DE VELAZQUEZ 2141	10
29696	29696	15103-930	PROVIDENCIA	930	GUARDIA VIEJA 202	15
29697	29697	15103-930	PROVIDENCIA	930	GUARDIA VIEJA 230	6
29698	29698	15103-930	PROVIDENCIA	930	R LYON 249	10
29699	29699	15103-930	PROVIDENCIA	930	R LYON 289	4
29700	29700	15103-933	PROVIDENCIA	933	COYANCURA 2241	11

## Datos inconsistentes

# ELIMINAR Y RENOMBRAR COLUMNAS

- El dataset crudo puede contener columnas innecesarias, o con etiquetas que hacen su manipulación poco eficiente.
- Funciones relevantes:
  - `df.columns()`: permite chequear listado de columnas
  - `df.drop()`: eliminar columnas
  - `df=df[['col1','col2',...]]`: selección de un listado de columnas del dataframe
  - `df.rename(columns={'old1':'new1','old2':'new2',...})`: renombrar columnas

# TIPOS DE DATOS

- **Tipo de dato:** definición interna usada por un lenguaje de programación para entender como guardar y manipular los datos.
- Cada dato debe tener el tipo adecuado para su uso en el análisis, para evitar resultados inesperados o errores

	country	bags_60kg	tons_metric	pounds
0	Brazil	44,200,000	2,652,000	5,714,381,000
1	Vietnam	27,500,000	1,650,000	3,637,627,000
2	Colombia	13,500,000	810,000	1,785,744,000
3	Indonesia	11,000,000	660,000	1,455,050,000
4	Ethiopia	6,400,000	384,000	846,575,000
5	Honduras	5,800,000	348,000	767,208,000
6	India	5,800,000	348,000	767,208,000

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 51 entries, 0 to 50
Data columns (total 4 columns):
country      51 non-null object
bags_60kg    51 non-null object
tons_metric  51 non-null object
pounds       51 non-null object
dtypes: object(4)
memory usage: 1.7+ KB
```

```
df['pounds'].sum()
```

```
'5,714,381,0003,637,627,0001,785,744,0001,455,050,000846,575,000767,208,000767,208,000634,931,000515,881,000449,74
3,000423,287,000291,010,000257,544,000238,099,000197,357,000110,187,000105,821,000105,821,000100,795,00092,594,0007
5,398,00068,784,00068,784,00066,138,00066,138,00066,138,00052,910,00046,297,00044,312,00033,069,00026,455,00026,45
5,00026,455,00021,164,00015,873,00013,227,00013,227,00011,904,00010,582,0008,598,0005,291,0004,894,0004,761,0004,62
9,0002,777,0002,645,0002,116,0001,587,0001,322,000793,000264,000'
```

# TIPOS DE DATOS

- **Tipo de dato:** definición interna usada por un lenguaje de programación para entender como guardar y manipular los datos.
- Cada dato debe tener el tipo adecuado para su uso en el análisis, para evitar resultados inesperados o errores.
- Al leer datos, **pandas** infiere el tipo de los datos, pero frecuentemente se pueden requerir transformaciones explícitas.

Tipo de dato	Ejemplo	Python type	Pandas dtype
Texto	Nombre, apellido, dirección, país...	<code>str</code>	<code>object</code>
Enteros	Edad, cantidad de personas, Nº de transacciones, etc.	<code>int</code>	<code>int64</code>
Decimales	Temperatura, densidad, área, moneda	<code>float</code>	<code>float64</code>
Binario	Sí/no, verdadero/falso, animal/vegetal, blanco/negro...	<code>bool</code>	<code>bool</code>
Fecha	Nacimiento, defunción, envío, venta, informe, etc....	<code>datetime</code>	<code>datetime</code>
Categorías	Región, estado civil, \$/\$\$/ \$\$, casa/departamento/condominio,...etc	--	<code>category</code>

# TIPOS DE DATOS: TEXTOS Y DATOS CATEGÓRICOS

- Datos tipo texto: variables tipo “string”, pandas los identifica como “object”
- Datos categóricos: variables que definen un conjunto de estados (categorías) predefinidos.
  - Para utilizarlos en modelos de machine learning, típicamente se codifican como números.
  - Puede requerirse sólo una conversión de tipo de datos, o un procesamiento previo para codificar los datos en un conjunto dado de valores o categorías.
- Ejemplo:

Type of data	Example values	Numeric representation
Marriage Status	unmarried , married	0 , 1
Household Income Category	0-20K , 20-40K , ...	0 , 1 , ..
Loan Status	default , payed , no_loan	0 , 1 , 2

