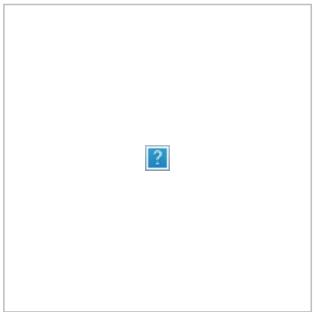


IMT 2200 - Introducción a Ciencia de Datos

Pontificia Universidad Católica de Chile
Instituto de Ingeniería Matemática y Computacional
Semestre 2023-1

Profesor: Rodrigo A. Carrasco

Clase 03: Trabajo con Datos Estructurados

Este ejercicio busca que los estudiantes usen algunas librerías para importar datos y luego aprendan algunos comandos de Pandas para analizar e inspeccionar los datos.

1. Datos para los ejemplos

Usaremos dos conjuntos de datos para este Notebook:

1. Datos de viajes en Taxi en la ciudad de Nueva York:
   El proyecto *Open Data* de la Ciudad de Nueva York nos da acceso a una gran cantiadad de datos del quehacer de la ciudad. En este caso usaremos el sitio con los datos de viajes en Taxi, disponibles en https://www.nyc.gov/site/tlc/about/tlc-trip-record-data.page En la carpeta '`data`' está disponible la base de datos de todos los viajes realizados en Mayo de 2023. El archivo se llama '`yellow_tripdata_2023-05.parquet`'. El formato PARQUET, que es open-source desarrollado por Apache, es un formato eficiente para almacenar y leer bases de datos de gran tamaño. Para poder leer este formato desde Python, deberán instalar una nueva librería llamada ' `pyarrow` '. Para instalarla use el comando: `> conda install pyarrow`
2. Datos de casos de COVID en Chile:
   Durante la pandemia, el Ministerio de Ciencia y Tecnología, con el apoyo de múltiples grupos de investigación y universidades, armó un repositorio abierto de datos sobre la situación de la pandemia en Chile. Los datos a utilizar en este ejemplo provienen del repositorio GitHub **Datos-COVID19** disponible en https://github.com/MinCiencia/Datos-COVID19. Estaremos usando el "*Data Product 1 - Casos totales por comuna incremental: el archivo Covid-19.csv contiene las columnas 'Región', 'Código Región', 'Comuna', 'Código comuna', 'Población', múltiples columnas correspondientes a '[fecha]', y una columna 'Tasa'. Estas últimas columnas, '[fecha]', contienen los 'Casos Confirmados' reportados por el Ministerio de Salud de Chile en cada una de las fechas que se indican en las respectivas columnas.*" https://github.com/MinCiencia/Datos-COVID19/tree/master/output/producto1 En la carpeta '`data`' está disponible la base de datos de todos los casos confirmados hasta enero de 2023.

2. Librerías

El trabajo de esta clase se centrará en el uso de NumPy y Pandas, pero necesitamos otras librerías adicionales para leer los datos y graficar información.

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import pyarrow.parquet as pq
```

3. Navegación en carpetas y acceso a datos

Para improtar los archvios de datos, necesitamos identificar en qué directorio están guardados en nuestro sistema y en qué directorio estamos trabajando ("working directory").

Algunos comandos importantes:

- `%ls` : lista el contenido del directorio actual command lists all content in the current working directory.
- `%cd 'subdirectorio'` : permite cambiar la ubicación actual a 'subdirectorio'
- `%cd ..` : permite navegar hacia atrás al directorio superior del actual
- `%pwd` : entrega la ruta del directorio actual

In [2]:
```
%pwd
```

Out[2]:
```
'C:\\Users\\rodri\\OneDrive\\Personal\\UC\\courses\\IMC pregrado\\imt2200 introducción a ciencia de datos\\lectures\\clase 03'
```

In [3]:
```
%ls
```

```
 Volume in drive C has no label.
 Volume Serial Number is 104F-17BC

 Directory of C:\Users\rodri\OneDrive\Personal\UC\courses\IMC pregrado\imt2200 introducci¢n a ciencia de datos\
lectures\clase 03

08/21/2023  11:47 AM    <DIR>          .
08/21/2023  11:47 AM    <DIR>          ..
08/21/2023  11:43 AM    <DIR>          .ipynb_checkpoints
08/16/2023  04:34 PM         1,598,291 3 Tipos de datos.pdf
08/16/2023  04:33 PM        16,102,192 3 Tipos de datos.pptx
08/16/2023  06:39 PM    <DIR>          data
08/21/2023  11:46 AM            87,826 IMT2200_Clase_03.ipynb
               3 File(s)     17,788,309 bytes
               4 Dir(s)  488,429,350,912 bytes free
```

In [4]:
```
%cd data
```

