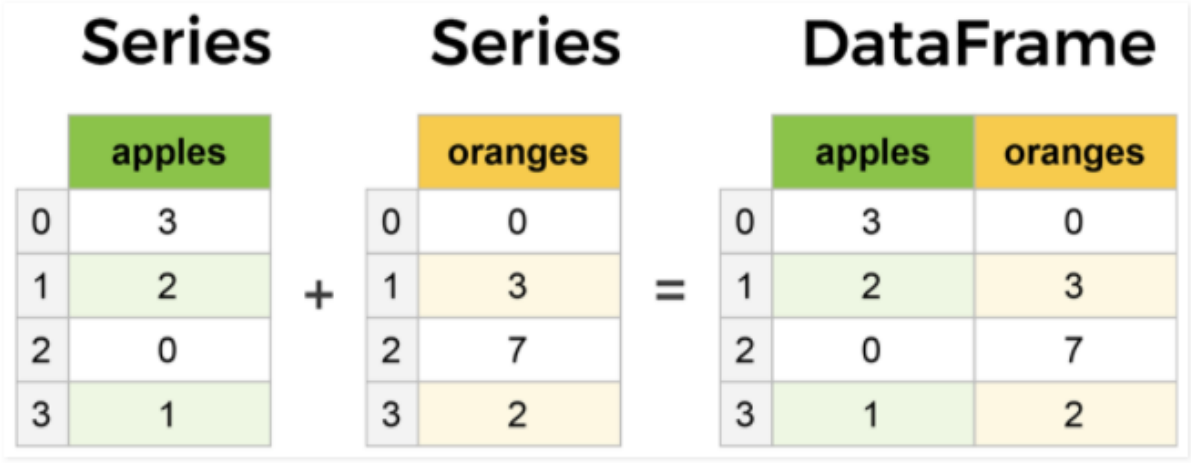
PANDAS

https://www.learndatasci.com/tutorials/python-pandas-tutorial-complete-introduction-for-beginners/ (https://www.learndatasci.com/tutorials/python-pandas-tutorial-complete-introduction-for-beginners/)




```
data = {
    'apples': [3, 2, 0, 1],
    'oranges': [0, 3, 7, 2]
}
```

DATAFRAME CONSTRUCTOR

In [14]:
```
data = {
    'apples': [3, 2, 0, 1],
    'oranges': [0, 3, 7, 2]
}
purchases = pd.DataFrame(data) #DATA FRAME CONSTRUCTOR
```

Out[14]:

	apples	oranges
0	3	0
1	2	3
2	0	7
3	1	2

CUSTOM KEYS

In [11]:
```
purchases = pd.DataFrame(data, index=['June', 'Robert', 'Lily', 'David'])
```

Out[11]:

	apples	oranges
June	3	0
Robert	2	3
Lily	0	7
David	1	2

LOCATE

In [12]:
```
```

Out[12]:
```
apples      3
oranges     0
Name: June, dtype: int64
```

Viewing your data

- .head()
- .tail()

Getting info about your data

- .info()
- .shape

Handling duplicates

To demonstrate, let's simply just double up our movies DataFrame by appending it to itself (generar una copia del dataframe para que el original no se vea afectado:

```
temp_df = movies_df.append(movies_df)
temp_df.shape
```

- drop_duplicates() // temp_df.drop_duplicates(inplace=True)--> modificación directa de la copia previa, puesto que drop_duplicates() devuelve otra copia con los duplicados eliminados.

Another important argument for drop_duplicates() is **keep**, which has three possible options:

- first: (default) Drop duplicates except for the first occurrence.
- last: Drop duplicates except for the last occurrence.
- False: Drop all duplicates.

Column cleanup

- .columns
- .rename() ((cambia el nombre a partir de un diccionario)):

```
movies_df.rename(columns={
   'Runtime (Minutes)': 'Runtime',
   'Revenue (Millions)': 'Revenue_millions'
 }, inplace=True)
```

Se puede cambiar de otras maneras, como por ejemplo:

```
    movies_df.columns = ['rank', 'genre', 'description', 'directo
r', 'actors', 'year', 'runtime',
                'rating', 'votes', 'revenue_millions', 'metascor
e']
```

Especialmente usando list comprehension:

```
    movies_df.columns = [col.lower() for col in movies_df]
```

How to work with missing values

Enontrar null values:

- df.isnull()
- df.isnull().sum()

Eliminar null values:

- df.dropna(): eliminar las filas donde haya valores null. No recomendado en muchas ocasiones, pues podemos estar eliminando datos importantes.
- df.dropna(axis=1): elimina las columnas. También peligroso.

Imputation:

There may be instances where dropping every row with a null value removes too big a chunk from your dataset, so instead we can impute that null with another value, usually the mean or the median of that column.

1. First we'll extract that column into its own variable. ex: revenue = movies_df['revenue_millions']
2. .mean():
3. With the mean, let's fill the nulls using **fillna()**. ex: revenue.fillna(revenue_mean, inplace=True)

4. Notice that by using inplace=True we have actually affected the original movies_df.

Understanding your variables

- .describe()
- movies_df['genre'].describe()
- movies_df['genre'].value_counts().head(10)

Relationships between continuous variables

By using the correlation method **.corr()** we can generate the relationship between each continuous variable:

	rank	year	runtime	rating
rank	1.000000	-0.261605	-0.221739	-0.219555
year	-0.261605	1.000000	-0.164900	-0.211219
runtime	-0.221739	-0.164900	1.000000	0.392214
rating	-0.219555	-0.211219	0.392214	1.000000
votes	-0.283876	-0.411904	0.407062	0.511537
revenue_millions	-0.252996	-0.117562	0.247834	0.189527
metascore	-0.191869	-0.079305	0.211978	0.631897

(Mirar la explicación del link porque es muy buena)

DataFrame slicing, selecting, extracting

It's important to note that, although many methods are the same, **DataFrames and Series have different attributes**, so you'll need be sure to know which type you are working with or else you will receive attribute errors.

