

ACTIVIDAD 2

MANEJO DE CONJUNTOS DE DATOS

1 Definición del conjunto de datos (dataset)	3
1.1 Descripción información útil a extraer	3
1.2 Análisis del repositorio Open Data utilizado	3
1.3 Calidad de los Datos	4
1.4 información útil	4
2 Estrategia de programación y librerías usadas	5
2.1 Conexión de Robo3T a la base de datos.	5
2.2 Pruebas de conexión de Jupyter y Mongo DB Atlas	7
2.2 Creación de la los documentos de la colección covid19	8
2.3 Tratamiento de datos.	10
2.4 Librerías	12
3 Resultados de consultas y gráficas	13
4 Enlace a repositorio GitLab	23
5 Conclusiones	23

1 Definición del conjunto de datos (dataset)

Para esta actividad de manejo de datos hemos buscado en la web de la Generalitat Valenciana el apartado de Open Data, dirigiéndonos a la web: <https://portaldadesobertes.gva.es/es>. Una vez dentro de la web hemos ido a ver los conjuntos de datos y allí hemos seleccionado en el filtro todos los datos disponibles en formato CSV. Al ver la lista elegimos la lista de los casos de Covid-19 en los últimos 14 días.

1.1 Descripción información útil a extraer

Una vez descargado el archivo y abrirlo hemos podido ver que la información que contiene es la siguiente:

Id: Se trata de un tipo de dato **numérico**, que nos muestra el identificador de cada municipio en el propio Dataset

CodMunicipio: Se trata de un tipo de dato **numérico**, que nos muestra el código del municipio (según codificación del Instituto Nacional de Estadística, INE).

Municipi: Se trata de un dato de tipo **texto**, que nos muestra la denominación del municipio.

Casos PCR+: Se trata de un dato de tipo **numérico**, que nos muestra el número de casos de covid-19 diagnosticados por PCR acumulados desde el día 31 de enero de 2020 (casos PCR positivos) ubicados en el municipio (considerando municipio de riesgo).


Incidencia acumulada PCR+: Se trata de un dato de tipo **texto**, que nos muestra la incidencia acumulada de casos PCR positivos por 100.000 habitantes: número de casos PCR positivos del municipio (de riesgo) dividido por el número de habitantes del municipio y multiplicado por 100.000.

Casos PCR+ 14 días: Se trata de un dato de tipo **numérico**, que nos muestra el número de casos de covid-19 diagnosticados por PCR en los últimos 14 días (casos PCR positivos) según municipio (de riesgo).

Incidència acumulada PCR+ 14: Se trata de un dato de tipo **texto**, que nos muestra la incidencia acumulada de casos PCR positivos en los últimos 14 días por 100.000 habitantes: número de casos PCR positivos del municipio (de riesgo) de los últimos 14 días dividido por el número de habitantes del municipio y multiplicado por 100.000.

Defunciones: Se trata de un dato de tipo **numérico**, que nos muestra el número de defunciones de covid-19 según municipio (de riesgo) acumuladas desde el día 31 de enero de 2020.

Tasa de defunción: Se trata de un dato de tipo **texto**, que nos muestra la tasa de mortalidad por 100.000 habitantes: número de defunciones según municipio (de riesgo) dividido por el número de habitantes del municipio y multiplicado por 100.000.

 _id	CodMun...	Municipi	Casos PCR+	Incidència acumulada...	Casos PCR+ 14 dies	Incidènci...	Defunci...	Taxa de ...
1	46001	Ademuz	475	45805,21	18	1735,78	2	192,86
2	46002	Ador	773	44682,08	33	1907,51	5	289,02
3	46003	Albalat de l'Arxib	22	5501,00	1	100,00	1	100,00

1.2 Análisis del repositorio Open Data utilizado

Los datos que se nos proporciona desde la Generalitat se podrían catalogar como datos de 3 estrellas. Llegamos a esta conclusión hemos podido encontrarlos desde la web de datos abiertos de la Generalitat Valenciana, se tratan de datos estructurados, la Generalitat los facilita en formato de CSV, que es un formato no propietario y facilita desde la misma página acceso a otros datos, en este casos los datos de los 14 días de las semanas anteriores.

En la propia web nos indica el grado de apertura de los datos como 3 estrellas.

<https://dadesobertes.gva.es/es/dataset/covid-19-casos-confirmats-pcr-casos-pcr-en-els-ultims-14-dies-i-persones-mortes-per-municipi-2022>

1.3 Calidad de los Datos

A continuación vamos a analizar la calidad de los datos que nos ofrece la web de datos abiertos de la Generalitat.

En primer lugar analizaremos la disponibilidad del sitio. Al entrar podemos ver que posee un menú superior para elegir el tipo de datos que se quiera observar y distintas formas de catálogo, así como también un buscador. Al entrar a uno de los conjuntos de datos podemos ver al final de la página información adicional donde nos dan mas detalles sobre los datos.

Información adicional

CAMPO	VALOR
Origen de datos	Conselleria de Sanitat Universal i Salut Pública
Fuente	http://www.san.gva.es
Frecuencia de actualización	bisemanal
Geolocalización	No
Visitas en los últimos 14 días	540
Visitas totales	2061
Creado	7 de enero de 2022, 12:15 (UTC+01:00)
Última actualización	22 de febrero de 2022, 17:54 (UTC+01:00)
Fecha de inicio de datos	31 enero 2020
Fecha de fin de datos	21 febrero 2022

En esta tabla podemos ver que los datos se actualizan con frecuencia ya que la última actualización es del 22 de febrero de este año y que la actualización es bisemanal. También nos muestra el origen de los datos y la fuente, dándole esto credibilidad a los datos.

Cuando analizamos los datos en sí podemos ver que nos da exactamente los datos que necesitamos y además también las incidencias acumuladas para poder comparar todos los municipios por igual tanto en general como los últimos 14 días.

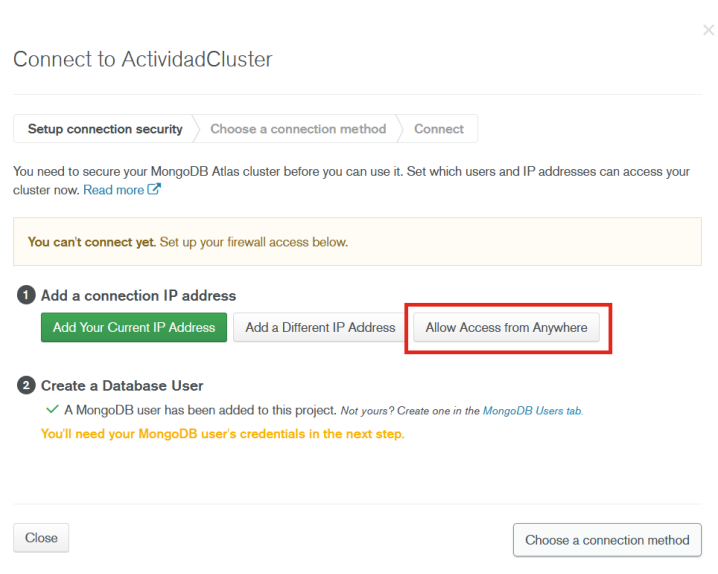
1.4 información útil

A partir de los datos citados anteriormente podemos sacar información útil como los positivos de covid por provincia, la media de infectados por cada 100.000 habitantes, cuáles han sido los pueblos con menor cantidad de fallecimientos o el número de habitantes de cada lugar.

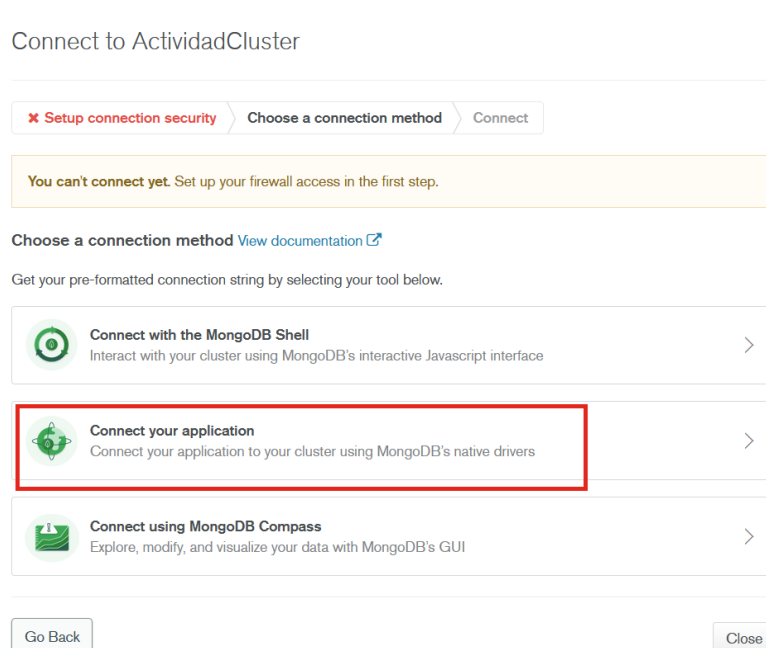
2 Estrategia de programación y librerías usadas

2.1 Conexión de Robo3T a la base de datos.

Para poder realizar la conexión desde Robo3T o desde Jupyter a una base de datos en MongoDB Atlas, se debe permitir que la dirección IP que iniciara la conexión, esté permitida por MongoDB Atlas; en la siguiente imagen se configura las IP permitidas, por ser caso de ejemplo se permite acceso desde cualquier IP.