```
C:\Users\rodri\OneDrive\Personal\UC\courses\IMC pregrado\imt2200 introducción a ciencia de datos\lectures\clase
03\data
```

4. Estudio de datos de viajes en NYC

El objetivo de este ejercicio es entender cuántos viajes ocurrieron en la ciudad de Nueva York durante mayo de 2023 y cuáles son los lugares más relevantes para tomar pasajeros.

4.1 Importar datos

El primer paso será importar los datos, que están en un archivo en formato Parquet, y pasarlos a un DataFrame de Pandas.

In [5]:
```
# leer la base parquet
trips = pq.read_table('yellow_tripdata_2023-05.parquet')
```

4.2 `pandas` y DataFrames

El método `<x>.to_pandas()` permite transformar el archivo Parquet en un DataFrame de Pandas. Un DataFrame es una base de datos estructurada, que posee columnas y filas con la información relevante. Una referencia rápida a varias cosas que se pueden hacer con DataFrames está disponible acá: https://pandas.pydata.org/pandas-docs/stable/user_guide/10min.html

In [6]:
```
# transformar a dataframe de pandas
trips = trips.to_pandas()
```

4.3 Leer e inspeccionar un DataFrame

Un DataFrame de Pandas posee una serie de métodos que permiten revisar los datos contenidos en el DataFrame. Algunos de los más relevantes los vemos a continuación.

In [7]:
```
trips.head()
```

Out[7]:

	VendorID	tpep_pickup_datetime	tpep_dropoff_datetime	passenger_count	trip_distance	RatecodeID	store_and_fwd_flag	PULocationID	DO
0	1	2023-05-01 00:33:13	2023-05-01 00:53:01	0.0	7.80	1.0	N	138	
1	1	2023-05-01 00:42:49	2023-05-01 01:11:18	2.0	8.10	1.0	N	138	
2	1	2023-05-01 00:56:34	2023-05-01 01:13:39	2.0	9.10	1.0	N	138	
3	2	2023-05-01 00:00:52	2023-05-01 00:20:12	1.0	8.21	1.0	N	138	
4	1	2023-05-01 00:05:50	2023-05-01 00:19:41	0.0	7.90	1.0	N	138	

In [8]:
```
trips.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3513649 entries, 0 to 3513648
Data columns (total 19 columns):
 #   Column                 Dtype
---  ------                 -----
 0   VendorID               int32
 1   tpep_pickup_datetime   datetime64[ns]
 2   tpep_dropoff_datetime  datetime64[ns]
 3   passenger_count        float64
 4   trip_distance          float64
 5   RatecodeID             float64
 6   store_and_fwd_flag     object
 7   PULocationID           int32
 8   DOLocationID           int32
 9   payment_type           int64
 10  fare_amount            float64
 11  extra                  float64
 12  mta_tax                float64
 13  tip_amount             float64
 14  tolls_amount           float64
 15  improvement_surcharge  float64
 16  total_amount           float64
 17  congestion_surcharge   float64
 18  Airport_fee            float64
dtypes: datetime64[ns](2), float64(12), int32(3), int64(1), object(1)
memory usage: 469.1+ MB
```

In [9]:
```
# nombres de las columnas
trips.columns
```

Out[9]:
```
Index(['VendorID', 'tpep_pickup_datetime', 'tpep_dropoff_datetime',
       'passenger_count', 'trip_distance', 'RatecodeID', 'store_and_fwd_flag',
       'PULocationID', 'DOLocationID', 'payment_type', 'fare_amount', 'extra',
       'mta_tax', 'tip_amount', 'tolls_amount', 'improvement_surcharge',
       'total_amount', 'congestion_surcharge', 'Airport_fee'],
      dtype='object')
```

In [10]:
```
# obtener los datos de una columna
trips["trip_distance"]
```

Out[10]:
```
0           7.80
1           8.10
2           9.10
3           8.21
4           7.90
           ...
3513644     2.18
3513645     0.67
3513646     0.00
3513647    13.95
3513648     3.37
Name: trip_distance, Length: 3513649, dtype: float64
```

In [11]:
```
# obtener una fila del DataFrame
trips["trip_distance"][1]
```

Out[11]:
```
8.1
```

4.4 Contestando la pregunta

A continuación haremos una serie de cálculos para contestar nuestra pregunta inicial: ¿Cuál es el mejor lugar en NYC para tomar pasajeros?

Cada columna de un DataFrame es una Serie, que corresponde a un arreglo 1-D con una etiqueta. Por lo tanto, en el caso de columnas con datos numéricos, podemos aplicar todas las operaciones matemáticas disponibles en `numpy`: https://numpy.org/doc/stable/reference/routines.math.html

Partamos con la cantidad total de pasajeros que viajaron en Mayo de 2023.

In [12]:
```
# suma de la columna de pasajeros
total_pasajeros = trips["passenger_count"].sum()
print(f'La cantidad total de pasajeros fue de {total_pasajeros}')
```