```
...    marriage_status    ...
...          3      ...
...          1      ...
...          2      ...
```

0 = Never married

1 = Married

2 = Separated

3 = Divorced

# TIPOS DE DATOS: TEXTOS Y DATOS CATEGÓRICOS

- Datos tipo texto: variables tipo “string”, pandas los identifica como “object”
- Datos categóricos: variables que definen un conjunto de estados (categorías) predefinidos.
  - Para utilizarlos en modelos de machine learning, típicamente se codifican como números.
  - Puede requerirse sólo una conversión de tipo de datos, o un procesamiento previo para codificar los datos en un conjunto dado de valores o categorías.
- Ejemplo:

```
df['marriage_status'].describe()
```

marriage\_status

...	
mean	1.4
std	0.20
min	0.00
50%	1.8 ...

```
# Convert to categorical
```

```
df["marriage_status"] = df["marriage_status"].astype('category')
```

```
df.describe()
```

marriage\_status

count	241
unique	4
top	1
freq	120

# TIPOS DE DATOS

## Funciones relevantes:

- **df.info()**: permite chequear definición de tipos de datos y valores nulos en un DataFrame.
  - **df.describe()**: entrega estadísticas de resumen del DataFrame, el output depende del tipo de dato.
  - **df.dtypes**: entrega el tipo de datos de las columnas
  - **.astype()**: conversión al tipo de dato deseado.
  - **assert()**: permite verificar una condición de entrada (por ejemplo, un tipo de dato), devuelve
    - **AssertionError ➔** si la condición es **False**
    - Nada ➔ si es **True**.

	name	price	rating	phone
0	Barrica 94	\$\$	4.5	5.627325e+09
1	Galindo	\$\$	4.0	5.622777e+10
2	In Pasta	\$\$	5.0	5.622635e+10
3	Mestizo		4.5	5.697478e+10
4	Fuente Italiana	NaN	5.0	NaN

```
1 dat=pd.read_csv('restaurants_stgo_yelp.csv')
2 dat.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 25 columns):
Unnamed: 0           1000 non-null int64
alias               1000 non-null object
categories          1000 non-null object
coordinates_latitude 999 non-null float64
coordinates_longitude 999 non-null float64
display_phone       810 non-null object
distance            1000 non-null float64
id                  1000 non-null object
image_url           948 non-null object
is_closed            1000 non-null bool
location_address1   1000 non-null object
location_address2   106 non-null object
location_address3   11 non-null object
location_city        1000 non-null object
location_country    1000 non-null object
location_display_address 1000 non-null object
location_state       1000 non-null object
location_zip_code   541 non-null float64
name                1000 non-null object
phone               810 non-null float64
price               612 non-null object
rating              1000 non-null float64
review_count         1000 non-null int64
transactions        1000 non-null object
url                 1000 non-null object
dtypes: bool(1), float64(6), int64(2), object(16)
memory usage: 188.6+ KB
```

# TIPOS DE DATOS

Funciones relevantes:

- **df.info()**: permite chequear definición de tipos de datos y valores nulos en un DataFrame.
- **df.describe()**: entrega estadísticas de resumen del DataFrame, el output depende del tipo de dato.
- **df.dtypes**: entrega el tipo de datos de las columnas
- **.astype()**: conversión al tipo de dato deseado.
- **assert()**: permite verificar una condición de entrada (por ejemplo, un tipo de dato), devuelve
  - **AssertionError ➔** si la condición es **False**
  - Nada ➔ si es **True**.