Para obtener la cadena de conexión con el servidor se accede a la conectividad del cluster, y luego se procede como se indica en la siguiente figura.



Después, se procede a seleccionar el tipo de aplicación que se conectar con MongoDB Atlas, y se copia la cadena para conexión, esto se indica en la siguiente figura

Connect to ActividadCluster

✖ Setup connection security ✔ Choose a connection method Connect

You can't connect yet. Set up your firewall access in the first step.

1 Select your driver and version

DRIVER	VERSION
Python	3.12 or later

2 Add your connection string into your application code

☐ Include full driver code example

```
mongodb+srv://admin:<password>@actividadcluster.nzsm0.mongodb.net/myFirstDatabase?retryWrites=true&w=majority
```

Replace <password> with the password for the admin user. Replace myFirstDatabase with the name of the database that connections will use by default. Ensure any option params are URL encoded.

Having trouble connecting? [View our troubleshooting documentation](#)

Go Back Close

Para realizar la conectividad de MongoDB Atlas con Robo3T, se configura la conexión en la aplicación Robo3T como se indica en la siguiente figura.

Connection Settings

Connection Authentication SSH TLS Advanced

Type: Replica Set

Name: New Connection

Members: localhost:27017

Set Name:

From URI mongodb.net/myFirstDatabase?retryWrites=true&w=majority

Test Save Cancel

Connection Settings

Connection Authentication SSH TLS Advanced

Type: Replica Set

Name: ActividadMongo

Members: actividadcluster-shard-00-02.nzsm0.mongodb.ne...
actividadcluster-shard-00-00.nzsm0.mongodb.ne...
actividadcluster-shard-00-01.nzsm0.mongodb.ne...

Set Name: atlas-g4hu7b-shard-0

From URI Import connection details from MongoDB URI connection ...

Test Save Cancel

Connection Settings

Connection Authentication SSH TLS Advanced

☒ Perform authentication

Database: admin

The admin database is unique in MongoDB. Users with normal access to the admin database have read and write access to all databases.

User Name: admin

Password:

Auth Mechanism: SCRAM-SHA-1

☐ Manually specify visible databases

Test Save Cancel

Connection Settings

Connection Authentication SSH TLS Advanced

☒ Use TLS protocol

Authentication Method: Self-signed Certificate

In general, avoid using self-signed certificates unless the network is trusted. If self-signed certificate is used, the communications channel will be encrypted however there will be no validation of server identity.

☐ Use PEM Cert./Key: Enable this option to connect to a MongoDB that requires CA-signed client certificates/key file.

☐ Advanced Options

Test Save Cancel

2.2 Pruebas de conexión de Jupyter y Mongo DB Atlas

Al realizar las pruebas de comunicación de Jupyter Notebook, con la base de datos de MongoDB Atlas, se detectó el siguiente error.

```
import numpy as np
import pymongo
import matplotlib.pyplot as plt
import pandas as pd
import scipy as sp
import seaborn as sns
from pymongo import MongoClient
```

```
uri_mongo2 = "mongodb+srv://admin:admin@actividadcluster.nzsm0.mongodb.net"
conexion = pymongo.MongoClient(uri_mongo2,27017)
db = conexion["Actividad2"]
coll_covid19 = db.covid19
print(coll_covid19)
```

```
Collection(Database(MongoClient(host=['actividadcluster-shard-00-00.nzsm0.mongodb.net:27017', 'actividadcluster-shard-00-01.nzsm0.mongodb.net:27017', 'actividadcluster-shard-00-02.nzsm0.mongodb.net:27017'], document_class=dict, tz_aware=False, connect=True, authsource='admin', replicaset='atlas-g4hu7b-shard-0', tls=True), 'Actividad2'), 'covid19')
```

```
#Lista de las colecciones de la BBDD
db.list_collection_names()
```

```
-----
ServerSelectionTimeoutError: Traceback (most recent call last)
Input In [253], in <module>
      1 #Lista de las colecciones de la BBDD
----> 2 db.list_collection_names()
```

```
ServerSelectionTimeoutError: actividadcluster-shard-00-01.nzsm0.mongodb.net:27017: [SSL: CERTIFICATE_VERIFY_FAILED] certificate verify failed: certificate has expired (_ssl.c:997),actividadcluster-shard-00-02.nzsm0.mongodb.net:27017: [SSL: CERTIFICATE_VERIFY_FAILED] certificate verify failed: certificate has expired (_ssl.c:997),actividadcluster-shard-00-00.nzsm0.mongodb.net:27017: [SSL: CERTIFICATE_VERIFY_FAILED] certificate verify failed: certificate has expired (_ssl.c:997), Timeout: 30s, Topology Description: <TopologyDescription id: 6218f502f71974a357558ab4, topology_type: ReplicaSetNoPrimary, servers: [<ServerDescription ('actividadcluster-shard-00-00.nzsm0.mongodb.net', 27017) server_type: Unknown, rtt: None, error=AutoReconnect('actividadcluster-shard-00-00.nzsm0.mongodb.net:27017: [SSL: CERTIFICATE_VERIFY_FAILED] certificate verify failed: certificate has expired (_ssl.c:997)')>, <ServerDescription ('actividadcluster-shard-00-01.nzsm0.mongodb.net', 27017) server_type: Unknown, rtt: None, error=AutoReconnect('actividadcluster-shard-00-01.nzsm0.mongodb.net:27017: [SSL: CERTIFICATE_VERIFY_FAILED] certificate verify failed: certificate has expired (_ssl.c:997)')>, <ServerDescription ('actividadcluster-shard-00-02.nzsm0.mongodb.net', 27017) server_type: Unknown, rtt: None, error=AutoReconnect('actividadcluster-shard-00-02.nzsm0.mongodb.net:27017: [SSL: CERTIFICATE_VERIFY_FAILED] certificate verify failed: certificate has expired (_ssl.c:997)')>]>
```

En las diferentes pruebas realizadas se detecta que uno de los integrantes del grupo si puede conectarse a la base de datos, de mongoDB Atlas. El error es notificado por correo electrónico y se tiene la autorización para realizar las pruebas de la base de datos en modo Local. En la siguiente figura se indican las pruebas de conexión en modo local. La actividad se realiza en modo local y en la nube.

```
import numpy as np
import pymongo
import matplotlib.pyplot as plt
import pandas as pd
import scipy as sp
import seaborn as sns
from pymongo import MongoClient
```

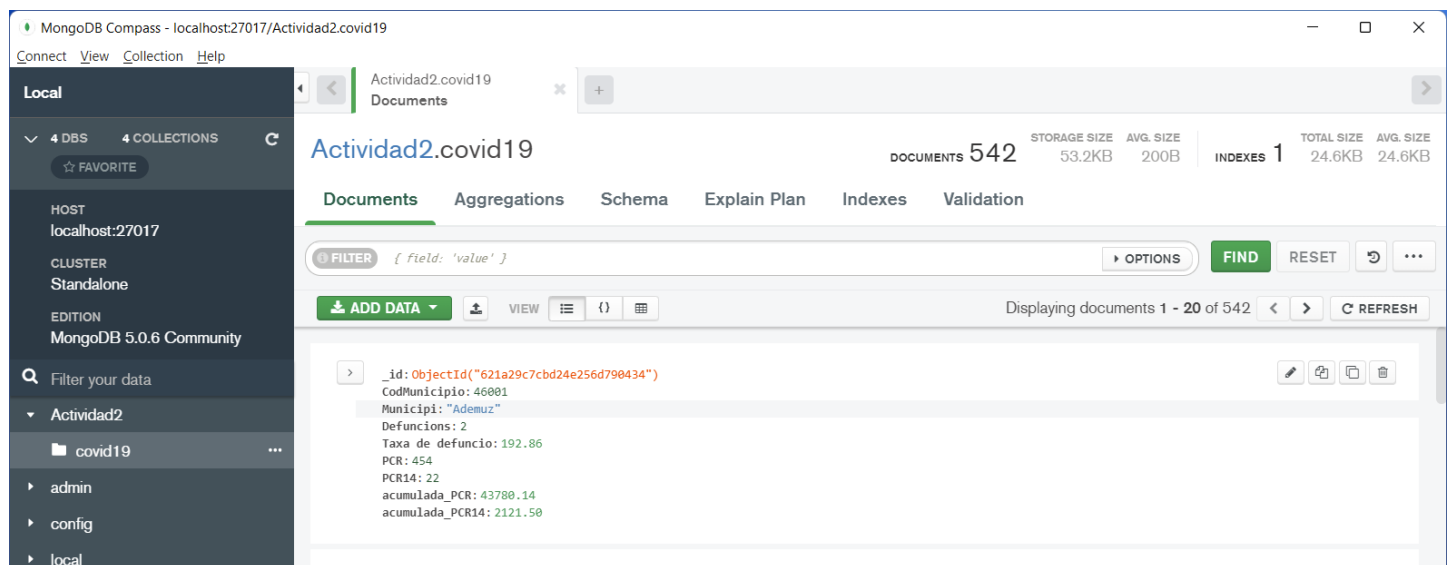
```
uri_mongo1 = "localhost"
conexion = pymongo.MongoClient(uri_mongo1,27017)
db = conexion["Actividad2"]
coll_covid19 = db.covid19
print(coll_covid19)
```

```
Collection(Database(MongoClient(host=['localhost:27017'], document_class=dict, tz_aware=False, connect=True), 'Actividad2'), 'covid19')
```

```
#Lista de las colecciones de la BBDD
db.list_collection_names()
```

```
['covid19']
```

Para el funcionamiento en modo local, se instala la herramienta MongoDB Compass, la cual permite visualizar la base de datos, en la siguiente figura se puede observar el despliegue de MongoDb Compass.