```
La cantidad total de pasajeros fue de 4636025.0
```

In [13]:
```
# ahora veamos esa suma por cada zona
trips_by_loc = trips[["PULocationID", "passenger_count"]].groupby("PULocationID").sum()
trips_by_loc.head()
```

Out[13]:

PULocationID	passenger_count
1	806.0
2	1.0
3	61.0
4	4777.0
5	58.0

In [14]:
```
max_loc = trips_by_loc.idxmax()
max_loc
```

Out[14]:
```
passenger_count    132
dtype: int64
```

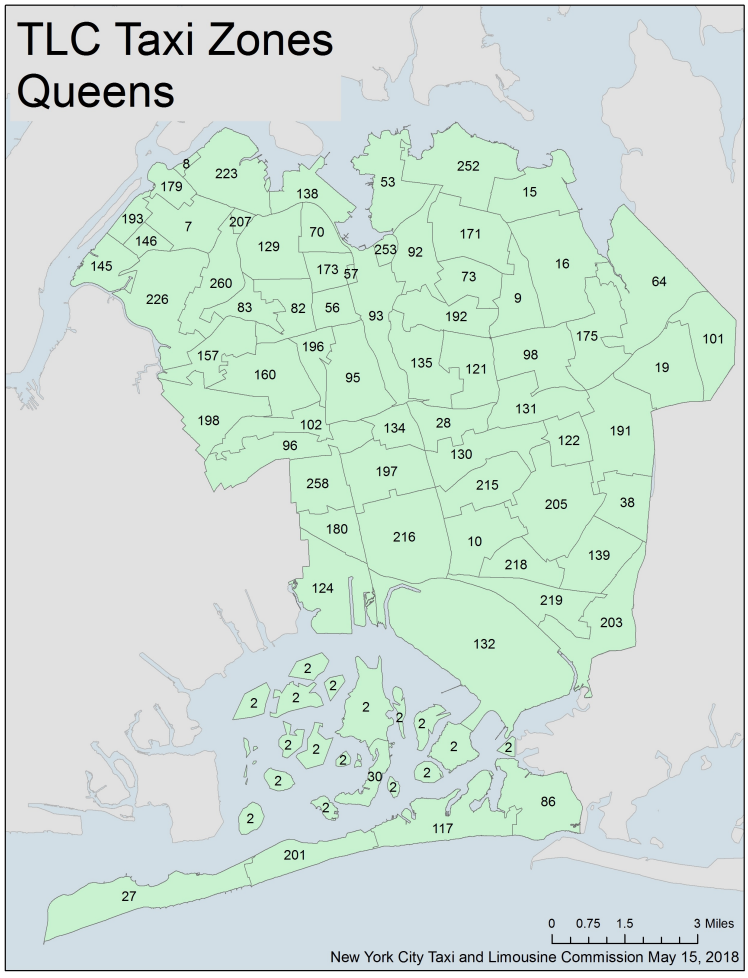
In [15]:
```
trips_by_loc.loc[max_loc]
```

Out[15]:

PULocationID	passenger_count
132	257867.0

¿Era esperable el resultado?

Acá pueden ver un mapa con el ID de los diferentes lugares:



4.5 Puntaje de premio

Con esto, ¿Cómo podríamos calcular el mejor horario y lugar durante la semana laboral en el cual recoger pasajeros en NYC?

Los tres primeros estudiantes que manden a nuestro correo el Jupyter Notebook con el código implementado para contestar esta pregunta tendrán 0.1 adicional en la Tarea 01. Debe venir la salida indicando el horario correspondiente y el código de cómo lo lograron calcular.

Posible forma de abordar el problema

La siguiente es la respuesta entergada por Vicente Muñoz a esta pregunta, calculando por separado el mejor horario y el mejor lugar durante la semana:

In [16]:
```
# Respuesta de Vicente Muñoz

# Mejor horario por dia laboral

trips_and_hour = trips[["tpep_pickup_datetime", "passenger_count", "PULocationID"]].copy()

#En python segun la documentacion https://docs.python.org/3/library/datetime.html los se puede usar weekday de
#Asi que los dias laborales son [0, 4] para eso el weekday < 5
trips_and_hour = trips_and_hour[trips_and_hour['tpep_pickup_datetime'].dt.weekday < 5]

#el .hour nos devuelve la hora de las 0 hasta las 23
trips_and_hour['hora'] = trips_and_hour['tpep_pickup_datetime'].dt.hour

#agrupar los pasejeros por hora
trips_by_hora = trips_and_hour[["hora", "passenger_count"]].groupby("hora").sum()

#en la documentacion se puede ordener asi por lo que la hora con mas pasajero en dia labrola es a las 18h
trips_by_hora_sorted = trips_by_hora.sort_values(by='passenger_count', ascending=False)
trips_by_hora_sorted.head(5)
```

Out[16]:

hora	passenger_count
18	256649.0
17	236828.0
19	230034.0
21	217037.0
15	216656.0

In [17]:
```
# Respuesta de Vicente Muñoz

trips_daywork_loc = trips[["tpep_pickup_datetime", "passenger_count", "PULocationID"]].copy()

trips_daywork_loc = trips_daywork_loc[trips_daywork_loc['tpep_pickup_datetime'].dt.weekday < 5]

trips_daywork_loc = trips[["PULocationID", "passenger_count"]].groupby("PULocationID").sum()
trips_daywork_loc = trips_daywork_loc.sort_values(by='passenger_count', ascending=False)
trips_daywork_loc.head(5)

#Por lo que la Location 132 es la mejor para recoger pasajeros en dia laboral
```

Out[17]:

PULocationID	passenger_count
132	257867.0
237	224815.0
161	215751.0
236	196753.0
138	176528.0

Posible forma de abordar el problema

La siguiente es la respuesta entergada por Tomás Romero a esta pregunta, calculando en forma conjunta el horario y lugar. El horario sigue siendo el mismo, pero el lugar cambia al filtrar:

In [18]:
```
#Seleccionamos las columnas que nos interesan
trips_by_hour_and_loc = trips[["passenger_count", "tpep_pickup_datetime", "PULocationID"]]

#Seleccionamos los dias de la semana laboral
trips_by_hour_and_loc = trips_by_hour_and_loc[trips_by_hour_and_loc["tpep_pickup_datetime"].apply(lambda x: x.d
trips_by_hour_and_loc = trips_by_hour_and_loc[trips_by_hour_and_loc["tpep_pickup_datetime"].apply(lambda x: x.d

#Agrupamos por hora y contamos
trips_by_hour_and_loc["hour"] = trips_by_hour_and_loc["tpep_pickup_datetime"].apply(lambda x: x.hour)
trips_by_hour_and_loc = trips_by_hour_and_loc.groupby(["hour", "PULocationID"]).count()

trips_by_hour_and_loc = trips_by_hour_and_loc.drop(columns=["tpep_pickup_datetime"])