```
# Import CSV file and output header
sales = pd.read_csv('sales.csv')
sales.head(2)
```

	SalesOrderID	Revenue	Quantity
0	43659	23153\$	12
1	43660	1457\$	2

```
# Get data types of columns
sales.dtypes
```

```
SalesOrderID    int64
Revenue        object
Quantity      int64
dtype: object
```

```
# Print sum of all Revenue column
sales['Revenue'].sum()
```

```
'23153$1457$36865$32474$472$27510$16158$5694$6876$40487$'
```

```
# Remove $ from Revenue column
sales['Revenue'] = sales['Revenue'].str.strip('$')
sales['Revenue'] = sales['Revenue'].astype('int')
```

```
# Verify that Revenue is now an integer
assert sales['Revenue'].dtype == 'int'
```

# DATOS DUPLICADOS

- Corresponden a registros para los cuales se repite exactamente la misma información, para todas o algunas de las columnas.

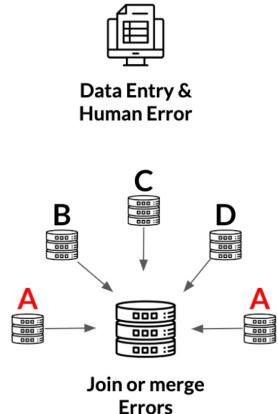
first_name	last_name	address	height	weight
Justin	Saddlemeyer	Boulevard du Jardin Botanique 3, Bruxelles	193 cm	87 kg
Justin	Saddlemeyer	Boulevard du Jardin Botanique 3, Bruxelles	193 cm	87 kg

first_name	last_name	address	height	weight
Justin	Saddlemeyer	Boulevard du Jardin Botanique 3, Bruxelles	193 cm	87 kg
Justin	Saddlemeyer	Boulevard du Jardin Botanique 3, Bruxelles	194 cm	87 kg

- Se generan típicamente por:
  - Errores humanos en el ingreso de datos
  - Errores al unir o fusionar distintas bases de datos
  - Mal diseño o errores en el proceso de recolección de datos
- Para identificar duplicados:
  - `df.duplicated()`
- Para eliminar duplicados:
  - `df.drop_duplicates()`

Todas las columnas idénticas  
→ eliminar

Algunas las columnas idénticas  
→ determinar qué valor conservar



Bugs and design errors

# DATOS DUPLICADOS

```
# Print the header  
height_weight.head()
```

```
first_name last_name address height weight  
0 Lane Reese 534-1559 Nam St. 181 64  
1 Ivor Pierce 102-3364 Non Road 168 66  
2 Roary Gibson P.O. Box 344, 7785 Nisi Ave 191 99  
3 Shannon Little 691-2550 Consectetuer Street 185 65  
4 Abdul Fry 4565 Risus St. 169 65
```

```
# Get duplicate rows  
duplicates = height_weight.duplicated()  
height_weight[duplicates]
```

```
first_name last_name address height weight  
100 Mary Colon 4674 Ut Rd. 179 75  
101 Ivor Pierce 102-3364 Non Road 168 88  
102 Cole Palmer 8366 At, Street 178 91  
103 Desirae Shannon P.O. Box 643, 5251 Consectetuer, Rd. 196 83
```

`DataFrame.duplicated(subset=None, keep='first')`

[\[source\]](#)

Return boolean Series denoting duplicate rows.

Considering certain columns is optional.

Parameters: `subset : column label or sequence of labels, optional`

Only consider certain columns for identifying duplicates, by default use all of the columns.

`keep : {'first', 'last', False}, default 'first'`

Determines which duplicates (if any) to mark.

- `first` : Mark duplicates as `True` except for the first occurrence.
- `last` : Mark duplicates as `True` except for the last occurrence.
- `False` : Mark all duplicates as `True`.

Returns: `Series`

Boolean series for each duplicated rows.

```
# Column names to check for duplication
```

```
column_names = ['first_name', 'last_name', 'address']  
duplicates = height_weight.duplicated(subset = column_names, keep = False)
```

```
# Output duplicate values
```

```
height_weight[duplicates]
```

```
first_name last_name address height weight  
1 Ivor Pierce 102-3364 Non Road 168 66  
22 Cole Palmer 8366 At, Street 178 91  
28 Desirae Shannon P.O. Box 643, 5251 Consectetuer, Rd. 195 83  
37 Mary Colon 4674 Ut Rd. 179 75  
100 Mary Colon 4674 Ut Rd. 179 75  
101 Ivor Pierce 102-3364 Non Road 168 88  
102 Cole Palmer 8366 At, Street 178 91  
103 Desirae Shannon P.O. Box 643, 5251 Consectetuer, Rd. 196 83
```

# DATOS DUPLICADOS

```
# Print the header  
height_weight.head()
```

```
first_name last_name address height weight  
0 Lane Reese 534-1559 Nam St. 181 64  
1 Ivor Pierce 102-3364 Non Road 168 66  
2 Roary Gibson P.O. Box 344, 7785 Nisi Ave 191 99  
3 Shannon Little 691-2550 Consectetuer Street 185 65  
4 Abdul Fry 4565 Risus St. 169 65
```

```
# Get duplicate rows  
duplicates = height_weight.duplicated()  
height_weight[duplicates]
```

```
first_name last_name address height weight  
100 Mary Colon 4674 Ut Rd. 179 75  
101 Ivor Pierce 102-3364 Non Road 168 88  
102 Cole Palmer 8366 At, Street 178 91  
103 Desirae Shannon P.O. Box 643, 5251 Consectetuer, Rd. 196 83
```

`DataFrame.duplicated(subset=None, keep='first')`

[\[source\]](#)

Return boolean Series denoting duplicate rows.

Considering certain columns is optional.

Parameters: `subset : column label or sequence of labels, optional`

Only consider certain columns for identifying duplicates, by default use all of the columns.

`keep : {'first', 'last', False}, default 'first'`

Determines which duplicates (if any) to mark.

- `first` : Mark duplicates as `True` except for the first occurrence.
- `last` : Mark duplicates as `True` except for the last occurrence.
- `False` : Mark all duplicates as `True`.

Returns: `Series`

Boolean series for each duplicated rows.

```
# Column names to check for duplication
```

```
column_names = ['first_name', 'last_name', 'address']  
duplicates = height_weight.duplicated(subset = column_names, keep = False)
```

```
# Output duplicate values
```

```
height_weight[duplicates].sort_values(by = 'first_name')
```

	first_name	last_name	address	height	weight
22	Cole	Palmer	8366 At, Street	178	91
102	Cole	Palmer	8366 At, Street	178	91
28	Desirae	Shannon	P.O. Box 643, 5251 Consectetuer, Rd.	195	83
103	Desirae	Shannon	P.O. Box 643, 5251 Consectetuer, Rd.	196	83
1	Ivor	Pierce	102-3364 Non Road	168	66
101	Ivor	Pierce	102-3364 Non Road	168	88
37	Mary	Colon	4674 Ut Rd.	179	75
100	Mary	Colon	4674 Ut Rd.	179	75

# DATOS DUPLICADOS: TRATAMIENTO

- **Duplicados completos** → eliminamos registros repetidos usando la función **df.drop\_duplicates()**
- **Duplicados parciales** → la decisión de qué valor conservar depende de la naturaleza de los datos, y el conocimiento que tengamos de ellos. Se evalúa caso a caso.
  - Elección informada.
  - Promedio, máximo, mínimo.
  - Para agregar mediante funciones estadísticas: **pd.groupby()**,

```
# Group by column names and produce statistical summaries
column_names = ['first_name', 'last_name', 'address']
summaries = {'height': 'max', 'weight': 'mean'}
height_weight = height_weight.groupby(by = column_names).agg(summaries).reset_index()
```

```
DataFrame.drop_duplicates(subset=None, keep='first', inplace=False, ignore_index=False)
[source]

Return DataFrame with duplicate rows removed.

Considering certain columns is optional. Indexes, including time indexes are ignored.

Parameters: subset : column label or sequence of labels, optional
Only consider certain columns for identifying duplicates, by default use all of the columns.

keep : {'first', 'last', False}, default 'first'
Determines which duplicates (if any) to keep. - first : Drop duplicates except for the first occurrence. - last : Drop duplicates except for the last occurrence. - False : Drop all duplicates.

inplace : bool, default False
Whether to drop duplicates in place or to return a copy.

ignore_index : bool, default False
If True, the resulting axis will be labeled 0, 1, ..., n - 1.
New in version 1.0.0.

Returns: DataFrame or None
DataFrame with duplicates removed or None if inplace=True.
```

```
# Column names to check for duplication
column_names = ['first_name', 'last_name', 'address']
duplicates = height_weight.duplicated(subset = column_names, keep = False)
```

```
# Output duplicate values
height_weight[duplicates].sort_values(by = 'first_name')
```

	first_name	last_name	address	height	weight
22	Cole	Palmer	8366 At, Street	178	91
102	Cole	Palmer	8366 At, Street	178	91
28	Desirae	Shannon	P.O. Box 643, 5251 Consectetuer, Rd.	195	83
103	Desirae	Shannon	P.O. Box 643, 5251 Consectetuer, Rd.	196	83
1	Ivor	Pierce	102-3364 Non Road	168	66
101	Ivor	Pierce	102-3364 Non Road	168	88
37	Mary	Colon	4674 Ut Rd.	179	75
100	Mary	Colon	4674 Ut Rd.	179	75

# DATOS FALTANTES

- Es importante analizar la data faltante para identificar problemas de recolección de datos o potenciales sesgos.
  - Para encontrar datos nulos: `pd.isnull()`, `pd.notnull()`
- Para llenar los valores faltantes, se pueden aplicar distintas estrategias, dependiendo de la naturaleza de los datos:
  - Eliminar los registros
  - Un valor constante
  - Un valor constante por columna
  - Valor medio, mediana, interpolación, algoritmos de imputación.

Argument	Description
dropna	Filter axis labels based on whether values for each label have missing data, with varying thresholds for how much missing data to tolerate.
fillna	Fill in missing data with some value or using an interpolation method such as 'ffill' or 'bfill'.
isnull	Return boolean values indicating which values are missing/NA.
notnull	Negation of <code>isnull</code> .

*Table 7-2. fillna function arguments*

Argument	Description
value	Scalar value or dict-like object to use to fill missing values
method	Interpolation; by default 'ffill' if function called with no other arguments
axis	Axis to fill on; default <code>axis=0</code>
inplace	Modify the calling object without producing a copy
limit	For forward and backward filling, maximum number of consecutive periods to fill

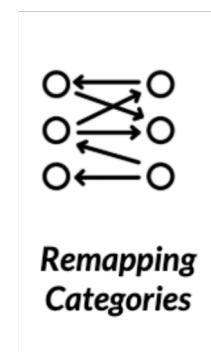
# DATOS CATEGÓRICOS

- Variables que sólo pueden adoptar un conjunto acotado de valores.
- Es frecuente **codificarlas en categorías numéricas** para su uso en modelos de machine learning.
- Pueden existir **datos con valores inconsistentes con las categorías** definidas, lo cual requiere:
  - Eliminar datos
  - Remapear categorías
  - Inferir y asignar categorías
- La identificación y corrección de inconsistencias es particular a cada dataset.

	<b>id</b>	<b>price</b>	<b>cancellation_policy</b>
0	14576	\$81,730.00	moderate
1	30950	\$19,615.00	flexible
2	47936	\$44,952.00	flexible
3	49392	\$44,952.00	moderate
4	50466	\$69,471.00	flexible



*Dropping Data*



*Remapping Categories*



*Inferring Categories*

**Funciones útiles:** `pd.unique()`,  
`pd.isin()`, `pd.map()`, `pd.drop()`,  
`pd.replace()`, `string methods`

## CÁLCULO DE VARIABLES INDICATIVAS O “DUMMIES”

- Para ciertas aplicaciones de machine learning, se **requiere transformar variables categóricas en una matriz de “dummies”** o “indicadores”.
    - Dummy → variable binaria que indica si
    - También llamado: “one hot encoding”

Human-Readable		Machine-Readable			
Pet		Cat	Dog	Turtle	Fish
Cat		1	0	0	0
Dog		0	1	0	0
Turtle		0	0	1	0
Fish		0	0	0	1
Cat		1	0	0	0

# OTRAS TÉCNICAS ÚTILES PARA LA LIMPIEZA DE DATOS

- **Validación de valores** entre distintos campos
  - Uso de múltiples columnas del dataset para validar la integridad de los datos y detectar inconsistencias/errores.
  - Ejemplos:
    - edad vs. fecha de nacimiento
    - total población vs. subtotales por sexo/edad/región, etc.
- Chequear uniformidad de los datos
  - Formatos
  - Unidades