2.2 Creación de la los documentos de la colección covid19

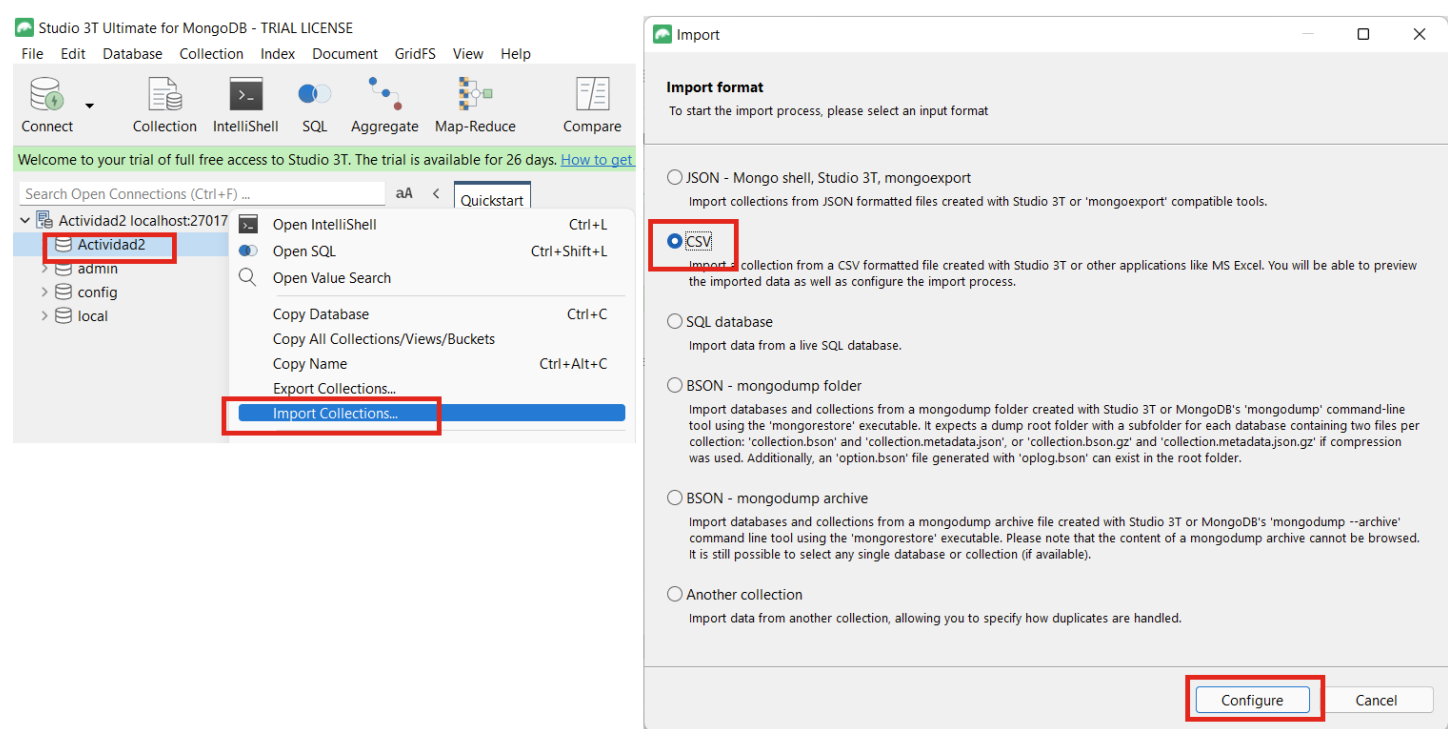
La creación de la colección covit19, se implementa tanto en la nube como en local, y el procedimiento de creación varía en la cadena de conexión.

Primero se realiza la descarga de la base de datos tipo CSV, de la dirección: <https://dadesobertes.gva.es/es/dataset/covid-19-casos-confirms-pcr-casos-pcr-en-els-ultims-14-dies-i-persones-mortes-per-municipi-2022>

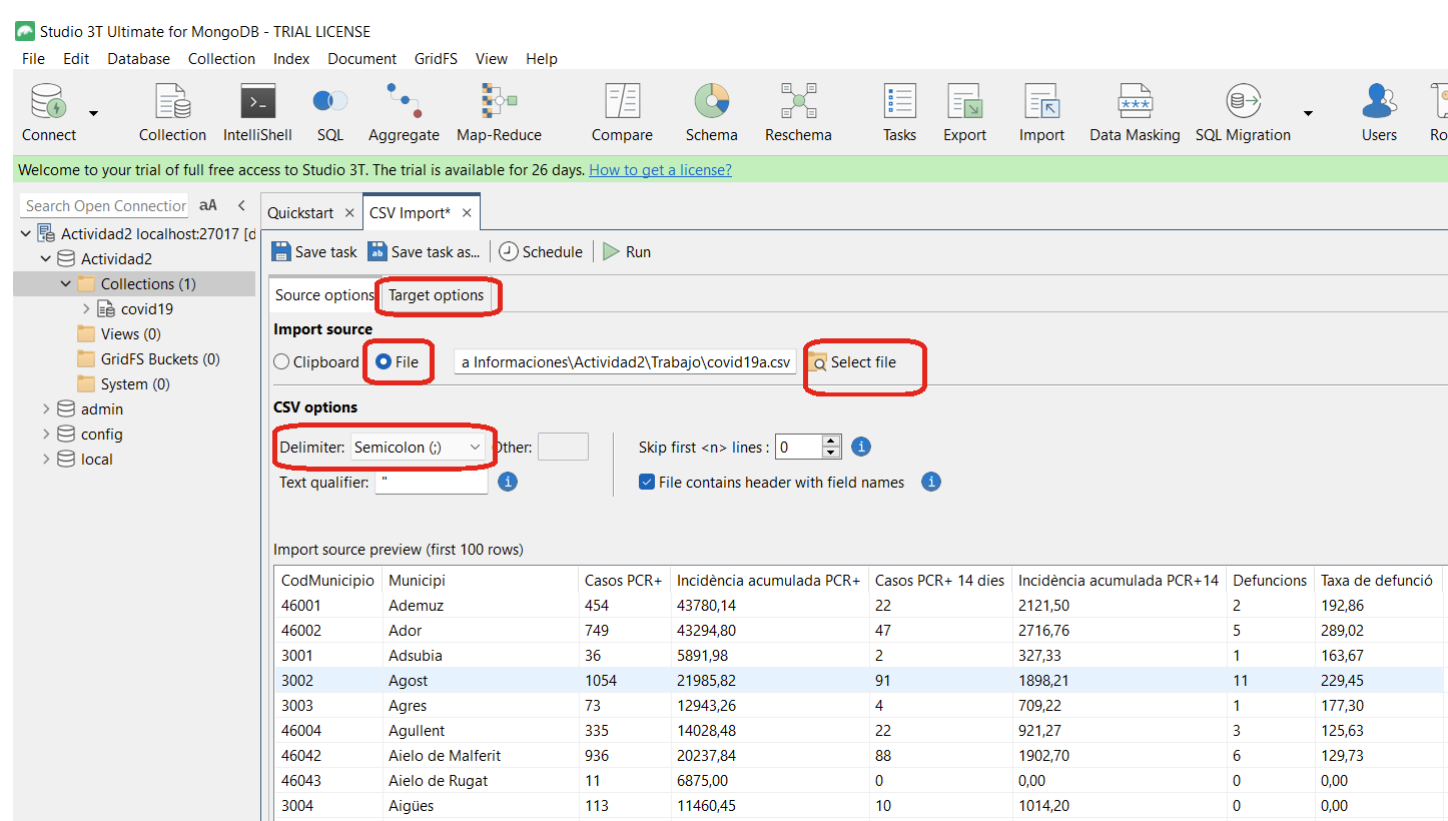
Con la herramienta RapidMiner, se realiza una evaluación de la base de datos del archivo CSV, donde se puede observar que no existen datos faltantes en los campos, también se realiza una estadística rápida de los datos y del tipo de datos utilizados en el documento CSV. los detalles observados en la herramienta RapidMiner se indica en la siguiente imagen.