#Ordenamos de forma descendente por la cantidad de pasajeros
#De esta manera obtenemos los mejores horarios y lugares para recoger pasajeros en New York
trips_by_hour_and_loc.sort_values(by='passenger_count', ascending=False).head()
```

Out[18]:

hour	PULocationID	passenger_count
18	**161**	11513
19	**161**	10758
18	**237**	10737
14	**237**	10704
17	**161**	10692

5. Estudio de datos de enfermos COVID

El primer paso es importar los datos estructurados del CSV. Para ello, podemos usar una librería de NumPy que permite importar datos en formato CSV directamente.

5.1 `numpy`: np.loadtxt() y np.genfromtxt()

Numpy provee funciones para leer archivos de texto estructurado directamente como arreglos (`np.ndarray`).

En primer lugar la función `np.loadtxt()` , permite cargar archivos cuyo contenido es solamente numérico. Generalmente trabajaremos con datasets que tienen distintos tipos de datos en distintas columnas; por ejemplo, strings y floats. En este caso, es necesario utilizar la función `np.genfromtxt()` , que puede manejar este tipo de datos. Si usamos como argumento `dtype=None` , la función infiere el tipo de datos de cada columna en forma automática.

La documentación de ambas funciones se encuentra en:

- https://numpy.org/doc/stable/reference/generated/numpy.loadtxt.html
- https://numpy.org/doc/stable/reference/generated/numpy.genfromtxt.html

In [19]:
```
# nombre del archivo a leer
data_file='Covid-19.csv'
```

In [20]:
```
# cargar el archivo
data = np.loadtxt(data_file, delimiter=',',dtype='str')#skiprows=1

# Algunas formas de explorar los datos:
print(data[1])
#print(data.shape)
#print(data)
```

```
['Arica y Parinacota' '15' 'Arica' '15101' '247552.0' '6.0' '6.0' '12.0'
 '41.0' '63.0' '87.0' '115.0' '124.0' '134.0' '166.0' '224.0' '270.0'
 '297.0' '310.0' '328.0' '353.0' '371.0' '405.0' '477.0' '525.0' '596.0'
 '653.0' '806.0' '904.0' '1046.0' '1176.0' '1371.0' '1533.0' '1758.0'
 '1887.0' '2139.0' '2464.0' '2721.0' '3123.0' '3372.0' '3882.0' '4211.0'
 '4636.0' '4874.0' '5220.0' '5543.0' '5907.0' '6131.0' '6374.0' '6577.0'
 '6783.0' '6969.0' '7172.0' '7344.0' '7503.0' '7646.0' '7819.0' '7948.0'
 '8134.0' '8332.0' '8597.0' '8772.0' '8996.0' '9138.0' '9343.0' '9472.0'
 '9679.0' '9763.0' '9888.0' '9974.0' '10103.0' '10208.0' '10294.0'
 '10363.0' '10443.0' '10489.0' '10530.0' '10586.0' '10630.0' '10672.0'
 '10720.0' '10798.0' '10875.0' '10934.0' '11028.0' '11080.0' '11212.0'
 '11296.0' '11434.0' '11650.0' '11886.0' '12171.0' '12453.0' '12711.0'
 '13047.0' '13335.0' '13606.0' '13885.0' '14168.0' '14427.0' '14734.0'
 '15007.0' '15358.0' '15627.0' '15978.0' '16253.0' '16614.0' '16913.0'
 '17431.0' '17771.0' '18370.0' '18637.0' '19157.0' '19396.0' '19845.0'
 '20142.0' '20550.0' '20894.0' '21325.0' '21692.0' '22130.0' '22415.0'
 '22821.0' '23137.0' '23547.0' '23856.0' '24247.0' '24548.0' '24929.0'
 '25231.0' '25609.0' '25913.0' '26258.0' '26531.0' '26916.0' '27195.0'
 '27576.0' '27813.0' '28074.0' '28299.0' '28532.0' '28688.0' '28995.0'
 '29117.0' '29158.0' '29203.0' '29242.0' '29302.0' '29348.0' '29408.0'
 '29450.0' '29493.0' '29501.0' '29535.0' '29557.0' '29579.0' '29662.0'
 '29681.0' '29763.0' '29836.0' '29929.0' '30002.0' '30128.0' '30239.0'
 '30376.0' '30463.0' '30536.0' '30579.0' '30661.0' '30727.0' '30828.0'
 '30890.0' '30983.0' '31053.0' '31125.0' '31208.0' '31361.0' '31438.0'
 '31519.0' '31566.0' '31643.0' '31662.0' '31713.0' '31735.0' '31782.0'
 '31811.0' '31855.0' '31889.0' '31933.0' '31973.0' '32046.0' '32151.0'
 '32655.0' '33207.0' '34072.0' '35016.0' '37024.0' '39034.0' '43677.0'
 '45910.0' '48893.0' '50671.0' '52463.0' '53430.0' '54455.0' '55052.0'
 '55732.0' '56143.0' '56668.0' '56924.0' '57291.0' '57528.0' '57874.0'
 '58024.0' '58273.0' '58479.0' '58633.0' '58703.0' '58877.0' '58945.0'
 '59026.0' '59066.0' '59128.0' '59159.0' '59214.0' '59274.0' '59352.0'
 '59435.0' '59529.0' '59608.0' '59701.0' '59810.0' '59975.0' '60105.0'
 '60292.0' '60431.0' '60642.0' '60816.0' '61165.0' '61502.0' '61999.0'
 '62408.0' '63156.0' '63755.0' '64624.0' '65167.0' '65951.0' '66507.0'
 '67301.0' '67978.0' '69095.0' '69825.0' '70887.0' '71627.0' '72812.0'
 '73457.0' '74222.0' '74694.0' '75195.0' '75449.0' '75761.0' '75894.0'
 '76085.0' '76203.0' '76395.0' '76477.0' '76553.0' '76751.0' '76925.0'
 '77026.0' '77102.0' '77214.0' '77303.0' '77431.0' '77497.0' '77636.0'
 '77791.0' '78043.0' '78210.0' '78479.0' '78678.0' '78952.0' '79100.0'
 '79383.0' '79564.0' '79891.0' '80139.0' '80584.0' '80924.0' '81364.0'
 '81713.0' '82516.0' '82960.0' '83092.0' '33512.2']
```

In [21]:
```python
# Importar data como floats y saltar la primera fila: data_float
data = np.genfromtxt(data_file, delimiter=',', dtype=None, skip_header=1, encoding=None)

print(data[0])
print(data.shape)