How to use iloc

https://www.marsja.se/how-to-use-iloc-and-loc-for-indexing-and-slicing-pandas-dataframes/ (https://www.marsja.se/how-to-use-iloc-and-loc-for-indexing-and-slicing-pandas-dataframes/)

By column

- genre_col = movies_df['genre'] ---> **series**
- genre_col = movies_df[['genre']] ---> **DataFrame**

By rows

- .loc[] : locates by name. Ejemplo interesante: movies_df.loc['Prometheus':'Sing'] ((de prometheus a sing))
- .iloc[] : locates by numerical index. Ejemplo interesante: movies_df.iloc[1:4]

Conditional selections

1. condition = (movies_df['director'] == "Ridley Scott") ---> esto retorna falsos y verdaderos, no interesa mucho pro sta bn
2. movies_df[movies_df['director'] == "Ridley Scott"] ---> esto nos da las peliculas en las que unicamente ridley scott es el director
3. isin()

We can make some richer conditionals by using logical operators | for "or" and & for "and".

Applying functions

It is possible to iterate over a DataFrame or Series as you would with a list, but doing so — especially on large datasets — is very slow.

An efficient alternative is to apply() a function to the dataset. For example, we could use a function to convert movies with an 8.0 or greater to a string value of "good" and the rest to "bad" and use this transformed values to create a new column.

1. crear la función
2. movies_df["rating_category"] = movies_df["rating"].apply(rating_function) // usar lambda

OTRO TUTORIAL

https://tutswiki.com/pandas-cookbook/chapter1/ (https://tutswiki.com/pandas-cookbook/chapter1/)

In [3]:
```
# Render our plots inline
%matplotlib inline

import pandas as pd
```

Reading data from a CSV file

You can read data from a CSV file using the **read_csv function**.

In [3]:
```
broken_df = pd.read_csv('bikes.csv')
# Look at the first 3 rows
```

Out[3]:

	Date;Berri 1;Br?beuf (donn?es non disponibles);C?te-Sainte-Catherine;Maisonneuve 1;Maisonneuve 2;du Parc;Pierre-Dupuy;Rachel1;St-Urbain (donn?es non disponibles)
0	01/01/2012;35;;0;38;51;26;10;16;
1	02/01/2012;83;;1;68;153;53;6;43;
2	03/01/2012;135;;2;104;248;89;3;58;

Este file, como se puede ver, está roto. read_csv tiene una serie de opciones que nos permiten

arreglarlo:

- Change the column separator to a **;**
- Set the encoding to **'*latin1*'** (the default is '*utf8*')
- Parse the dates in the 'Date' column
- Tell it that our dates have the date first instead of the month first
- **Set the index** to be the 'Date' column

In [7]:
```
fixed_df = pd.read_csv('bikes.csv', sep=';', encoding='latin1', parse_dat
fixed_df[:3]
```

Out[7]:

Date	Berri 1	Br?beuf (donn?es non disponibles)	C?te-Sainte-Catherine	Maisonneuve 1	Maisonneuve 2	du Parc	Pierre-Dupuy	Rachel1	di
2012-01-01	35	NaN	0	38	51	26	10	16	
2012-01-02	83	NaN	1	68	153	53	6	43	
2012-01-03	135	NaN	2	104	248	89	3	58	

Seleccionar una columna

Cuando lees un CVS, obtienes un objeto llamado DataFrame formado **por filas y columnas**.

- **Para obtener una columna, accedes a ella de la misma forma con la que accedes a los elementos de un diccionario.**

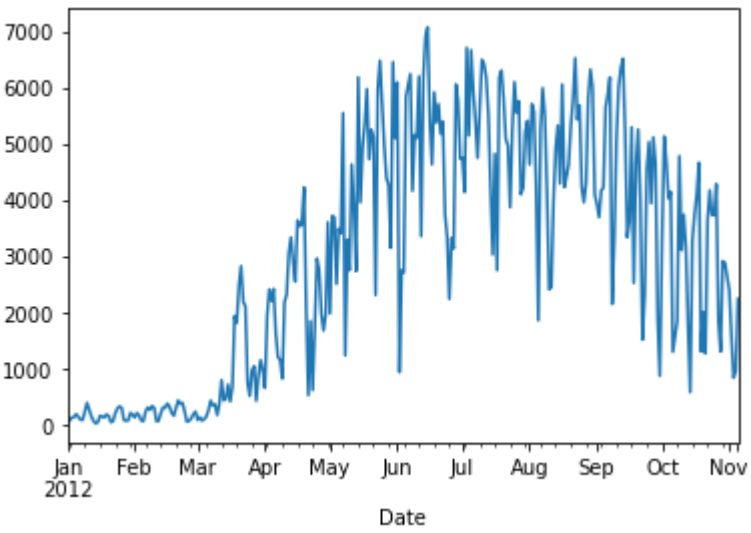
In [8]:
```
fixed_df['Berri 1']
```

Out[8]:
```
Date
2012-01-01      35
2012-01-02      83
2012-01-03     135
2012-01-04     144
2012-01-05     197
              ...
2012-11-01    2405
2012-11-02    1582
2012-11-03     844
2012-11-04     966
2012-11-05    2247
Name: Berri 1, Length: 310, dtype: int64
```