Name	Type	Missing	Statistics		Filter (8 / 8 attributes):
CodMunicipio	Integer	0	Min 3001	Max 46904	Average 26570.126
Municipi	Nominal	0	Least Zucaina (1)	Most Ademuz (1)	Values Ademuz (1), Ador (1), ...[515 more]
Casos PCR+	Integer	0	Min 0	Max 208280	Average 2292.385
Incidència acumulada PCR+	Real	0	Min 0	Max 51535.090	Average 18182.917
Casos PCR+ 14 dies	Integer	0	Min 0	Max 19415	Average 208.762
Incidència acumulada PCR+14	Real	0	Min 0	Max 8884.300	Average 1721.112
Defuncions	Integer	0	Min 0	Max 1327	Average 16.478
Taxa de defunció*	Real	0	Min 0	Max 3099.170	Average 165.179

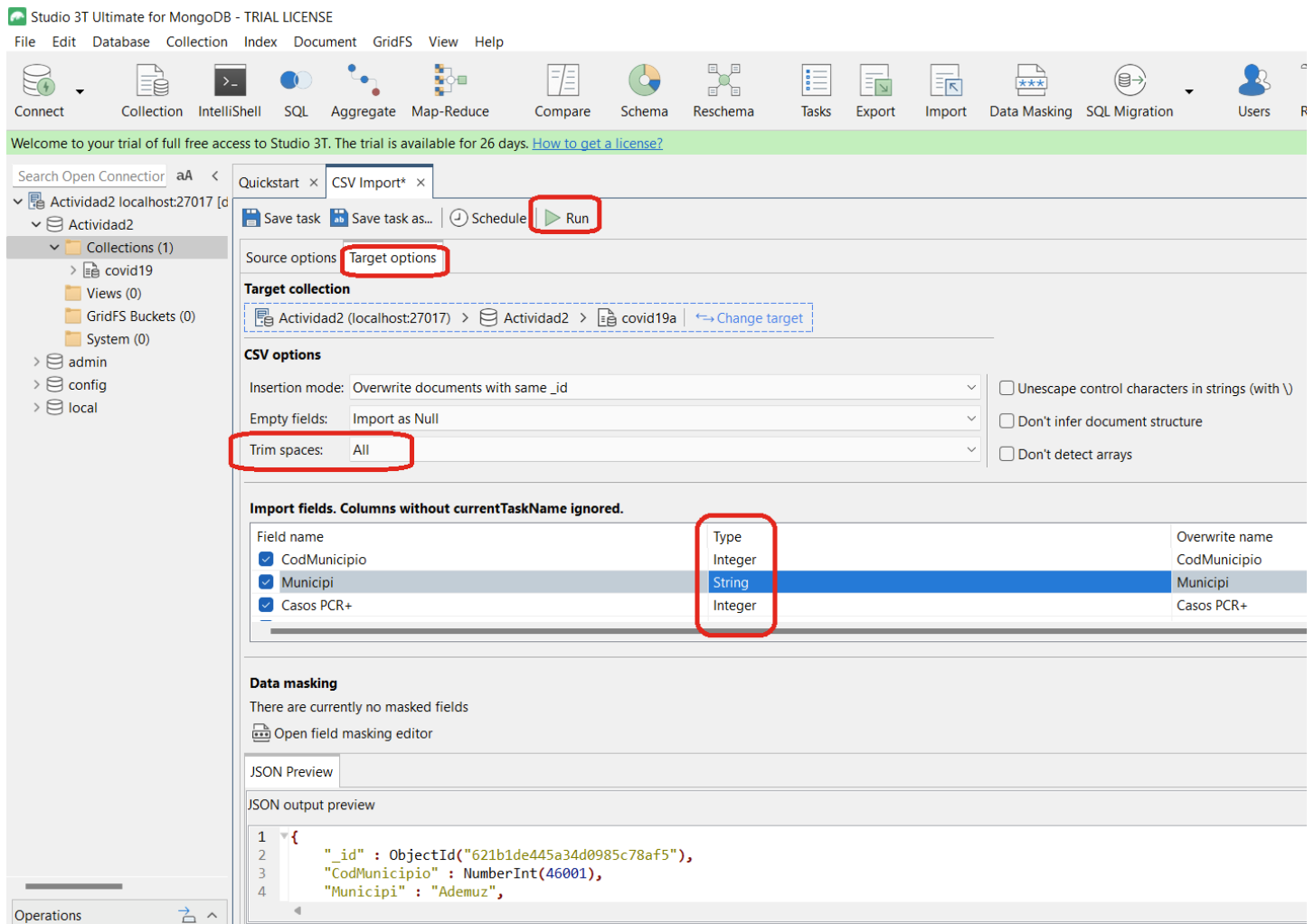
Concluida el análisis de los datos, se procede a importar la data tipo CSV, de manera directa con la herramienta sugerida por Robo 3T, qué es: Studio 3T. Se procede a realizar la conexión de la base de datos según la cadena de conexión para MongoDB Atlas y se procede a importar el archivos de datos, como se indica en la siguiente figura.



En la siguiente gráfica se indica el proceso de selección del archivo CSV y la configuración del tipo de limitador “;”



Configuración de target options, se configura el tipo de datos, la eliminación de espacios de los datos y el nombre las columnas, como observación se aclara que no se recomienda configurar las señales de tipo flotante o decimales ya que en jupyter no se puede cambiar el tipo de las señales. Al dar en play, automáticamente crea la colección y se completa el proceso de carga de documentos en la base de datos.



2.3 Tratamiento de datos.

En la base de datos creada, el tratamiento de datos que se puede realizar es la transformación del tipo de datos. En la siguiente figura se observa los tipos de datos existentes en la base de datos.

```
In [257]: #Consulta para ver el contenido e la colección covi19
dc_covid = coll_covid19.find()
# Definimos un dataframe
df_covid = pd.DataFrame(dc_covid)
df_covid.dtypes
```

```
Out[257]: _id                object
CodMunicipio             int64
Municipi                 object
Defuncions               int64
Taxa de defuncio         object
PCR                      int64
PCR14                   int64
acumulada_PCR            object
acumulada_PCR14          object
dtype: object
```

De los tipos de datos obtenidos, se puede observar que los datos de tipo flotante se encuentran definidos como tipo objeto, en este caso es un string, por tal motivo los datos no están en la capacidad de ser utilizados para cálculos matemáticos. Por lo que en la siguiente figura se realiza una transformación de tipos de datos.

```
In [132]: df_covid["CodMunicipio"] = df_covid["CodMunicipio"].astype(int)
df_covid["Defuncions"] = df_covid["Defuncions"].astype(int)
df_covid["Taxa de defuncio"] = df_covid["Taxa de defuncio"].astype(str).astype(float)
df_covid["PCR"] = df_covid["PCR"].astype(int)
df_covid["PCR14"] = df_covid["PCR14"].astype(int)
df_covid["acumulada_PCR"] = df_covid["acumulada_PCR"].astype(str).astype(float)
df_covid["acumulada_PCR14"] = df_covid["acumulada_PCR14"].astype(str).astype(float)
df_covid.dtypes
```

```
Out[132]: _id                object
CodMunicipio             int32
Municipi                 object
Defuncions               int32
Taxa de defuncio         float64
PCR                      int32
PCR14                   int32
acumulada_PCR            float64
acumulada_PCR14          float64
dtype: object
```

Adicionalmente se requiere la creación de datos tipos Nominales, o datos que son etiquetados y que corresponden a diferentes rangos de una variable, en la siguiente figura se indica las transformaciones requeridas.

```
In [258]: dc_covid = coll_covid19.find({},{"Provincia":1,"CodMunicipio":1,"Taxa de defuncio":1,"Defuncions":1,"Municipi":1})
df_covid = pd.DataFrame(dc_covid)
df_covid["Taxa de defuncio"] = df_covid["Taxa de defuncio"].astype(str).astype(float)
df_covid["Provincia"] = pd.cut(df_covid["CodMunicipio"],[3000,12000,46000,47000],labels=["Alicante", "Castellon", "Valencia"])
df_covid["Severidad"] = pd.cut(df_covid["Taxa de defuncio"],4,labels=["Baja", "Media", "Alta", "Critica"])
df_covid
```

Out[258]:

	_id	CodMunicipio	Municipi	Defuncions	Taxa de defuncio	Provincia	Severidad
0	621a29c7cbd24e256d790434	46001	Ademuz	2	192.86	Valencia	Baja
1	621a29c7cbd24e256d790435	46002	Ador	5	289.02	Valencia	Baja
2	621a29c7cbd24e256d790436	3001	Adsubia	1	163.67	Alicante	Baja
3	621a29c7cbd24e256d790437	3002	Agost	11	229.45	Alicante	Baja
4	621a29c7cbd24e256d790438	3003	Agres	1	177.30	Alicante	Baja
...
537	621a29c7cbd24e256d79064d	46261	Yátova	0	0.00	Valencia	Baja
538	621a29c7cbd24e256d79064e	46263	Zarra	0	0.00	Valencia	Baja
539	621a29c7cbd24e256d79064f	12141	Zorita del Maestrazgo	0	0.00	Castellon	Baja
540	621a29c7cbd24e256d790650	12142	Zucaina	0	0.00	Castellon	Baja
541	621a29c7cbd24e256d790651	46119	Ènova (l')	6	665.19	Valencia	Baja

542 rows x 7 columns

2.4 Librerías

numpy

Numpy es una biblioteca que da soporte para crear vectores y matrices grandes multidimensionales, junto con una gran colección de funciones matemáticas para el lenguaje de programación Python. Numpy permite al usuario escribir programas rápidos siempre que la mayoría de las operaciones funcionen en vectores o matrices en lugar de escalares.

esta librería la instalamos ya que es necesaria para poder trabajar con las otras librerías de representación gráfica

pymongo

PyMongo es una distribución de Python que contiene herramientas para trabajar con MongoDB y es la forma recomendada de trabajar con MongoDB desde Python.