#numpy se las puede arreglar con datos mezclados pero es mejor pandas
```

```
('Arica y Parinacota', 15, 'Arica', 15101, 247552., 6., 6., 12., 41., 63., 87., 115., 124., 134., 166., 224., 2
70., 297., 310., 328., 353., 371., 405., 477., 525., 596., 653., 806., 904., 1046., 1176., 1371., 1533., 1758.,
1887., 2139., 2464., 2721., 3123., 3372., 3882., 4211., 4636., 4874., 5220., 5543., 5907., 6131., 6374., 6577.,
6783., 6969., 7172., 7344., 7503., 7646., 7819., 7948., 8134., 8332., 8597., 8772., 8996., 9138., 9343., 9472.,
9679., 9763., 9888., 9974., 10103., 10208., 10294., 10363., 10443., 10489., 10530., 10586., 10630., 10672., 107
20., 10798., 10875., 10934., 11028., 11080., 11212., 11296., 11434., 11650., 11886., 12171., 12453., 12711., 13
047., 13335., 13606., 13885., 14168., 14427., 14734., 15007., 15358., 15627., 15978., 16253., 16614., 16913., 1
7431., 17771., 18370., 18637., 19157., 19396., 19845., 20142., 20550., 20894., 21325., 21692., 22130., 22415.,
22821., 23137., 23547., 23856., 24247., 24548., 24929., 25231., 25609., 25913., 26258., 26531., 26916., 27195.,
27576., 27813., 28074., 28299., 28532., 28688., 28995., 29117., 29158., 29203., 29242., 29302., 29348., 29408.,
29450., 29493., 29501., 29535., 29557., 29579., 29662., 29681., 29763., 29836., 29929., 30002., 30128., 30239.,
30376., 30463., 30536., 30579., 30661., 30727., 30828., 30890., 30983., 31053., 31125., 31208., 31361., 31438.,
31519., 31566., 31643., 31662., 31713., 31735., 31782., 31811., 31855., 31889., 31933., 31973., 32046., 32151.,
32655., 33207., 34072., 35016., 37024., 39034., 43677., 45910., 48893., 50671., 52463., 53430., 54455., 55052.,
55732., 56143., 56668., 56924., 57291., 57528., 57874., 58024., 58273., 58479., 58633., 58703., 58877., 58945.,
59026., 59066., 59128., 59159., 59214., 59274., 59352., 59435., 59529., 59608., 59701., 59810., 59975., 60105.,
60292., 60431., 60642., 60816., 61165., 61502., 61999., 62408., 63156., 63755., 64624., 65167., 65951., 66507.,
67301., 67978., 69095., 69825., 70887., 71627., 72812., 73457., 74222., 74694., 75195., 75449., 75761., 75894.,
76085., 76203., 76395., 76477., 76553., 76751., 76925., 77026., 77102., 77214., 77303., 77431., 77497., 77636.,
77791., 78043., 78210., 78479., 78678., 78952., 79100., 79383., 79564., 79891., 80139., 80584., 80924., 81364.,
81713., 82516., 82960., 83092., 33512.2)
(362,)
```

En general, `numpy` hace un buen trabajo identificando los tipos en conjuntos de datos con tipos mezclados, pero la librería natural para trabajar con datos estructurados es `pandas`.

5.2 `pandas`: read_csv para pasar a DataFrame

La función `pd.read_csv()` permite leer un archivo de texto en formato CSV (comma separated value) y generar un DataFrame. El delimitador por defecto es la coma (,), pero también pueden leerse datasets con otros tipos de separación, especificando el parámetro `delimiter`.

https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.read_csv.html

In [22]:
```python
# leer los datos
```