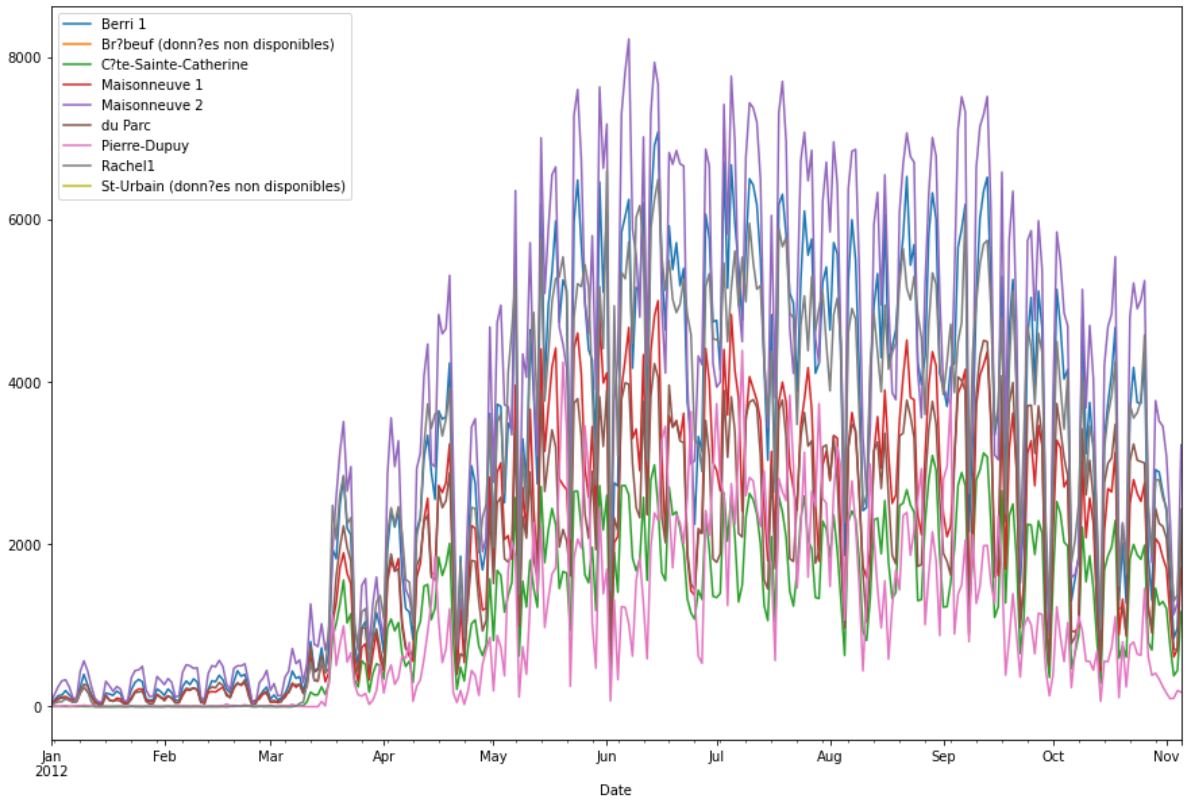
- Para mostrar el plot, utilizamos **.plot()**:

In [11]:

Out[11]: <AxesSubplot:xlabel='Date'>



In [12]:
```
#figsize como parámetro para hacer la imagen más grande
#para mostrar todas las columnas hacemos un plot general
fixed_df.plot(figsize=(15, 10))
```



SELECTING AND FINDING DESIRED DATA

Larger data set

In [13]:

```
C:\Users\elgab\anaconda3\lib\site-packages\IPython\core\interactiveshe
ll.py:3146: DtypeWarning: Columns (8) have mixed types.Specify dtype o
ption on import or set low_memory=False.
  has_raised = await self.run_ast_nodes(code_ast.body, cell_name,
```

Da un error, para ver qué significa meterse en el chapter 2 del curso.

In [14]:

Out[14]:

```
0         Noise - Street/Sidewalk
1                  Illegal Parking
2               Noise - Commercial
3                  Noise - Vehicle
4                          Rodent
                    ...
111064    Maintenance or Facility
111065             Illegal Parking
111066    Noise - Street/Sidewalk
111067         Noise - Commercial
111068           Blocked Driveway
Name: Complaint Type, Length: 111069, dtype: object
```

- **Mostrar 5 filas**:

In [15]:

Out[15]:

	Unique Key	Created Date	Closed Date	Agency	Agency Name	Complaint Type	Descriptor	Location Typ
0	26589651	10/31/2013 02:08:41 AM	NaN	NYPD	New York City Police Department	Noise - Street/Sidewalk	Loud Talking	Street/Sidewal
1	26593698	10/31/2013 02:01:04 AM	NaN	NYPD	New York City Police Department	Illegal Parking	Commercial Overnight Parking	Street/Sidewal
2	26594139	10/31/2013 02:00:24 AM	10/31/2013 02:40:32 AM	NYPD	New York City Police Department	Noise - Commercial	Loud Music/Party	Club/Ba /Restaurar
3	26595721	10/31/2013 01:56:23 AM	10/31/2013 02:21:48 AM	NYPD	New York City Police Department	Noise - Vehicle	Car/Truck Horn	Street/Sidewal
4	26590930	10/31/2013 01:53:44 AM	NaN	DOHMH	Department of Health and Mental Hygiene	Rodent	Condition Attracting Rodents	Vacant Lo

- Se puede combinar la visualización de filas y columnas:

In [16]:

```
Out[16]: 0    Noise - Street/Sidewalk
         1               Illegal Parking
         2          Noise - Commercial
         3             Noise - Vehicle
         4                      Rodent
         Name: Complaint Type, dtype: object
```

- Varias columnas **(ojo a los dos pares de claudators)**:

In [18]:

Out[18]:

	Complaint Type	Borough
0	Noise - Street/Sidewalk	QUEENS
1	Illegal Parking	QUEENS
2	Noise - Commercial	MANHATTAN
3	Noise - Vehicle	MANHATTAN
4	Rodent	MANHATTAN
...	...	...
111064	Maintenance or Facility	BROOKLYN
111065	Illegal Parking	QUEENS
111066	Noise - Street/Sidewalk	MANHATTAN
111067	Noise - Commercial	BROOKLYN
111068	Blocked Driveway	BROOKLYN

111069 rows × 2 columns

Cómo ver cuáles son los valores que más se han repetido?

- **value_counts()**

In [19]:

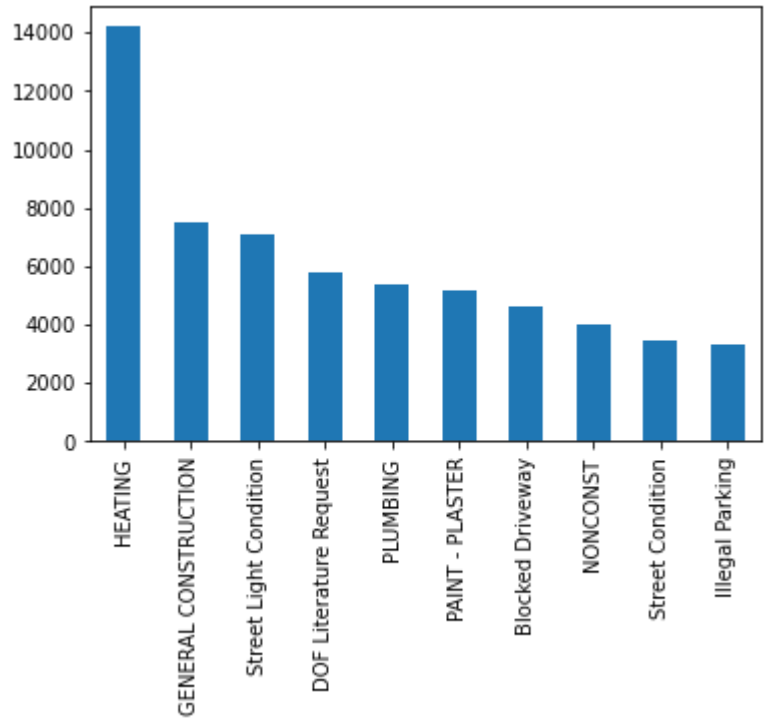
```
Out[19]: HEATING                    14200
         GENERAL CONSTRUCTION        7471
         Street Light Condition      7117
         DOF Literature Request      5797
         PLUMBING                    5373
                                     ...
         Highway Sign - Damaged         1
         Open Flame Permit              1
         Snow                           1
         Trans Fat                      1
         Tunnel Condition               1
         Name: Complaint Type, Length: 165, dtype: int64
```

In [26]:
```
#ver el TOP 10
complaint_counts = complaints['Complaint Type'].value_counts()
```

Out[26]:
```
HEATING                    14200
GENERAL CONSTRUCTION        7471
Street Light Condition      7117
DOF Literature Request      5797
PLUMBING                    5373
PAINT - PLASTER             5149
Blocked Driveway            4590
NONCONST                    3998
Street Condition            3473
Illegal Parking             3343
Name: Complaint Type, dtype: int64
```

In [27]:
```
#PLOT
```

Out[27]: <AxesSubplot:>



Obtener las filas en las que se encuentre un valor específico

Para obtener solamente las filas donde el tipo de "Complaint Type" sea el de "Noise - etc", utilizamos una condición como la que vemos a continuación:

```
complaints['Complaint Type'] == "Noise - Street/Sidewalk"
```

El operador de equivalencia (==) juega como "muestrame ahí solo donde sea igual a tal".

Para mostrar dos de estas a la vez, podemos utilizar **&** como se ve en:

```
complaints[is_noise & in_brooklyn]
```

In [28]:
```
is_noise = complaints['Complaint Type'] == "Noise - Street/Sidewalk"
in_brooklyn = complaints['Borough'] == "BROOKLYN"
```

Out[28]:

	Unique Key	Created Date	Closed Date	Agency	Agency Name	Complaint Type	Descriptor	Location Ty
31	26595564	10/31/2013 12:30:36 AM	NaN	NYPD	New York City Police Department	Noise - Street/Sidewalk	Loud Music/Party	Street/Sidew
49	26595553	10/31/2013 12:05:10 AM	10/31/2013 02:43:43 AM	NYPD	New York City Police Department	Noise - Street/Sidewalk	Loud Talking	Street/Sidew
109	26594653	10/30/2013 11:26:32 PM	10/31/2013 12:18:54 AM	NYPD	New York City Police Department	Noise - Street/Sidewalk	Loud Music/Party	Street/Sidew
236	26591992	10/30/2013 10:02:58 PM	10/30/2013 10:23:20 PM	NYPD	New York City Police Department	Noise - Street/Sidewalk	Loud Talking	Street/Sidew
370	26594167	10/30/2013 08:38:25 PM	10/30/2013 10:26:28 PM	NYPD	New York City Police Department	Noise - Street/Sidewalk	Loud Music/Party	Street/Sidew

Añadiendo **.value** obtenemos el array que hay detrás de la serie de Pandas. Para más info ver el ejemplo del curso.

.copy() para hacer una copia de un data.

.index para acceder al índice. --> se pueden modificar los nombres de los índices así:

```
weekday_counts.index = ['Monday', 'Tuesday', 'Wednesday', 'Thursd
ay', 'Friday', 'Saturday', 'Sunday']
```

.day // .weekday

.groupby() sirve para agrupar: https://pandas.pydata.org/pandas-docs/stable/user_guide/groupby.html (https://pandas.pydata.org/pandas-docs/stable/user_guide/groupby.html)

```
berri_bikes.groupby('weekday').aggregate(sum)
```

"Group the rows by weekday and then add up all the values with the same weekday."

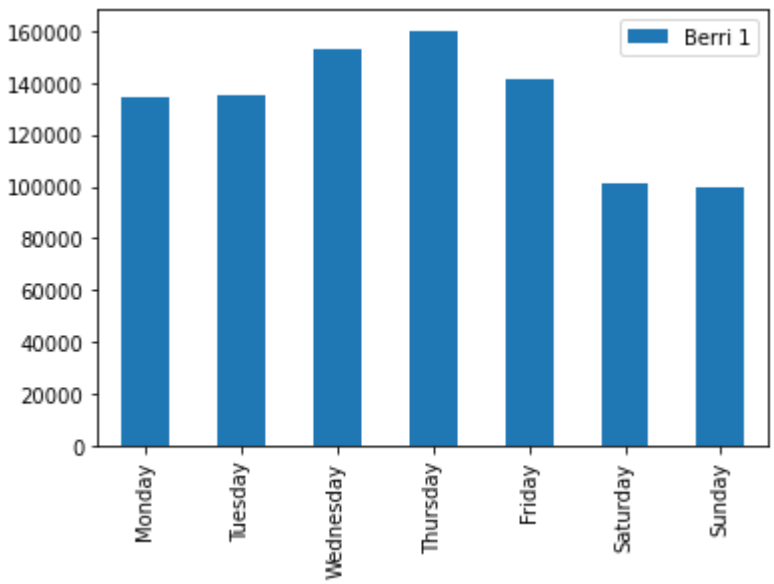
In [33]:
```
bikes = pd.read_csv('bikes.csv',
                    sep=';', encoding='latin1',
                    parse_dates=['Date'], dayfirst=True,
                    index_col='Date')
# Add the weekday column
berri_bikes = bikes[['Berri 1']].copy()
berri_bikes.loc[:,'weekday'] = berri_bikes.index.weekday
```

Out[33]:

Date	Berri 1	weekday
2012-01-01	35	6
2012-01-02	83	0
2012-01-03	135	1
2012-01-04	144	2
2012-01-05	197	3

In [32]:
```
# Add up the number of cyclists by weekday, and plot!
weekday_counts = berri_bikes.groupby('weekday').aggregate(sum)
weekday_counts.index = ['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Fr
weekday_counts.plot(kind='bar')
```

Out[32]: <AxesSubplot:>



WEB SCRAPING

Sirve para extraer info de páginas web.

In [8]:
```
url_template = "http://climate.weather.gc.ca/climate_data/bulk_data_e.htm
```

En esta variable tendremos la url de la web del histórico de datos sobre el clima de Canadá.

In [9]:
```
url = url_template.format(month=3, year=2012)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-9-7667bde91c9f> in <module>
      1 url = url_template.format(month=3, year=2012)
----> 2 weather_mar2012 = pd.read_csv(url, index_col='Date/Time', parse_dates=True)

~\anaconda3\lib\site-packages\pandas\io\parsers.py in read_csv(filepath_or_buffer, sep, delimiter, header, names, index_col, usecols, squeeze, prefix, mangle_dupe_cols, dtype, engine, converters, true_values, false_values, skipinitialspace, skiprows, skipfooter, nrows, na_values, keep_default_na, na_filter, verbose, skip_blank_lines, parse_dates, infer_datetime_format, keep_date_col, date_parser, dayfirst, cache_dates, iterator, chunksize, compression, thousands, decimal, lineterminator, quotechar, quoting, doublequote, escapechar, comment, encoding, dialect, error_bad_lines, warn_bad_lines, delim_whitespace, low_memory, memory_map, float_precision)
    684     )
    685
--> 686     return _read(filepath_or_buffer, kwds)
    687
    688

~\anaconda3\lib\site-packages\pandas\io\parsers.py in _read(filepath_or_buffer, kwds)
    450
```

Así hemos obtenido el clima en Marzo de 2012 del template previo y lo hemos leído como una base de datos normal.

STRING OPERATIONS

resample()

PLATZI

In [1]:
```
import pandas as pd
sr = pd.Series([1, 2, 3, 4, 59])
```

Out[1]:
```
0     1
1     2
2     3
3     4
4    59
dtype: int64
```

In [5]:
```

```

Out[5]:
```
array([ 1,  2,  3,  4, 59], dtype=int64)
```

In [8]:
```
sr = pd.Series([1, 2, 3, 4, 59], index=['a', 'b', 'c', 'd', 'e'])
```

Out[8]:
```
a     1
b     2
c     3
d     4
e    59
dtype: int64
```

In [9]:

Out[9]:
```
Index(['a', 'b', 'c', 'd', 'e'], dtype='object')
```

In [2]:
```
import numpy as np
dict_data = {
    'edad': [10,9,13,14,12,11,12],
    'cm': [115,110,130,155,125,120,125],
    'pais':['co','mx','co','mx','mx','ch','ch'],
    'genero':['M','F','F','M','M','M','F'],
    'Q1':[5,10,8,np.nan,7,8,3],
    'Q2':[7,9,9,8,8,8,9]
}

df = pd.DataFrame(dict_data, index = ['ana','benito','camilo','daniel','e
df
```

Out[2]:

	edad	cm	pais	genero	Q1	Q2
ana	10	115	co	M	5.0	7
benito	9	110	mx	F	10.0	9
camilo	13	130	co	F	8.0	9
daniel	14	155	mx	M	NaN	8
erika	12	125	mx	M	7.0	8
fabian	11	120	ch	M	8.0	8
gabriela	12	125	ch	F	3.0	9

In [12]:

Out[12]:

	edad	cm	Q1
ana	10	115	5.0
benito	9	110	10.0
camilo	13	130	8.0
daniel	14	155	NaN
erika	12	125	7.0
fabian	11	120	8.0
gabriela	12	125	3.0

Ojo a esto: acceder a los datos de ana solamente:

In [13]:

```
Out[13]: edad     10
         cm      115
         Q1        5
         Name: ana, dtype: object
```

In [15]:

```
#un valor específico
```

```
Out[15]: nan
```

Query

In [16]:

```
# QUERY
```

Out[16]:

	edad	cm	pais	genero	Q1	Q2
camilo	13	130	co	F	8.0	9
daniel	14	155	mx	M	NaN	8

Comparar

In [17]:

```
#COMPARAR
```

Out[17]:

	edad	cm	pais	genero	Q1	Q2
ana	10	115	co	M	5.0	7
camilo	13	130	co	F	8.0	9
erika	12	125	mx	M	7.0	8
fabian	11	120	ch	M	8.0	8
gabriela	12	125	ch	F	3.0	9

Dtypes

In [22]:

```
#dtypes
```

```
Out[22]: edad       int64
         cm         int64
         pais      object
         genero    object
         Q1       float64
         Q2         int64
         dtype: object
```

In [4]:

```
#capítulo 10: Estructuras de dataframes en detalle (min 3:20)
df_meteo = pd.read_csv('Meteorite_Landings.csv', sep = ',', encoding = 'u
```

```
df_meteo.head()
```

Out[4]:

	name	id	nametype	recclass	mass (g)	fall	year	reclat	reclong	GeoLo
0	Aachen	1	Valid	L5	21.0	Fell	01/01/1880 12:00:00 AM	50.77500	6.08333	(5 6.(
1	Aarhus	2	Valid	H6	720.0	Fell	01/01/1951 12:00:00 AM	56.18333	10.23333	(56. 10.2
2	Abee	6	Valid	EH4	107000.0	Fell	01/01/1952 12:00:00 AM	54.21667	-113.00000	(54.2 -
3	Acapulco	10	Valid	Acapulcoite	1914.0	Fell	01/01/1976 12:00:00 AM	16.88333	-99.90000	(16.8
4	Achiras	370	Valid	L6	780.0	Fell	01/01/1902 12:00:00 AM	-33.16667	-64.95000	(-33. -

In [5]:

Out[5]:
```
name            object
id               int64
nametype        object
recclass        object
mass (g)       float64
fall            object
year            object
reclat         float64
reclong        float64
GeoLocation     object
dtype: object
```

nunique() : mirar variedad de elementos en cada variable

In [6]:

Out[6]:
```
name           45716
id             45716
nametype           2
recclass         466
mass (g)       12576
fall               2
year             266
reclat         12738
reclong        14640
GeoLocation    17100
dtype: int64
```

CONVERTIR A VARIABLES CATEGÓRICAS

In [7]:
```
df_meteo[['fall', 'nametype']] = df_meteo[['fall', 'nametype']].astype('c
df_meteo.dtypes
```

Out[7]:
```
name             object
id                int64
nametype       category
recclass         object
mass (g)        float64
fall           category
year             object
reclat          float64
reclong         float64
GeoLocation      object
dtype: object
```

unique()

In [8]:

Out[8]:
```
['Fell', 'Found']
Categories (2, object): ['Fell', 'Found']
```

In [9]:

Out[9]:
```
Found    44609
Fell      1107
Name: fall, dtype: int64
```

CONVERTIR A DUMMY VARIABLES

In [10]:

Out[10]:

	Fell	Found
0	1	0
1	1	0
2	1	0
3	1	0
4	1	0

In [11]:
```
#crear columnas dummies en el dataframe
df_meteo[['fell', 'found']] = pd.get_dummies(df_meteo['fall'])
```

Out[11]:

	name	id	nametype	recclass	mass (g)	fall	year	reclat	reclong	GeoLo
0	Aachen	1	Valid	L5	21.0	Fell	01/01/1880 12:00:00 AM	50.77500	6.08333	(5 6.(

	name	id	nametype	recclass	mass (g)	fall	year	reclat	reclong	GeoLo
1	Aarhus	2	Valid	H6	720.0	Fell	01/01/1951 12:00:00 AM	56.18333	10.23333	(56. 10.
2	Abee	6	Valid	EH4	107000.0	Fell	01/01/1952 12:00:00 AM	54.21667	-113.00000	(54. -
3	Acapulco	10	Valid	Acapulcoite	1914.0	Fell	01/01/1976 12:00:00 AM	16.88333	-99.90000	(16.

CONVERTIR A DATETIME

In [12]:
```
df_meteo
```

Out[12]:
```
0         01/01/1880 12:00:00 AM
1         01/01/1951 12:00:00 AM
2         01/01/1952 12:00:00 AM
3         01/01/1976 12:00:00 AM
4         01/01/1902 12:00:00 AM
                   ...
45711     01/01/1990 12:00:00 AM
45712     01/01/1999 12:00:00 AM
45713     01/01/1939 12:00:00 AM
45714     01/01/2003 12:00:00 AM
45715     01/01/1976 12:00:00 AM
Name: year, Length: 45716, dtype: object
```

In [13]:
```
df_meteo.year = pd.to_datetime(
                               df_meteo.year,
                               errors = 'coerce', #si detecta un formato que
                               format = '%m/%d/%Y %H:%M:%S %p'
                               )
df_meteo.dtypes
```

Out[13]:
```
name                    object
id                       int64
nametype              category
recclass                object
mass (g)               float64
fall                  category
year           datetime64[ns]
reclat                 float64
reclong                float64
GeoLocation             object
fell                     uint8
found                    uint8
dtype: object
```

Ahora puedo acceder a las horas porque datetime tiene una serie de atributos relacionados al tiempo:

In [26]:

```
Out[26]: 0         12.0
         1         12.0
         2         12.0
         3         12.0
         4         12.0
                   ...
         45711     12.0
         45712     12.0
         45713     12.0
         45714     12.0
         45715     12.0
         Name: year, Length: 45716, dtype: float64
```

RENAME

In [17]:

```
df_meteo.rename(columns={'mass (g)':'mass'}, inplace=True) #inplace hace
df_meteo
```

Out[17]:

	name	id	nametype	recclass	mass	fall	year	reclat	reclo
0	Aachen	1	Valid	L5	21.0	Fell	1880-01-01 12:00:00	50.77500	6.083
1	Aarhus	2	Valid	H6	720.0	Fell	1951-01-01 12:00:00	56.18333	10.233
2	Abee	6	Valid	EH4	107000.0	Fell	1952-01-01 12:00:00	54.21667	-113.000
3	Acapulco	10	Valid	Acapulcoite	1914.0	Fell	1976-01-01 12:00:00	16.88333	-99.900
4	Achiras	370	Valid	L6	780.0	Fell	1902-01-01 12:00:00	-33.16667	-64.950
...	...	...	...	...	...	...	...	...	
45711	Zillah 002	31356	Valid	Eucrite	172.0	Found	1990-01-01 12:00:00	29.03700	17.018
45712	Zinder	30409	Valid	Pallasite, ungrouped	46.0	Found	1999-01-01 12:00:00	13.78333	8.966
45713	Zlin	30410	Valid	H4	3.3	Found	1939-01-01 12:00:00	49.25000	17.666
45714	Zubkovsky	31357	Valid	L6	2167.0	Found	2003-01-01 12:00:00	49.78917	41.504
45715	Zulu Queen	30414	Valid	L3.7	200.0	Found	1976-01-01 12:00:00	33.98333	-115.683

45716 rows × 12 columns

HACER UNA COPIA DEL DF, IMPORTANTE HACER DEEP=TRUE

In [19]:
```
df  = df_meteo.copy(deep=True) #deep hace que no guarde una copia referen
```

tomar filas de 100 en 100

In [27]:

Out[27]:

	name	id	nametype	recclass	mass	fall	year	reclat	reclo
0	Aachen	1	Valid	L5	21.00	Fell	1880-01-01 12:00:00	50.77500	6.083
100	Benton	5026	Valid	LL6	2840.00	Fell	1949-01-01 12:00:00	45.95000	-67.55(
200	Chetrinahatti	5344	Valid	Stone-uncl	72.00	Fell	1880-01-01 12:00:00	14.50000	76.50(
300	Fermo	10091	Valid	H3-5	10200.00	Fell	1996-01-01 12:00:00	43.18111	13.753
400	Innisfree	12039	Valid	L5	4576.00	Fell	1977-01-01 12:00:00	53.41500	-111.337
...	...	...	...	...	...	...	...	...	
45300	Yamato 983643	40265	Valid	LL-melt breccia	4.61	Found	1998-01-01 12:00:00	0.00000	0.00(
45400	Yamato 983775	40397	Valid	H5	6.97	Found	1998-01-01 12:00:00	0.00000	0.00(
45500	Yamato 983885	30344	Valid	Lunar (anorth)	289.71	Found	1999-01-01 12:00:00	-71.56283	36.005
45600	Yamato 984043	40663	Valid	H5	21.69	Found	1998-01-01 12:00:00	0.00000	0.00(
45700	Zaragoza	48916	Valid	Iron, IVA-an	162000.00	Found	NaT	41.65000	-0.866

458 rows × 12 columns

APPLY / LAMBDA

In [28]:
```
def fun_1(x):
  y = x**2 + 1
  return y
```

Out[28]:

```
0         4.420000e+02
1         5.184010e+05
```

In [29]:

Out[29]:
```
0           221.0
1           920.0
2        107200.0
3          2114.0
4           980.0
            ...
45711       372.0
45712       246.0
45713       203.3
45714      2367.0
45715       400.0
Name: mass, Length: 45716, dtype: float64
```

ISIN()

In [37]:
```
filtro = df_meteo.name.isin(['Aachen', 'Benton']) #isin busca los valores
filtro
```

Out[37]:
```
0         True
1        False
2        False
3        False
4        False
         ...
45711    False
45712    False
45713    False
45714    False
45715    False
Name: name, Length: 45716, dtype: bool
```

In [38]:
```
df_meteo[filtro] #aplicamos el filtro sobre el df y obtenemos los valores
```

Out[38]:

	name	id	nametype	recclass	mass	fall	year	reclat	reclong	GeoLocation
0	Aachen	1	Valid	L5	21.0	Fell	1880-01-01 12:00:00	50.775	6.08333	(50.775, 6.08333)
100	Benton	5026	Valid	LL6	2840.0	Fell	1949-01-01 12:00:00	45.950	-67.55000	(45.95, -67.55)

UNSTACK()

In [ ]:

In [ ]:

In [ ]:

RESHAPING DATAFRAMES

Pivot, melt, stack, unstack:

- https://pandas.pydata.org/pandas-docs/stable/user_guide/reshaping.html (https://pandas.pydata.org/pandas-docs/stable/user_guide/reshaping.html)
- https://towardsdatascience.com/reshaping-pandas-dataframes-9812b3c1270e (https://towardsdatascience.com/reshaping-pandas-dataframes-9812b3c1270e)

Wide_to_long: https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.wide_to_long.html (https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.wide_to_long.html)

In [ ]:

TIPOS DE DATOS EN LA LIBRERÍA PANDAS

1. **Object**: Used for text or alpha-numeric values.
2. **Int64**: Used for Integer numbers.
3. **Float64**: Used for floating-point numbers.
4. **Bool**: Used for True/False values.
5. **Datetime64**: Used for date and time values.
6. **Timedelta[ns]**: Used for differences between two datetimes.
7. **Category**: Used for a list of text values.

REGEX + PANDAS

https://kanoki.org/2019/11/12/how-to-use-regex-in-pandas/ (https://kanoki.org/2019/11/12/how-to-use-regex-in-pandas/)

GEOIP2

- Instalación & getting started: https://www.youtube.com/watch?v=1-8eVrnmDa8&list=PLBiSyPZ0731-udEYaWcg_RTdk5dNfn3ap&index=1 (https://www.youtube.com/watch?v=1-8eVrnmDa8&list=PLBiSyPZ0731-udEYaWcg_RTdk5dNfn3ap&index=1)
- Database de Maxmind: GeoLite2-City

```
#importar paquete
import geoip2.database

#el archivo leído por el Reader es la base de datos "GeoLite
2-City" que se había descargado de Maxmind
reader = geoip2.database.Reader('C:\\Users\\elgab\\Desktop\\
NUBE MEGA\\PROGRAMACIÓN\\2020-2021\\IT_Academy\\itinerario D
S\\bases_datos\\GeoLite2-City_20210629\\GeoLite2-City.mmdb')

response = reader.city('66.249.76.216')

#métodos
response.country.name
response.city.name (no siempre está disponible)
response.location.latitude
response.location.longitude
---
```

GEOPANDAS

- Crear un geopandas df a partir de un df de pandas: https://geopandas.org/gallery/create_geopandas_from_pandas.html (https://geopandas.org/gallery/create_geopandas_from_pandas.html)
- Instalación: https://geopandas.org/getting_started.html (https://geopandas.org/getting_started.html)
- Tutorial de uso: https://geopandas.org/getting_started/introduction.html (https://geopandas.org/getting_started/introduction.html)
- Plotting con cartopy y geopandas: https://geopandas.org/gallery/cartopy_convert.html (https://geopandas.org/gallery/cartopy_convert.html)

Convertir datos de una columna o más

https://thecodingbot.com/converting-datatype-of-one-or-more-column-in-a-pandas-dataframe/ (https://thecodingbot.com/converting-datatype-of-one-or-more-column-in-a-pandas-dataframe/)

Extra

How to Concatenate Column Values in Pandas DataFrame: https://datatofish.com/concatenate-values-python/ (https://datatofish.com/concatenate-values-python/)

- .describe()
- .concat(): encadenar series
- .sample(n)
- .unique()
- **.dtype() / .astype()**

- pd.read_table(sep=, encoding=, header=)
- ROW DROPPING: https://www.datasciencelearner.com/how-to-drop-rows-in-pandas/ (https://www.datasciencelearner.com/how-to-drop-rows-in-pandas/)
- COLUMN DROPPING: https://www.geeksforgeeks.org/how-to-drop-one-or-multiple-columns-in-pandas-dataframe/ (https://www.geeksforgeeks.org/how-to-drop-one-or-multiple-columns-in-pandas-dataframe/)
- Calcular %: https://www.geeksforgeeks.org/how-to-calculate-the-percentage-of-a-column-in-pandas/ (https://www.geeksforgeeks.org/how-to-calculate-the-percentage-of-a-column-in-pandas/)
- Append rows: https://pythonexamples.org/pandas-dataframe-add-append-row/ (https://pythonexamples.org/pandas-dataframe-add-append-row/)
- AÑADIR COLUMNAS: https://re-thought.com/how-to-add-new-columns-in-a-dataframe-in-pandas/ (https://re-thought.com/how-to-add-new-columns-in-a-dataframe-in-pandas/)
- LOC Y ILOC: https://www.marsja.se/how-to-use-iloc-and-loc-for-indexing-and-slicing-pandas-dataframes/#How_to_Use_Pandas_loc (https://www.marsja.se/how-to-use-iloc-and-loc-for-indexing-and-slicing-pandas-dataframes/#How_to_Use_Pandas_loc)
- Crear empty dataframe: https://thispointer.com/pandas-how-to-create-an-empty-dataframe-and-append-rows-columns-to-it-in-python/ (https://thispointer.com/pandas-how-to-create-an-empty-dataframe-and-append-rows-columns-to-it-in-python/)
- TIMESERIES: https://jakevdp.github.io/PythonDataScienceHandbook/03.11-working-with-time-series.html (https://jakevdp.github.io/PythonDataScienceHandbook/03.11-working-with-time-series.html)

How to Find Unique Values in a Column

https://www.statology.org/pandas-unique-values-in-column/ (https://www.statology.org/pandas-unique-values-in-column/)

Lambda para dummies

https://www.analyticsvidhya.com/blog/2021/06/anonymous-or-lambda-functions-in-python-a-beginners-guide/ (https://www.analyticsvidhya.com/blog/2021/06/anonymous-or-lambda-functions-in-python-a-beginners-guide/)

WORKING WITH MISSING DATA

https://pandas-docs.github.io/pandas-docs-travis/user_guide/missing_data.html (https://pandas-docs.github.io/pandas-docs-travis/user_guide/missing_data.html)

WIDE VS LONG DATAFRAME

https://towardsdatascience.com/reshape-r-dataframes-wide-to-long-with-melt-tutorial-and-visualization-ddf130cd9299 (https://towardsdatascience.com/reshape-r-dataframes-wide-to-long-with-melt-tutorial-and-visualization-ddf130cd9299)