Utilizaremos pymongo para poder acceder a nuestra base de datos en Mongo Atlas utilizando MongoClient para indicarle la URI de nuestra base de datos.

matplotlib.pyplot

Matplotlib.pyplot es una colección de funciones que hacen que matplotlib funcione como MATLAB. Cada pyplotfunción realiza algún cambio en una figura: por ejemplo, crea una figura, crea un área de trazado en una figura, traza algunas líneas en un área de trazado, decora el trazado con etiquetas, etc.

Si matplotlib se limitara a trabajar con listas, sería bastante inútil para el procesamiento numérico. En general, utilizará matrices numpy . De hecho, todas las secuencias se convierten internamente en matrices numpy.

pandas

Pandas es una biblioteca de código abierto con licencia BSD que proporciona estructuras de datos y herramientas de análisis de datos fáciles de usar y de alto rendimiento para el lenguaje de programación Python .

Pandas nos permite leer y escribir fácilmente ficheros en formato CSV, Excel y bases de datos SQL; con ella podemos acceder a los datos mediante índices o nombres para filas y columnas; y también posee métodos para reordenar, dividir y combinar conjuntos de datos.

Pandas dispone de tres estructuras de datos diferentes:

- Series: Estructura de una dimensión.
- DataFrame: Estructura de dos dimensiones (tablas).
- Panel: Estructura de tres dimensiones (cubos).

scipy

SciPy es una biblioteca libre y de código abierto para Python. Se compone de herramientas y algoritmos matemáticos.

SciPy contiene módulos para optimización, álgebra lineal, integración, interpolación, funciones especiales, FFT, procesamiento de señales y de imagen, resolución de ODEs y otras tareas para la ciencia e ingeniería.

seaborn

Seaborn es una biblioteca de visualización de datos de Python basada en matplotlib . Proporciona una interfaz de alto nivel para dibujar gráficos estadísticos atractivos e informativos.

Bokeh

Bokeh es una biblioteca de Python para crear visualizaciones interactivas para navegadores web modernos. Ayuda a crear gráficos, que van desde gráficos simples hasta tableros complejos con conjuntos de datos de transmisión. Con Bokeh, puede crear visualizaciones basadas en JavaScript sin escribir JavaScript.

Bokeh trabaja a través de la implementación de objetos de la subclase bokeh.models o *modelos* para propósitos de este post. Estos *modelos* creados con Bokeh pueden ser *widgets*, *herramientas* o *gráficos* (por mencionar sólo algunos), Cada una de las partes de los objetos mencionados se construyen a partir de *modelos* más elementales.

3 Resultados de consultas y gráficas

Consulta que muestra los municipios

```
dc_covid = coll_covid19.find({}, {"Casos PCR+":1, "Municipi":1, "_id":0})
```

```
df_covid=pd.DataFrame(dc_covid)
```

```
df_covid
```

	Municipi
0	Ademuz
1	Ador
2	Adsubia
3	Agost
4	Agres
...	...
537	Yátova
538	Zarra
539	Zorita del Maestrazgo
540	Zucaína
541	Ènova (l')

Consulta que muestra los municipios de la provincia de Alicante, CodMunicipio 3XXX

En Jupyter

```
dc_covid = coll_covid19.find({"$and": [{"CodMunicipio":{"$gt":3000}],{"CodMunicipio":{"$lt" : 4000}}]}, {"CodMunicipio":1, "Municipi": 1, "_id": 0})
```

```
# Mirar los índices
```

```
df_covid=pd.DataFrame(dc_covid)
```

```
df_covid
```

	CodMunicipio	Municipi
0	3001	Adsubia
1	3002	Agost
2	3003	Agres
3	3004	Aigües
4	3005	Albatera
...
277	3137	Vall de Laguar (la)
278	3138	Verger (el)
279	3139	Villajoyosa/Vila Joiosa (la)
280	3140	Villena
281	3081	Xaló

En Robo 3T

`db.covid19.find({"$and": [{"CodMunicipio":{"$gt":3000}}, {"CodMunicipio":{"$lt" : 4000}}]}, {"CodMunicipio":1, "Municipi": 1, "_id": 0})`

db.covid19.find({"\$and": [{"CodMunicipio":{"\$gt":3000}}, {"CodMunicipio":{"\$lt" : 4000}}]}, {"CodMunicipio":1, "Municipi": 1, "_id": 0})

0.057 sec.

CodMunicipio	Municipi
1 (a) 3001	(a) Adsubia
2 (a) 3002	(a) Agost
3 (a) 3003	(a) Agres
4 (a) 3004	(a) Aigües
5 (a) 3005	(a) Albatera
6 (a) 3006	(a) Alcalá
7 (a) 3007	(a) Alcozer de ...
8 (a) 3008	(a) Alcoleja
9 (a) 3009	(a) Alcoy/Alcoi
10 (a) 3010	(a) Alfara
11 (a) 3011	(a) Alfás del Pi (l)
12 (a) 3012	(a) Algorfa
13 (a) 3013	(a) Algueta
14 (a) 3014	(a) Alicante/...
15 (a) 3015	(a) Almoradí
16 (a) 3016	(a) Almudaina
17 (a) 3017	(a) Alquería ...
18 (a) 3018	(a) Altea
19 (a) 3019	(a) Aspe
20 (a) 3020	(a) Balones
21 (a) 3021	(a) Banyeres de ...
22 (a) 3022	(a) Benasau
23 (a) 3023	(a) Benetusa
24 (a) 3024	(a) Benetusa
25 (a) 3025	(a) Benetusa
26 (a) 3026	(a) Benetusa
27 (a) 3027	(a) Benetusa
28 (a) 3028	(a) Benetusa
29 (a) 3030	(a) Benetusa
30 (a) 3031	(a) Benetusa

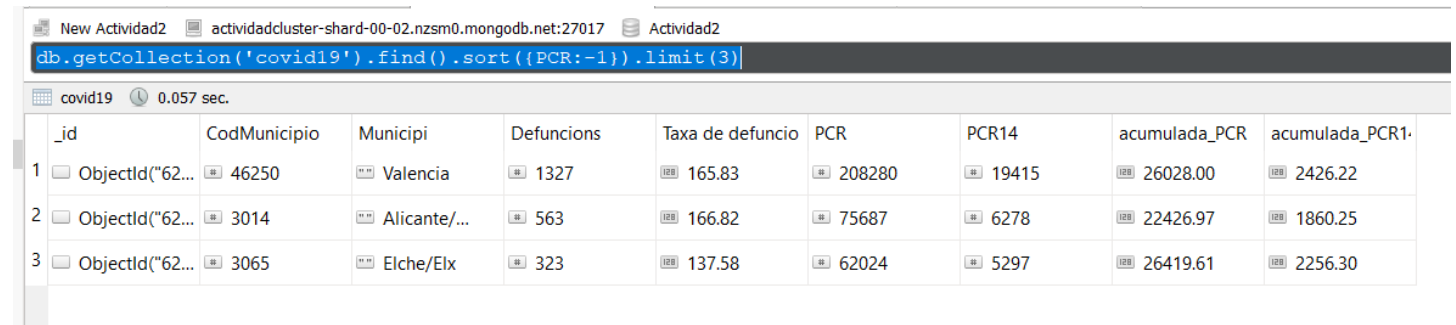
Consulta 2: Los 3 municipios con más casos positivos En Jupyter

`dc_covid=coll_covid19.find({}, {"_id":0, "Municipi":1, "CodMunicipio":1}).limit(3).sort("PCR", pymongo.DESCENDING)`
`df_covid=pd.DataFrame(dc_covid)`
`df_covid`

	CodMunicipio	Municipi
0	46250	Valencia
1	3014	Alicante/Alacant
2	3065	Elche/Elx

En Robo3T

```
db.getCollection('covid19').find().sort({PCR:-1}).limit(3)
```



The screenshot shows the Robo3T interface with a MongoDB query: `db.getCollection('covid19').find().sort({PCR:-1}).limit(3)`. The results are displayed in a table with 9 columns: `_id`, `CodMunicipio`, `Municipio`, `Defuncions`, `Taxa de defuncio`, `PCR`, `PCR14`, `acumulada_PCR`, and `acumulada_PCR14`. The first three rows are highlighted.

	_id	CodMunicipio	Municipio	Defuncions	Taxa de defuncio	PCR	PCR14	acumulada_PCR	acumulada_PCR14
1	ObjectId("62...)	46250	Valencia	1327	165.83	208280	19415	26028.00	2426.22
2	ObjectId("62...)	3014	Alicante/...	563	166.82	75687	6278	22426.97	1860.25
3	ObjectId("62...)	3065	Elche/Elx	323	137.58	62024	5297	26419.61	2256.30

➤ **Consulta 3:** en función de una operación estadística a un campo (ej.: nombre de las ciudades con densidad de población (hab/km2) mayor a la media de todas las ciudades).

Mostrar los habitantes de cada municipio

```
dc_covid = coll_covid19.find({}, {"Municipio":1, "PCR":1, "acumulada_PCR":1})
```

```
df_covid = pd.DataFrame(dc_covid)
```

```
df_covid["acumulada_PCR"] = df_covid["acumulada_PCR"].astype(str).astype(float)
```

```
df_covid["Habitantes"] = df_covid["PCR"] / (df_covid["acumulada_PCR"] / 100000)
```

```
df_covid
```

	_id	Municipio	PCR	acumulada_PCR	Habitantes
0	621a29c7cbd24e256d790434	Ademuz	454	43780.14	1036.999882
1	621a29c7cbd24e256d790435	Ador	749	43294.80	1729.999908
2	621a29c7cbd24e256d790436	Adsubia	36	5891.98	611.000037
3	621a29c7cbd24e256d790437	Agost	1054	21985.82	4793.999041
4	621a29c7cbd24e256d790438	Agres	73	12943.26	564.000105
...
537	621a29c7cbd24e256d79064d	Yátova	217	10437.71	2079.000087
538	621a29c7cbd24e256d79064e	Zarra	55	15895.95	346.000082
539	621a29c7cbd24e256d79064f	Zorita del Maestrazgo	11	9909.91	110.999999
540	621a29c7cbd24e256d790650	Zucaina	25	15432.10	161.999987
541	621a29c7cbd24e256d790651	Ènova (l')	175	19401.33	902.000018

Realizar al menos dos tipos de tratamiento y transformación sobre los datos (ej.: `pivot_table`, `groupby`, `aggregate`, etc.).

Cuenta cuántos municipios hay por provincia

```
dc_covid = coll_covid19.find({}, {"Provincia":1, "CodMunicipio":1, '_id': False})
```

```
df_covid = pd.DataFrame(dc_covid)
```

```
df_covid["Provincia"] = pd.cut(df_covid["CodMunicipio"], [3000, 12000, 46000, 47000], labels=["Alicante", "Castellon", "Valencia"])
```

```
df_covid.groupby('Provincia').count()
```

CodMunicipio	
Provincia	
Alicante	141
Castellon	135
Valencia	266

Consulta para contar en función de la severidad del índice de defunciones, organizado por provincia para ello se utiliza un pivot_table. El resultado obtenido es similar al obtenido en una tabla dinámica de Excel.

```
dc_covid = coll_covid19.find({},{'Provincia':1,'CodMunicipio':1, 'Taxa de defuncio':1, 'Defuncions':1, 'Municipi':1})
df_covid = pd.DataFrame(dc_covid)

df_covid['Taxa de defuncio'] = df_covid['Taxa de defuncio'].astype(str).astype(float)
df_covid['Provincia'] = pd.cut(df_covid['CodMunicipio'], [0,4000,13000,47000],labels=["Alicante", "Castellon", "Valencia"])
df_covid['Severidad'] = pd.cut(df_covid['Taxa de defuncio'],4,labels=["Baja", "Media", "Alta", "Critica"])

df_covid.to_csv('Severidad.csv')
pd.pivot_table(df_covid,
               values='Taxa de defuncio',
               index=['Severidad'],
               columns=['Provincia'],
               aggfunc='count')
```

Provincia	Alicante	Castellon	Valencia
Severidad			
Baja	141	131	259
Media	0	3	5
Alta	0	0	2
Critica	0	1	0

Consulta que agrupa los municipios por provincia y muestra la menor cantidad de casos, la media de los casos y la cantidad de casos más elevada

```
dc_covid = coll_covid19.find()
df_covid = pd.DataFrame(dc_covid)

df_covid['CodProvincia'] = pd.cut(df_covid['CodMunicipio'], [0,4000,13000,47000],labels=["Alicante", "Castellon", "Valencia"])
df_covid.groupby('CodProvincia').aggregate({'PCR': ['min', 'mean', 'max'],'Defuncions': ['min', 'mean', 'max']})
```


	PCR			Defuncions		
	min	mean	max	min	mean	max
CodProvincia						
Alicante	0	3066.851064	75687	0	23.936170	563
Castellon	0	1114.925926	43858	0	7.429630	286
Valencia	0	2418.090226	208280	0	16.593985	1327

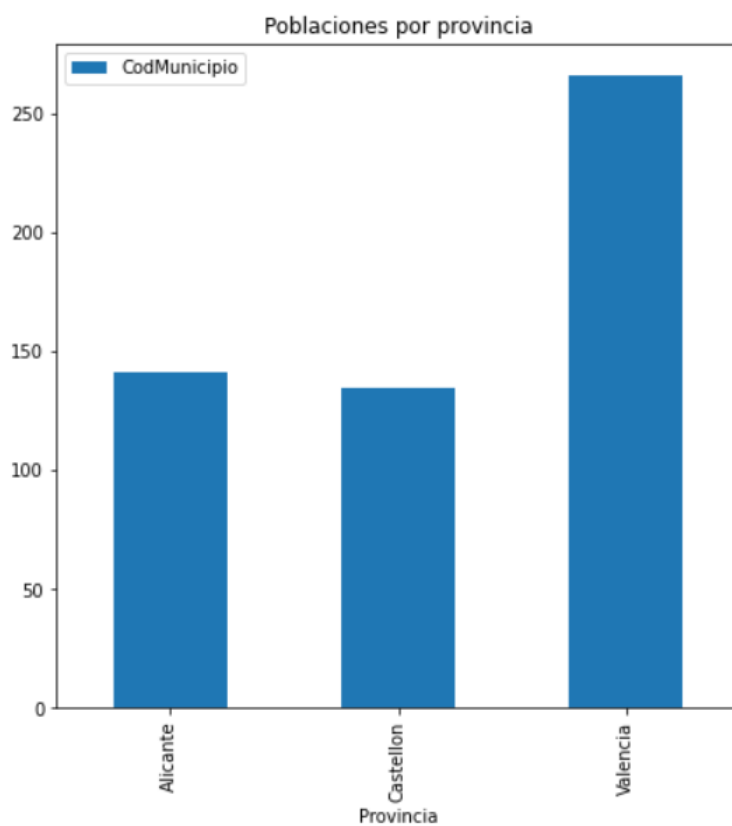
Obtener gráficas con diferentes librerías

Matplotlib

Gráfica que muestra la cantidad de poblaciones por provincia.

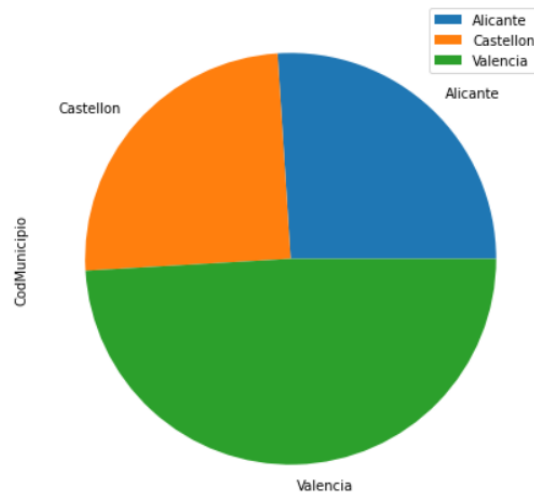
```
dc_covid = coll_covid19.find({},{"Provincia":1,"CodMunicipio":1})
df_covid = pd.DataFrame(dc_covid)
```

```
df_covid["Provincia"] = pd.cut(df_covid["CodMunicipio"],[3000,12000,46000,47000],labels=["Alicante",
"Castellon", "Valencia"])
df_poblaciones = df_covid.groupby('Provincia').count()
df_poblaciones.plot(y="CodMunicipio", kind="bar", figsize=(7,7))
plt.title("Poblaciones por provincia")
```



```
dc_covid = coll_covid19.find({},{"Provincia":1,"CodMunicipio":1,'_id': False})
df_covid = pd.DataFrame(dc_covid)
```

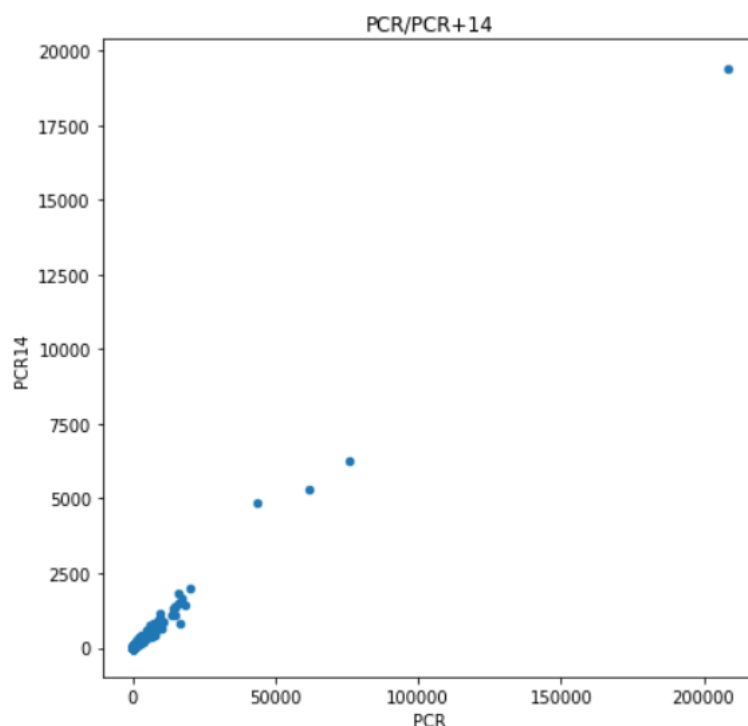
```
df_covid["Provincia"]= pd.cut(df_covid["CodMunicipio"],[3000,12000,46000,47000],labels=["Alicante",
"Castellon", "Valencia"])
df_poblaciones=df_covid.groupby('Provincia').count()
plot=df_poblaciones.plot.pie( subplots=True, figsize=(7,7))
```



Gráfica que muestra la relación entre los casos positivos de PCR y los que ha habido en los últimos 14 días.

```
dc_covid = coll_covid19.find()
```

```
# Definimos un dataframe
df_covid = pd.DataFrame(dc_covid)
# Mirar los índices
df_covid.plot( y= "PCR14", x="PCR", kind="scatter", figsize=(7,7))
plt.title("PCR/PCR+14")
```

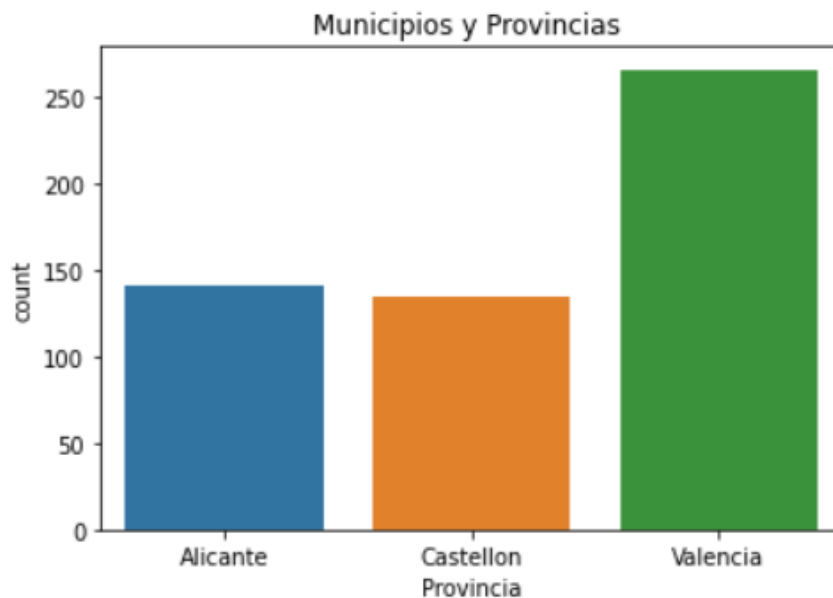


Seaborn

Gráfica que muestra la cantidad de poblaciones por provincia.

```
dc_covid = coll_covid19.find()
df_covid = pd.DataFrame(dc_covid)
df_covid["Taxa de defuncio"] = df_covid["Taxa de defuncio"].astype(str).astype(float)

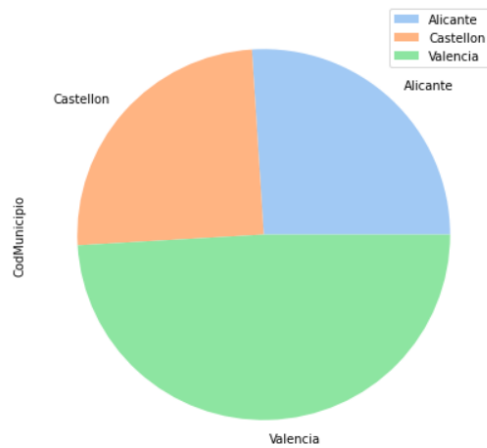
df_covid['Provincia'] = pd.cut(df_covid['CodMunicipio'], [0,4000,13000,47000],labels=["Alicante", "Castellon",
"Valencia"])
sns.countplot(x='Provincia', data=df_covid)
plt.title("Municipios y Provincias ")
```



```
dc_covid = coll_covid19.find({},{"Provincia":1,"CodMunicipio":1,'_id': False})
df_covid = pd.DataFrame(dc_covid)

#uso de la libreria seaborn
colors = sns.color_palette('pastel')[0:3]

df_covid["Provincia"] = pd.cut(df_covid["CodMunicipio"],[3000,12000,46000,47000],labels=["Alicante",
"Castellon", "Valencia"])
df_poblaciones=df_covid.groupby('Provincia').count()
plot=df_poblaciones.plot.pie( subplots=True, figsize=(7,7),colors=colors)
```



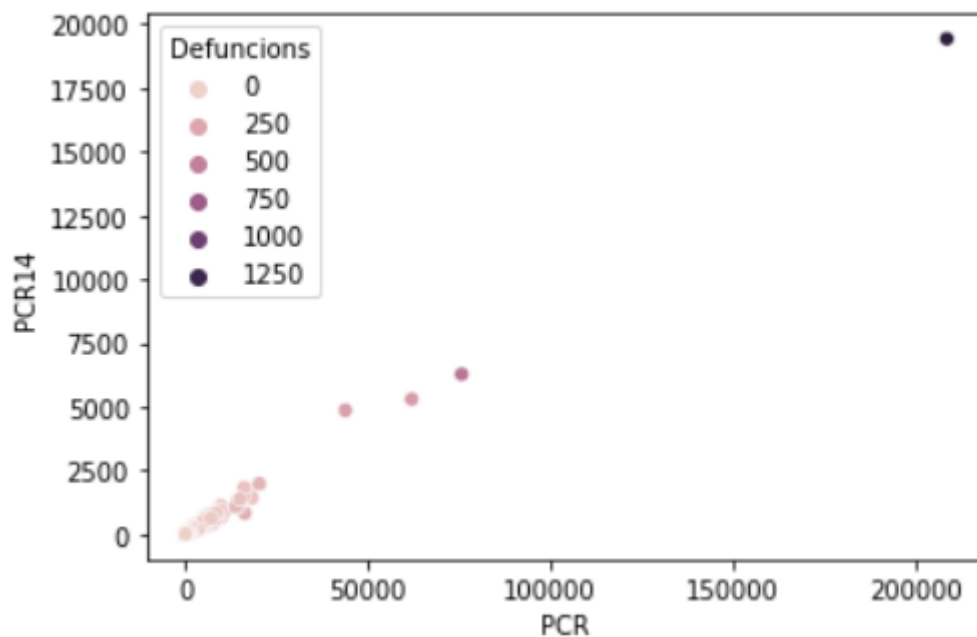
Gráfica que muestra la relación entre los casos positivos de PCR y los que ha habido en los últimos 14 días.

```
dc_covid = coll_covid19.find()
```

```
# Definimos un dataframe
```

```
df_covid = pd.DataFrame(dc_covid)
```

```
sns.scatterplot(x='PCR', y='PCR14', data=df_covid, hue='Defuncions')
```



Bokeh

Gráfica que muestra la cantidad de poblaciones por provincia.

```
dc_covid = coll_covid19.find({}, {"Provincia":1, "CodMunicipio":1, '_id': False})
```

```
df_covid = pd.DataFrame(dc_covid)
```

```
df_covid["Provincia"] = pd.cut(df_covid["CodMunicipio"], [3000, 12000, 46000, 47000], labels=["Alicante", "Castellon", "Valencia"])
```

```
df_poblaciones = df_covid.groupby('Provincia').count()
```

```
df_poblaciones
```

```
source = ColumnDataSource(df_poblaciones)
```

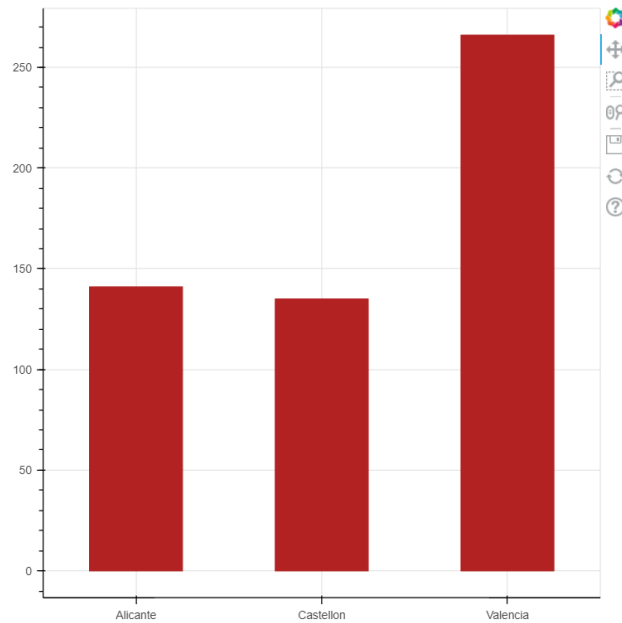
```
provincias = source.data['Provincia'].tolist()
```

```

p = figure(x_range=provincias)
p.vbar(x="Provincia", width=0.5, bottom=0,top='CodMunicipio', color="firebrick",source=source)

show(p)

```



```

dc_covid = coll_covid19.find({},{"Provincia":1,"CodMunicipio":1,'_id': False})
df_covid = pd.DataFrame(dc_covid)
df_covid["Provincia"] = pd.cut(df_covid["CodMunicipio"],[3000,12000,46000,47000],labels=["Alicante",
"Castellon", "Valencia"])
df_poblaciones=df_covid.groupby('Provincia').count()

```

```

df_poblaciones['angle'] = df_poblaciones['CodMunicipio']/df_poblaciones['CodMunicipio'].sum() * 2*pi
df_poblaciones['color'] = Category20c[len(df_poblaciones)]

```

```

p = figure(height=350, title="Casos por Provincia", toolbar_location=None,
tools="hover", tooltips="@Provincia: @CodMunicipio", x_range=(-0.5, 1.0))

```

```

p.wedge(x=0, y=1, radius=0.4,
start_angle=cumsum('angle', include_zero=True), end_angle=cumsum('angle'),
line_color="white", fill_color='color', legend_field='Provincia', source=df_poblaciones)

```

```

p.axis.axis_label = None
p.axis.visible = False
p.grid.grid_line_color = None

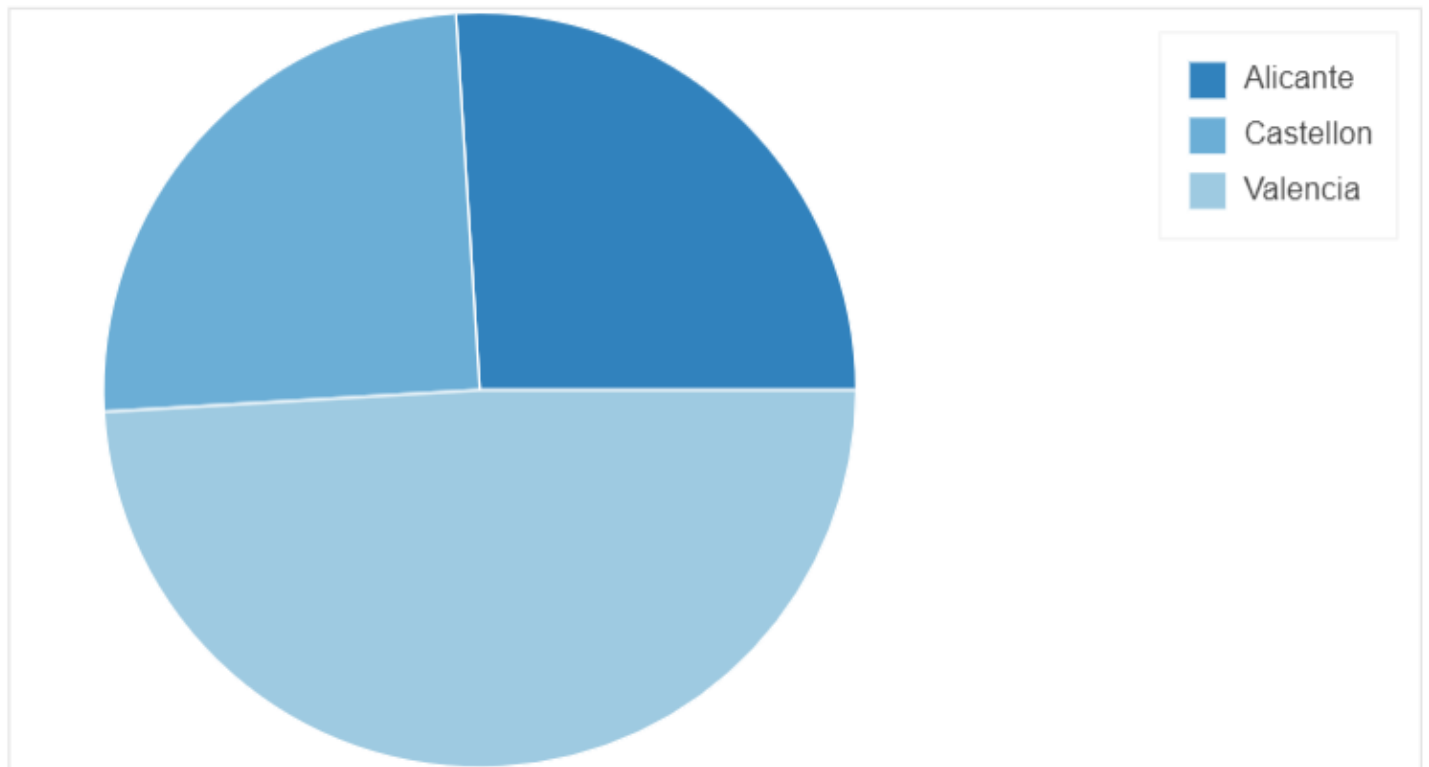
```

```

show(p)

```

Casos por Provincia



Gráfica que muestra la relación entre los casos positivos de PCR y los que ha habido en los últimos 14 días.

```
dc_covid = coll_covid19.find()
```

```
df_covid = pd.DataFrame(dc_covid)
```

```
p = figure(width=400, height=400)
```

```
source=ColumnDataSource(df_covid)
```

```
pcr=source.data['PCR'].tolist()
```

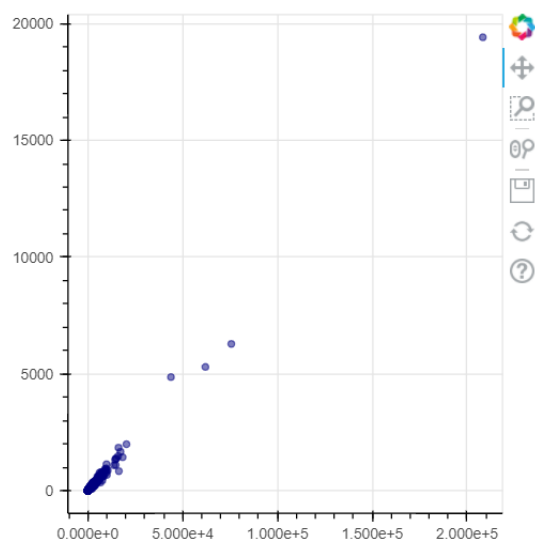
```
pcr14=source.data['PCR14'].tolist()
```

```
# add a circle renderer with a size, color, and alpha
```

```
p.circle(pcr, pcr14, size=5, color="navy", alpha=0.5)
```

```
# show the results
```

```
show(p)
```



4 Enlace a repositorio GitLab

<https://github.com/MarcosReolid/06MloTA1>

5 Conclusiones

Las bases de datos han permitido que el manejo de los datos y su explotación se conviertan en un área sumamente especializada de la computación, estableciendo nuevos retos en su funcionamiento, cumpliendo necesidades específicas en diversas áreas de negocios y generando nuevas de mayor complejidad y que requieren mayor dedicación de recursos y personal.

Las bases de datos relacionales no podían permanecer alejadas a esta tendencia de mejora continua. Para poder satisfacer necesidades específicas en las cuales las bases de datos relacionales resultan poco útiles, es que surgen las bases de datos NoSQL.

A lo largo de este trabajo se observó qué es, cómo operan y tipos de bases de datos NoSQL, en nuestro caso MongoDB.

Mientras que en el modelo relacional se tiene un conjunto de entidades con una estructura estrictamente idéntica, las bases de datos NoSQL orientadas a documentos limita esta restricción al proveer colecciones dentro de las cuales se almacenan documentos estructuralmente similares, con la facilidad de que son modificables sin afectar la información existente.

Esta es una gran ventaja, ya que de requerir un cambio en la estructura dentro de un modelo relacional, la demás información se ve afectada, sin mencionar la problemática que genera el modificar una nueva entidad, mientras que en un documento, basta con agregar los elementos requeridos sin mayor complicación. Debido a que la información se encuentra en un solo documento y no a través de múltiples tablas, la velocidad de respuesta es superior cuando la cantidad de documentos aumenta y la estructura de los mismos se vuelve más compleja.

Cuando elegir una Base de datos NoSql:

- Si se requiere un sistema altamente disponible, distribuido y a un bajo costo, NoSQL resuelve fácilmente este tema.
- Si se requiere un sistema que pueda manejar muy grandes volúmenes de datos y estar inherentemente preparados para tecnologías como Big Data, NoSQL es una opción altamente viable.
- Si se requiere mejor utilización de recursos físicos locales y en un sistema distribuido, además de una administración relativamente sencilla, NoSql es una solución ideal.
- Además la escalabilidad de un sistema NoSql es más sencillo y a un costo relativamente bajo.