```
data = pd.read_csv(data_file, delimiter=',')
data.head()
```

Out[22]:

	Region	Codigo region	Comuna	Codigo comuna	Poblacion	30-03-2020	01-04-2020	03-04-2020	06-04-2020	08-04-2020	...	09-12-2022	12-12-2022	16-12-2022	19-12-2022	23-12-2022	26-20
0	Arica y Parinacota	15	Arica	15101.0	247552.0	6.0	6.0	12.0	41.0	63.0	...	79891.0	80139.0	80584.0	80924.0	81364.0	8171
1	Arica y Parinacota	15	Camarones	15102.0	1233.0	0.0	0.0	0.0	0.0	0.0	...	205.0	207.0	207.0	208.0	212.0	21
2	Arica y Parinacota	15	General Lagos	15202.0	810.0	0.0	0.0	0.0	0.0	0.0	...	116.0	116.0	116.0	116.0	117.0	11
3	Arica y Parinacota	15	Putre	15201.0	2515.0	0.0	0.0	0.0	0.0	0.0	...	550.0	550.0	553.0	558.0	561.0	56
4	Arica y Parinacota	15	Desconocido Arica y Parinacota	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	228.0	228.0	228.0	228.0	228.0	22

5 rows × 295 columns

In [23]:
```
data.columns
```

Out[23]:
```
Index(['Region', 'Codigo region', 'Comuna', 'Codigo comuna', 'Poblacion',
       '30-03-2020', '01-04-2020', '03-04-2020', '06-04-2020', '08-04-2020',
       ...
       '09-12-2022', '12-12-2022', '16-12-2022', '19-12-2022', '23-12-2022',
       '26-12-2022', '30-12-2022', '05-01-2023', '09-01-2023', 'Tasa'],
      dtype='object', length=295)
```

In [24]:
```
type(data['Region'])
```

Out[24]:
```
pandas.core.series.Series
```

In [25]:
```
data.tail(5)
```

Out[25]:

	Region	Codigo region	Comuna	Codigo comuna	Poblacion	30-03-2020	01-04-2020	03-04-2020	06-04-2020	08-04-2020	...	09-12-2022	12-12-2022	16-12-2022	19-12-2022	23-12-2022	26-12-2022	30-12-2022
357	Magallanes	12	Rio Verde	12103.0	211.0	0.0	0.0	0.0	0.0	0.0	...	47.0	49.0	50.0	50.0	50.0	50.0	50.0
358	Magallanes	12	San Gregorio	12104.0	681.0	0.0	0.0	0.0	0.0	0.0	...	145.0	145.0	145.0	145.0	145.0	145.0	146.0
359	Magallanes	12	Timaukel	12303.0	282.0	0.0	0.0	0.0	0.0	0.0	...	54.0	55.0	55.0	55.0	55.0	55.0	55.0
360	Magallanes	12	Torres del Paine	12402.0	1021.0	0.0	0.0	0.0	0.0	0.0	...	117.0	117.0	117.0	117.0	118.0	118.0	119.0
361	Magallanes	12	Desconocido Magallanes	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	58.0	58.0	58.0	58.0	58.0	58.0	58.0

5 rows × 295 columns

In [26]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 362 entries, 0 to 361
Columns: 295 entries, Region to Tasa
dtypes: float64(292), int64(1), object(2)
memory usage: 834.4+ KB
```

5.2 Índices y acceso a información de celdas

Al igual que en el caso de NYC, podemos acceder a datos específicos de la base en forma simple.

In [27]:
```
# resumen de datos por comuna para un día en particular
datos_resumen = data[['Comuna','26-12-2022']]
datos_resumen
```

Out[27]:

	Comuna	26-12-2022
0	Arica	81713.0
1	Camarones	213.0
2	General Lagos	118.0
3	Putre	568.0
4	Desconocido Arica y Parinacota	228.0
...	...	...
357	Rio Verde	50.0
358	San Gregorio	145.0
359	Timaukel	55.0
360	Torres del Paine	118.0
361	Desconocido Magallanes	58.0

362 rows × 2 columns

In [28]:
```
# Columna de un día en particular
data['26-12-2022']
```

Out[28]:
```
0      81713.0
1        213.0
2        118.0
3        568.0
4        228.0
         ...
357       50.0
358      145.0
359       55.0
360      118.0
361       58.0
Name: 26-12-2022, Length: 362, dtype: float64
```

In [29]:
```
# nombre de las comunas
data['Comuna']
```

Out[29]:
```
0                            Arica
1                         Camarones
2                      General Lagos
3                             Putre
4      Desconocido Arica y Parinacota
                    ...
357                       Rio Verde
358                    San Gregorio
359                        Timaukel
360                  Torres del Paine
361             Desconocido Magallanes
Name: Comuna, Length: 362, dtype: object
```

In [30]:
```
# filtrar los datos para una comuna en particular
data_macul = data[data['Comuna']=='Macul']
data_macul
```

Out[30]:

	Region	Codigo region	Comuna	Codigo comuna	Poblacion	30-03-2020	01-04-2020	03-04-2020	06-04-2020	08-04-2020	...	09-12-2022	12-12-2022	16-12-2022	19-12-2022	23-12-2022	26
112	Metropolitana	13	Macul	13118.0	134635.0	10.0	15.0	23.0	34.0	37.0	...	34953.0	34979.0	35021.0	35056.0	35121.0	351

1 rows × 295 columns

In [31]:
```
# identificar las fechas disponibles
columnas = data.columns[5:-1]
columnas
```

Out[31]:
```
Index(['30-03-2020', '01-04-2020', '03-04-2020', '06-04-2020', '08-04-2020',
       '10-04-2020', '13-04-2020', '15-04-2020', '17-04-2020', '20-04-2020',
       ...
       '05-12-2022', '09-12-2022', '12-12-2022', '16-12-2022', '19-12-2022',
       '23-12-2022', '26-12-2022', '30-12-2022', '05-01-2023', '09-01-2023'],
      dtype='object', length=289)
```

In [32]:
```
# sólo los datos de las columnas con fecha
datos_enfermos = data[columnas]
datos_enfermos
```

Out[32]:

	30-03-2020	01-04-2020	03-04-2020	06-04-2020	08-04-2020	10-04-2020	13-04-2020	15-04-2020	17-04-2020	20-04-2020	...	05-12-2022	09-12-2022	12-12-2022	16-12-2022	19-12-2022	23-12-2022	26-12-2022	30-1
0	6.0	6.0	12.0	41.0	63.0	87.0	115.0	124.0	134.0	166.0	...	79564.0	79891.0	80139.0	80584.0	80924.0	81364.0	81713.0	8251
1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	203.0	205.0	207.0	207.0	208.0	212.0	213.0	21
2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	116.0	116.0	116.0	116.0	116.0	117.0	118.0	11
3	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	546.0	550.0	550.0	553.0	558.0	561.0	568.0	58
4	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	228.0	228.0	228.0	228.0	228.0	228.0	228.0	22
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
357	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	45.0	47.0	49.0	50.0	50.0	50.0	50.0	5
358	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	145.0	145.0	145.0	145.0	145.0	145.0	145.0	14
359	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	54.0	54.0	55.0	55.0	55.0	55.0	55.0	5
360	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	117.0	117.0	117.0	117.0	117.0	118.0	118.0	11
361	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	58.0	58.0	58.0	58.0	58.0	58.0	58.0	5

362 rows × 289 columns

In [33]:
```
# información estadística por cada columna
data.describe()
```

Out[33]:

	Codigo region	Codigo comuna	Poblacion	30-03-2020	01-04-2020	03-04-2020	06-04-2020	08-04-2020	10-04-2020	13-04-2020	...	
count	362.000000	346.000000	346.000000	343.000000	346.000000	346.00000	346.000000	346.000000	346.000000	346.000000	...	3
mean	8.784530	9034.997110	56237.890173	5.647230	8.141618	10.15896	13.254335	15.300578	18.138728	21.621387	...	146
std	3.884479	3818.147431	88945.967089	19.022787	25.224560	31.03781	38.455380	42.803249	49.523508	56.388132	...	235
min	1.000000	1101.000000	137.000000	0.000000	0.000000	0.00000	0.000000	0.000000	0.000000	0.000000	...	
25%	6.000000	6109.250000	9649.250000	0.000000	0.000000	0.00000	0.000000	0.000000	0.000000	0.000000	...	24
50%	8.000000	8313.500000	19770.000000	0.000000	0.000000	0.00000	0.000000	0.000000	0.000000	3.000000	...	52
75%	13.000000	13102.750000	55441.250000	0.000000	5.000000	6.75000	9.000000	11.000000	12.000000	14.000000	...	142
max	16.000000	16305.000000	645909.000000	181.000000	225.000000	293.00000	365.000000	407.000000	443.000000	471.000000	...	162

8 rows × 293 columns

5.3 Operaciones con columnas

Cada columna de un DataFrame es una Serie, que corresponde a un arreglo 1-D con una etiqueta. Por lo tanto, en el caso de columnas con datos numéricos, podemos aplicar todas las operaciones matemáticas disponibles en `numpy`:

https://numpy.org/doc/stable/reference/routines.math.html

También podemos realizar operaciones entre 2 o más columnas, o entre columnas y escalares.

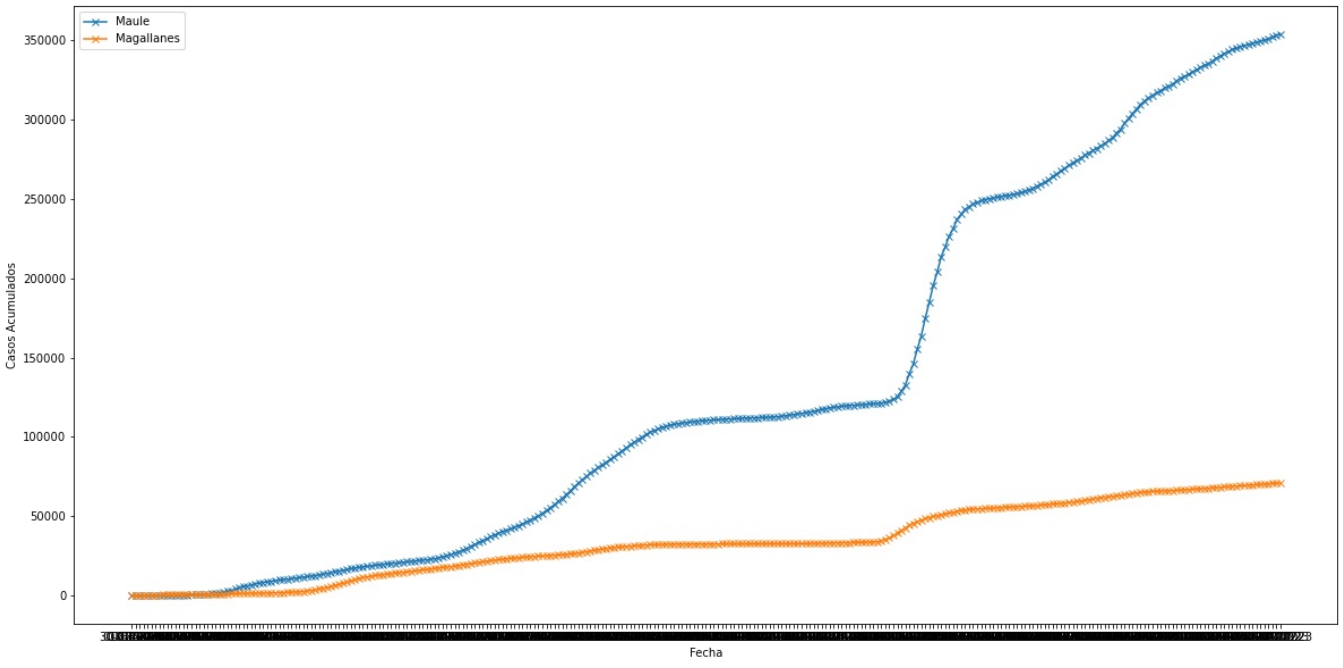
In [34]:
```
# calcular la cantidad de casos por region
enfermos_por_region = data.groupby("Region").sum(numeric_only=True)
# mostrar sólo los datos de enfermos
enfermos_por_region[columnas]
```

Out[34]:

Region	30-03-2020	01-04-2020	03-04-2020	06-04-2020	08-04-2020	10-04-2020	13-04-2020	15-04-2020	17-04-2020	20-04-2020	...	05-12-2022	09-12-2022	12-12-2022	16-12-2022
Antofagasta	29.0	32.0	49.0	65.0	72.0	105.0	149.0	176.0	211.0	264.0	...	179382.0	179971.0	180337.0	181071.0
Araucanía	187.0	292.0	414.0	541.0	628.0	680.0	758.0	823.0	884.0	1021.0	...	294386.0	294979.0	295304.0	295916.0
Arica y Parinacota	6.0	6.0	12.0	41.0	63.0	87.0	115.0	124.0	134.0	166.0	...	80657.0	80990.0	81240.0	81688.0
Atacama	0.0	0.0	4.0	4.0	5.0	12.0	13.0	13.0	13.0	14.0	...	109835.0	110435.0	110690.0	111168.0
Aysén	0.0	0.0	4.0	5.0	4.0	4.0	7.0	7.0	7.0	5.0	...	36155.0	36328.0	36388.0	36568.0
Biobío	72.0	224.0	263.0	360.0	409.0	439.0	507.0	542.0	573.0	627.0	...	526510.0	527807.0	528631.0	529911.0
Coquimbo	21.0	23.0	31.0	49.0	54.0	59.0	66.0	66.0	68.0	75.0	...	210698.0	211501.0	211943.0	212689.0
Los Lagos	104.0	169.0	198.0	250.0	283.0	304.0	380.0	390.0	412.0	431.0	...	249769.0	250446.0	250759.0	251349.0
Los Ríos	12.0	54.0	57.0	98.0	108.0	118.0	137.0	150.0	154.0	156.0	...	147753.0	148263.0	148542.0	149076.0
Magallanes	34.0	97.0	154.0	218.0	223.0	318.0	415.0	444.0	501.0	548.0	...	69337.0	69557.0	69679.0	69910.0
Maule	48.0	55.0	67.0	88.0	101.0	108.0	141.0	152.0	223.0	310.0	...	346491.0	347349.0	347851.0	348744.0
Metropolitana	1180.0	1483.0	1729.0	2193.0	2542.0	3180.0	3802.0	4228.0	4915.0	5597.0	...	1998612.0	2002505.0	2004733.0	2009193.0
O'Higgins	7.0	17.0	24.0	31.0	31.0	31.0	48.0	54.0	55.0	60.0	...	231639.0	232311.0	232697.0	233454.0
Tarapacá	5.0	9.0	10.0	18.0	22.0	33.0	52.0	62.0	73.0	93.0	...	115621.0	116085.0	116397.0	117038.0
Valparaíso	81.0	111.0	142.0	162.0	199.0	220.0	273.0	299.0	345.0	390.0	...	534951.0	536595.0	537554.0	539204.0
Ñuble	151.0	245.0	357.0	463.0	550.0	578.0	618.0	634.0	656.0	689.0	...	161336.0	161898.0	162235.0	162717.0

16 rows × 289 columns

In [35]:
```
fig = plt.figure(figsize=(20,10))
plt.plot(enfermos_por_region.loc['Maule'][columnas], 'x-', label='Maule')
plt.plot(enfermos_por_region.loc['Magallanes'][columnas], 'x-', label='Magallanes')
plt.legend()
plt.xlabel('Fecha')
plt.ylabel('Casos Acumulados')
plt.show()
```



5.4 Puntaje de premio

Con esto, ¿Cómo podríamos calcular en qué fecha fue el mayor aumento de casos en cada Región?

Los tres primeros estudiantes que manden a nuestro correo el Jupyter Notebook con el código implementado para contestar esta pregunta tendrán 0.1 adicional en la Tarea 01. Debe venir la salida indicando la fecha, en cada región, con el mayor aumento de casos y el código de cómo lo lograron calcular.

Posible forma de abordar el problema

La siguiente es la respuesta entergada por Nicolás Ortiz a esta pregunta:

In [36]:
```
# Respuesta de Nicolas Ortiz
```

```
enfermos_por_region.drop(columns=["Tasa"], inplace=True)

espacios_tabla = "|{:19}|{:11}|"
print(espacios_tabla.format("REGION", "FECHA"))
for index, row in enfermos_por_region.iterrows():
    index_max = 0
    dia_anterior = 0
    num_mayor = 0
    for i, dia_actual in enumerate(row[4:]):
        dif_dia = dia_actual  - dia_anterior
        dia_anterior = dia_actual
        if dif_dia > num_mayor:
            num_mayor = dif_dia
            index_max = i
    print(espacios_tabla.format(index, enfermos_por_region.columns[index_max + 4]))
```

```
|REGION             |FECHA      |
|Antofagasta        |28-01-2022 |
|Araucanía          |18-02-2022 |
|Arica y Parinacota |28-01-2022 |
|Atacama            |11-02-2022 |
|Aysén              |11-02-2022 |
|Biobío             |18-02-2022 |
|Coquimbo           |04-02-2022 |
|Los Lagos          |18-02-2022 |
|Los Ríos           |11-02-2022 |
|Magallanes         |04-02-2022 |
|Maule              |18-02-2022 |
|Metropolitana      |19-06-2020 |
|O’Higgins          |18-02-2022 |
|Tarapacá           |28-01-2022 |
|Valparaíso         |11-02-2022 |
|Ñuble              |25-02-2022 |
```

In [ ]: