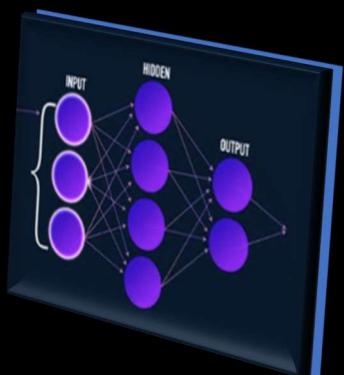


Explorando la relación entre la red de sonómetros de Bilbao y el nivel de ruido en la ciudad

Reto Final- IABD



Iker Lobo Pérez

Índice

1.	Introducción.....	3
2.	Sistemas de Aprendizaje Automático y Big Data Aplicado	3
a.	Seleccionando los datos.....	3
b.	Ingestión de datos.....	3
c.	Spark	4
d.	Introducción a Scala	5
e.	¿Scala en la consola de Spark o en IntelliJ?	5
f.	Problemas en Scala	5
g.	Python en Visual Studio Code	6
h.	Librerías Requests y Json	6
i.	Concatenación	7
j.	Franja horaria.....	7
i.	Resultado de la concatenación	8
k.	Peso de datos.....	8
l.	MongoDB	9
m.	Power BI.....	10
3.	Programación de Inteligencia Artificial y Sistemas de aprendizaje automático.....	12
a.	Selección de datos para su posterior análisis.....	12
b.	Google Colab	12
c.	Predicciones con Colab	25
1.	Árbol de decisiones.....	25
2.	Regresión lineal.....	26
d.	Red neuronal.....	30
e.	BigML	31
•	Algoritmo supervisado	33
•	Algoritmo No Supervisado	34
f.	Power BI.....	35
4.	Conclusión general.....	36
5.	Webgraffía.....	37
6.	Anexo	38
a.	Código Spark-Scala.....	38
b.	Código Python	39
c.	Consultas a MongoDB con PyMongo.....	43
a.	Colab	53
b.	Red neuronal.....	63

1. Introducción

En este documento presento la finalización del aprendizaje de la especialización a lo largo de todas las horas que lo componen.

Para ello he elegido analizar toda la red de sonómetros de Bilbao y hemos seleccionado unos datos de tráfico a ver si tienen alguna incidencia en los datos presentados.

Las muestras de datos de sonómetros son recogidas desde hace un mes hasta el día de hoy y se actualizan cada 15 minutos, mientras que las muestras de tráfico son recogidas en Bilbao las últimas 24 horas.

2. Sistemas de Aprendizaje Automático y Big Data Aplicado

a. Seleccionando los datos.

Para empezar, se han seleccionado datos de distintas fuentes y distintos formatos:

Para los sonómetros hemos seleccionado un Json con los datos de muestreo y un GeoJson que define la ubicación de estos expresando coordenadas en latitud y longitud.

Esto lo hemos recogido de la web de Open data Euskadi:

https://www.bilbao.eus/aytoonline/jsp/opendata/movilidad/od_sonometro_mediciones.jsp?idioma=c&formato=json

https://www.bilbao.eus/aytoonline/jsp/opendata/movilidad/od_sonometro_ubicacion.jsp?idioma=c&formato=geojson

Para el tráfico hemos seleccionado un GeoJson que indicará un mapa de zonas y con ello, podremos ver si se puede relacionar con los muestreos del sonómetro.

Esto lo hemos recogido de la Web del ayuntamiento de Bilbao del Data Set de tráfico:

<https://www.bilbao.eus/aytoonline/srvDatasetTrafico?formato=geojson>

b. Ingestión de datos

Para realizar la ingestión de datos, hemos considerado muchas opciones, pero nos hemos quedado con Spark y Python mediante Visual Studio Code.



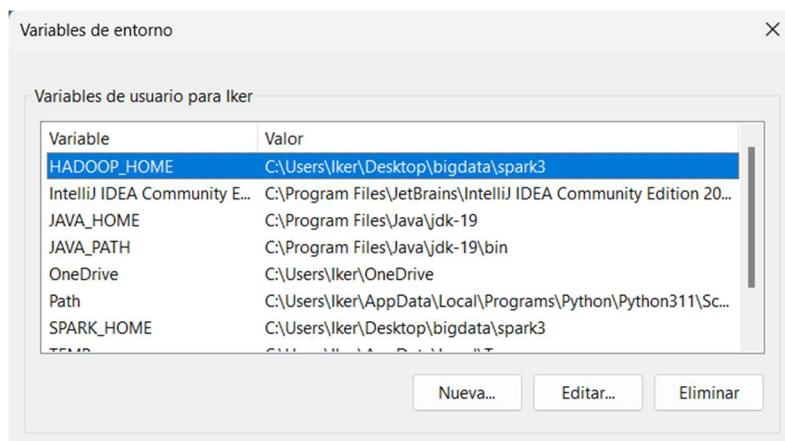
c. Spark

¿Qué es Spark?

Spark es un framework de computación distribuida que permite procesar grandes cantidades de datos en clústeres de servidores de manera rápida y eficiente. Proporciona una interfaz de programación para trabajar con datos en memoria y en disco, y cuenta con módulos para procesamiento de datos en tiempo real, análisis de datos y machine learning. Spark se ejecuta en una variedad de sistemas de procesamiento de clústeres, incluyendo Hadoop, Mesos y Kubernetes, y es compatible con varios lenguajes de programación, como Java, Scala, Python y R. Con su arquitectura de procesamiento en memoria, Spark puede procesar grandes conjuntos de datos hasta 100 veces más rápido que Hadoop MapReduce, lo que lo hace ideal para aplicaciones de análisis de big data.

Hemos seleccionado Spark para hacer la ingesta de datos del Json que corresponde a las mediciones del sonómetro, para ello, vamos a instalar Spark en el sistema operativo Windows 11, y configurar las siguientes variables de entorno, con sus respectivos directorios como podemos ver en la figura:

- HADOOP_HOME
 - JAVA_HOME
 - JAVA_PATH
 - SPARK_HOME



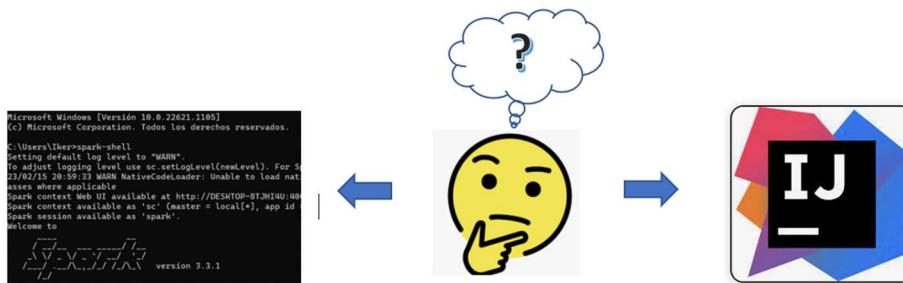
Para conectar Spark nos hemos metido en la consola de Windows y hemos introducido el comando Spark-Shell. A continuación, podremos trabajar con Scala.

d. Introducción a Scala

Scala es uno de los lenguajes de programación más populares para trabajar en Spark. La razón principal de su popularidad es que Scala es compatible con la máquina virtual de Java (JVM), lo que significa que puede ejecutarse en cualquier plataforma que tenga la JVM instalada. Además, Scala es un lenguaje de programación orientado a objetos y funcional, lo que hace que sea muy flexible para trabajar con grandes conjuntos de datos y procesamiento distribuido en Spark.

e. ¿Scala en la consola de Spark o en IntelliJ?

Ejecutar Scala en la consola de Spark es una forma rápida y sencilla de probar comandos de Spark. Es útil para tareas simples como explorar datos, para nuestro proyecto más que suficiente, ya que solo queremos hacer la ingestión del JSON de las mediciones de los sonómetros.



Por otro lado, IntelliJ Community es un programa más completo y poderoso que te ayuda a escribir, depurar y ejecutar aplicaciones de Spark más complejas. Proporciona herramientas avanzadas como autocompletado y depuración de código. Se utiliza para el desarrollo de aplicaciones más avanzadas y complejas.

En el proyecto se ha utilizado, pero al ejecutar ha dado un conflicto de versiones de Java, se ha podido arreglar, pero veía que era más simple y rápido hacer la ingestión en la consola de Spark.

El código completo se puede ver en el [anexo](#).

f. Problemas en Scala

En el proyecto se ha utilizado IntelliJ Community, pero al ejecutar ha dado un conflicto de versiones de Java, después se ha podido arreglar, pero al ver que era más simple y rápido hacer la ingestión en la consola de Spark, al final se ha optado por la consola. Para ello, se ha deshabilitado

Otro de los problemas es que para obtener el JSON es que el servidor que aloja el JSON está configurado para requerir una conexión segura y verificar la identidad del cliente que está solicitando el JSON a través de un certificado SSL. Si el certificado SSL del cliente no es válido o no se puede verificar, el servidor puede rechazar la conexión y negarse a enviar el JSON.

Para solucionarlo se ha deshabilitado la verificación del certificado SSL en la primera parte de la ejecución del programa, lo que permite conectarse a la URL incluso si el servidor requiere una conexión segura y tiene un certificado SSL no válido o no verificado.

g. Python en Visual Studio Code

¿Qué es Python?

Python es un lenguaje de programación de alto nivel que se utiliza para desarrollar aplicaciones en diversos ámbitos, incluyendo desarrollo web, ciencia de datos, inteligencia artificial, automatización de tareas, entre otros. Es un lenguaje de programación interpretado, lo que significa que el código se ejecuta línea por línea, y no necesita ser compilado previamente como otros lenguajes de programación.

El entorno que hemos elegido es Visual Studio Code, ya que nos permite cargar múltiples extensiones y a través de la consola podemos cargar muchas de las librerías necesarias para abordar el proyecto.

Para el proyecto se ha hecho la ingesta del Json en local, y dos GeoJson en URL. Para leer el archivo JSON local, se utiliza la librería "json" para convertir el archivo JSON en un objeto Python y para obtener los datos de los archivos GeoJSON en línea, se puede utilizar la librería "requests" para enviar solicitudes HTTP y recibir respuestas con la función "get".

h. Librerías Requests y Json

- Requests

La librería "requests" permite enviar solicitudes HTTP a servidores web y recibir respuestas en Python. Permite trabajar con diferentes métodos HTTP, como GET, POST, PUT y DELETE, y admite una amplia gama de opciones de configuración, como la definición de encabezados HTTP, parámetros de URL, autenticación y certificados SSL. "Requests" hace que el trabajo con HTTP sea más fácil y eficiente, pudiendo obtener de inmediato los datos y almacenarlos.

```
try:
    r_geo = requests.get(url_geo)
    r_geo.raise_for_status()
    print("*****")
    print("*Conectado a datos GEOJSON UBICACIÓN SONOMBROS*")
    print("*****")
```

- Json

La librería "json" permite trabajar con datos en formato JSON en Python. JSON es un formato de intercambio de datos muy popular utilizado por muchas aplicaciones web. "json" convierte los datos JSON en objetos Python, lo que permite acceder y manipular los datos en Python con facilidad. La librería "json" admite la conversión de datos JSON a diccionarios, listas y cadenas de caracteres, lo que permite la manipulación de los datos en diferentes formas. Además, la librería "json" también admite la validación y la codificación de datos JSON para asegurarse de que se ajusten a las especificaciones del formato.

```
#verificamos si es correcta la conexión es correcta a Json
try:
    #conectar a carpeta local
    respuesta= requests.Response()
    respuesta._content = json.dumps(contenido).encode('utf-8')
    respuesta.status_code = 200
    respuesta.headers['content-type'] = 'application/json'
```

i. Concatenación

Es la combinación de datos de ambos archivos en una estructura de datos común, en resumen, un diccionario de Python, que luego se puede guardar en un archivo Json.

En el proyecto para hacer la concatenación, hemos relacionado dos valores comunes que representan el nombre del dispositivo, entre los datos del Json y del GeoJson para hacer una estructura de datos común, más limpias a la hora de guardar en una base de datos o para realizar una presentación de datos en Power BI.

```
# Crea un nuevo diccionario que combina los datos de ambos diccionarios
concatenated_data = []
for item_json in r:
    name = item_json["nombre_dispositivo"]
    for item_geojson in r.geo["features"]:
        if item_geojson["properties"]["name"] == name:
            concatenated_item = item_json
            concatenated_item["address"] = item_geojson["properties"]["address"]
            concatenated_item["status"] = item_geojson["properties"]["status"]

            #separar fecha_medicion creando hora y fecha por separado
            fecha_medicion = item_json["fecha_medicion"]
            fecha_medicion = fecha_medicion.split(" ")
            fecha = fecha_medicion[0]
            hora = fecha_medicion[1]
            concatenated_item["fecha"] = fecha
            concatenated_item["hora"] = hora
```

j. Franja horaria

Hemos separado la fecha de la hora, y las horas las hemos categorizado como franja horaria, trabajaremos con cuatro franjas horarias, mañana, tarde, noche y madrugada

```
#formato de hora
hora = hora.split(":")
hora = hora[0]
hora = int(hora)

if hora >= 0 and hora <= 6:
    concatenated_item["hora"] = "Madrugada"
elif hora >= 7 and hora <= 12:
    concatenated_item["hora"] = "Mañana"
elif hora >= 13 and hora <= 20:
    concatenated_item["hora"] = "Tarde"
elif hora >= 21 and hora <= 23:
    concatenated_item["hora"] = "Noche"
```

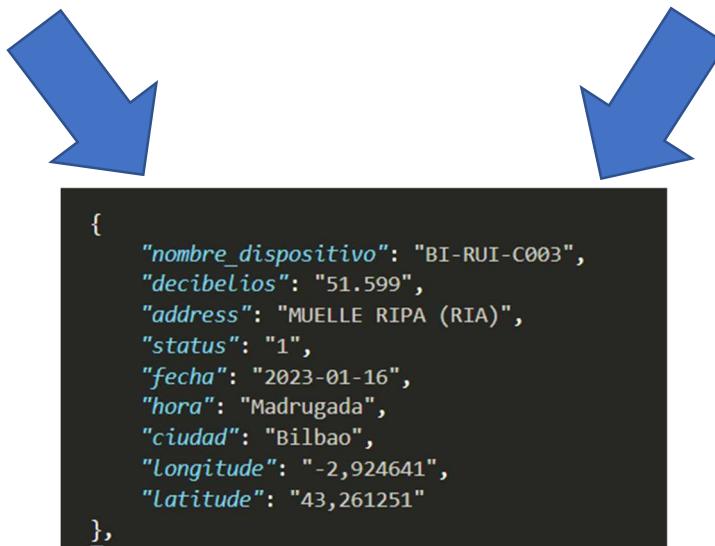
Ahora obtendremos unos datos más limpios, estructurados y unificados, será la que defina nuestra hoja de ruta de aquí en adelante, pudiendo almacenar estos datos en MongoDB y trabajar con ellos mismos en Power BI, Colab y BigML.



i. Resultado de la concatenación

```
[{"nombre_d": "BI-RUI-006", "decibelios": "55.801", "fecha_medicion": "2023-01-16 00:00:25.347"}, {"nombre_d": "BI-RUI-007", "decibelios": "71.792", "fecha_medicion": "2023-01-16 00:00:26.097"}, {"nombre_d": "BI-RUI-BR11", "decibelios": "42.60", "fecha_medicion": "2023-01-16 00:03:23.997"}, {"nombre_d": "BI-RUI-C008", "decibelios": "56.758", "fecha_medicion": "2023-01-16 00:03:53.123"}, {"nombre_d": "BI-RUI-P006", "decibelios": "72.414", "fecha_medicion": "2023-01-16 00:03:53.123"}, {"nombre_d": "BI-RUI-010", "decibelios": "72.414", "fecha_medicion": "2023-01-16 00:03:53.123"}]
```

```
{
  "type": "FeatureCollection",
  "features": [
    {
      "type": "Feature",
      "properties": {
        "name": "BI-RUI-001",
        "serialNumber": "051999999991001",
        "status": "1",
        "address": "RODRIGUEZ ARIAS 71 BIS",
        "deviceTypeId": "1",
        "longitude": "-2.944465",
        "latitude": "43.263875"
      },
      "geometry": {
        "type": "point",
        "coordinates": [
          -2.944465,
          43.263875
        ]
      }
    }
  ]
}
```



En el nuevo Json hemos asociado el nombre del dispositivo a la calle correspondiente, hemos añadido la ciudad y el status de conexión, y hemos separado la fecha y hora antigua, generando la fecha actual y la franja horario donde se produce la medición, además de todo esto, hemos cambiado los puntos de la longitud y latitud por comas, para que sean compatibles al formato de Power BI.

k. Peso de datos

Después de hacer limpieza de datos, generaremos un nuevo Json, y a su vez un CSV para importar a Power BI, que hace los datos más manejables debido a su menor peso, teniendo en cuenta las coordenadas y los decibelios, ya que estos están **separados por puntos**, **Power Bi los lee con comas**, para solucionarlo crearemos el CSV **separado por punto y coma**.

Entonces también instalaremos el Json concatenado en una base de datos no relacional, para ello utilizaremos MongoDB.

CSV → Power BI

I. MongoDB

MongoDB Compass es una herramienta gráfica de usuario desarrollada por MongoDB, que permite explorar y visualizar los datos almacenados en una base de datos MongoDB. Es una interfaz gráfica de usuario fácil de usar que ofrece una vista de árbol de la estructura de la base de datos y una vista de tabla para los documentos almacenados en las colecciones.

Conectaremos nuestra base de datos y la colección al localhost 27017.

The screenshot shows the MongoDB Compass interface connected to the 'sonometros' database. On the left, the 'sonometros' collection is selected under the 'sonometros' database. The main pane displays two documents from the 'sonometros.sonometros' collection. The first document has the following fields:

```

_id: ObjectId('63f340f96c2813c6078d67b4')
nombre_dispositivo: "BI-RUI-C003"
decibelios: 51.59
address: "MUELLE RIPA (RIA)"
status: "1"
fecha: "2023-01-16"
hora: "Madrugada"
ciudad: "Bilbao"
longitude: "-2,924641"
latitude: "43,261251"

```

The second document has the following fields:

```

_id: ObjectId('63f340f96c2813c6078d67b5')
nombre_dispositivo: "BI-RUI-006"
decibelios: 55.801
address: "BARRENKALE 7"
status: "1"
fecha: "2023-01-16"
hora: "Madrugada"
ciudad: "Bilbao"
longitude: "-2,925136"
latitude: "43,256697"

```

Por otro lado, Pymongo es un controlador de base de datos para MongoDB que permite a los desarrolladores de Python interactuar con bases de datos MongoDB. Pymongo es una biblioteca de Python que permite a los desarrolladores interactuar con MongoDB desde el código Python. Con Pymongo, los desarrolladores pueden escribir scripts y aplicaciones en Python que se conectan y manipulan datos en una base de datos MongoDB. Es una herramienta esencial para cualquier desarrollador de Python que trabaje con bases de datos MongoDB.

Para conectar Pymongo hemos instalado MongoDB Compass que nos facilita la conexión y para comunicarlo con Visual Studio Code hemos instalado la correspondiente extensión, hemos conectado a lo generado desde Compass que es el localhost 27017.

The screenshot shows the MongoDB Compass interface on the left and the Visual Studio Code interface on the right. A large green arrow points from the MongoDB Compass window to the Visual Studio Code window. In the Visual Studio Code window, the 'CONNECTIONS' section shows a connection to 'localhost:27017 connected'. The 'PLAYGROUNDS' section indicates 'No .mongodb playground files found in the workspace.' and has a 'Create New Playground' button.

En PyMongo hemos creado el espacio de trabajo de sonómetros para almacenaje de la información, y hemos realizado consultas algunas utilizando aggregate en un programa de Python con un menu de las consultas más recurrentes, que nos ayudará a sacar conclusiones a posteriori.

```
#menu consultas
print("1. ¿Cuántos registros hay en la base de datos de los sonómetros de Bilbao?")
print("2. Mostrar calles en Bilbao que cuentan con sonómetros")
print("3. ¿Cuántas veces ha contado un sonómetro en cada calle?")
print("4. ¿Cuál es el promedio de la calle con los decibelios más altos")
print("5. ¿Cuál es el promedio de la calle con los decibelios más bajos")
print("6. ¿Cuál es el registro más alto en decibelios de un sonómetro")
print("7. ¿Cuál es el registro más bajo en decibelios de un sonómetro")
print("8. Imprimir calles ordenadas de mayor a menor por promedio de decibelios")
print("9. ¿En cuál franja horaria(hora) se producen más decibelios?")
print("10. ¿En cuál franja horaria(hora) se producen menos decibelios?")
print("11. ¿En qué fecha se han producido más decibelios?")
print("12. ¿En qué fecha se han producido menos decibelios?")
print("13. ¿En qué fecha y hora se han producido más decibelios?")
print("14. ¿En qué fecha y hora se han producido menos decibelios?")
print("15. Promedio de decibelios de todo Bilbao")
print("16. Salir")
```

En la base de datos se guardaba como string en MongoDB, el campo decibelios entonces para realizar los cálculos hemos convertido el string a float.

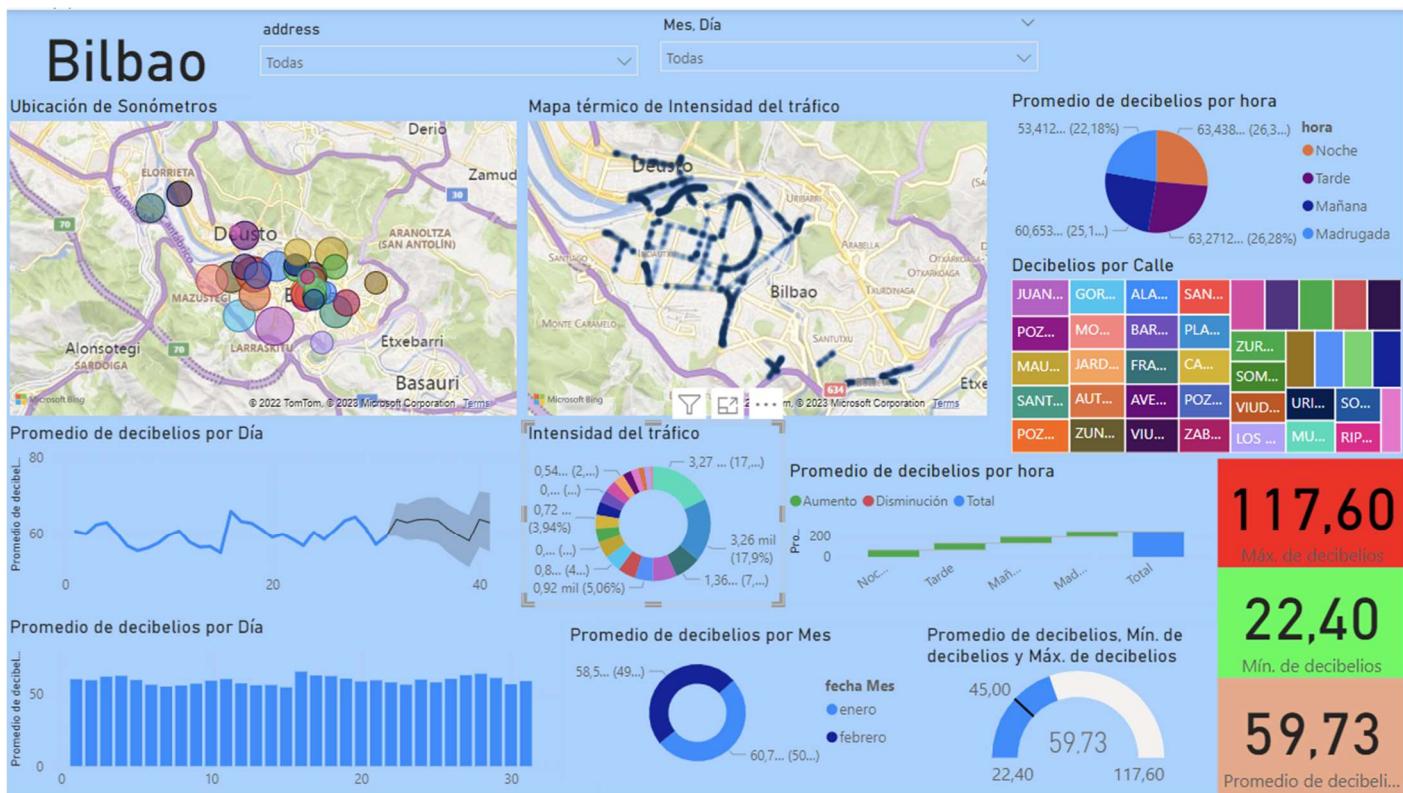
El código completo se puede consultar en el [anexo](#), junto con su salida.

m. Power BI

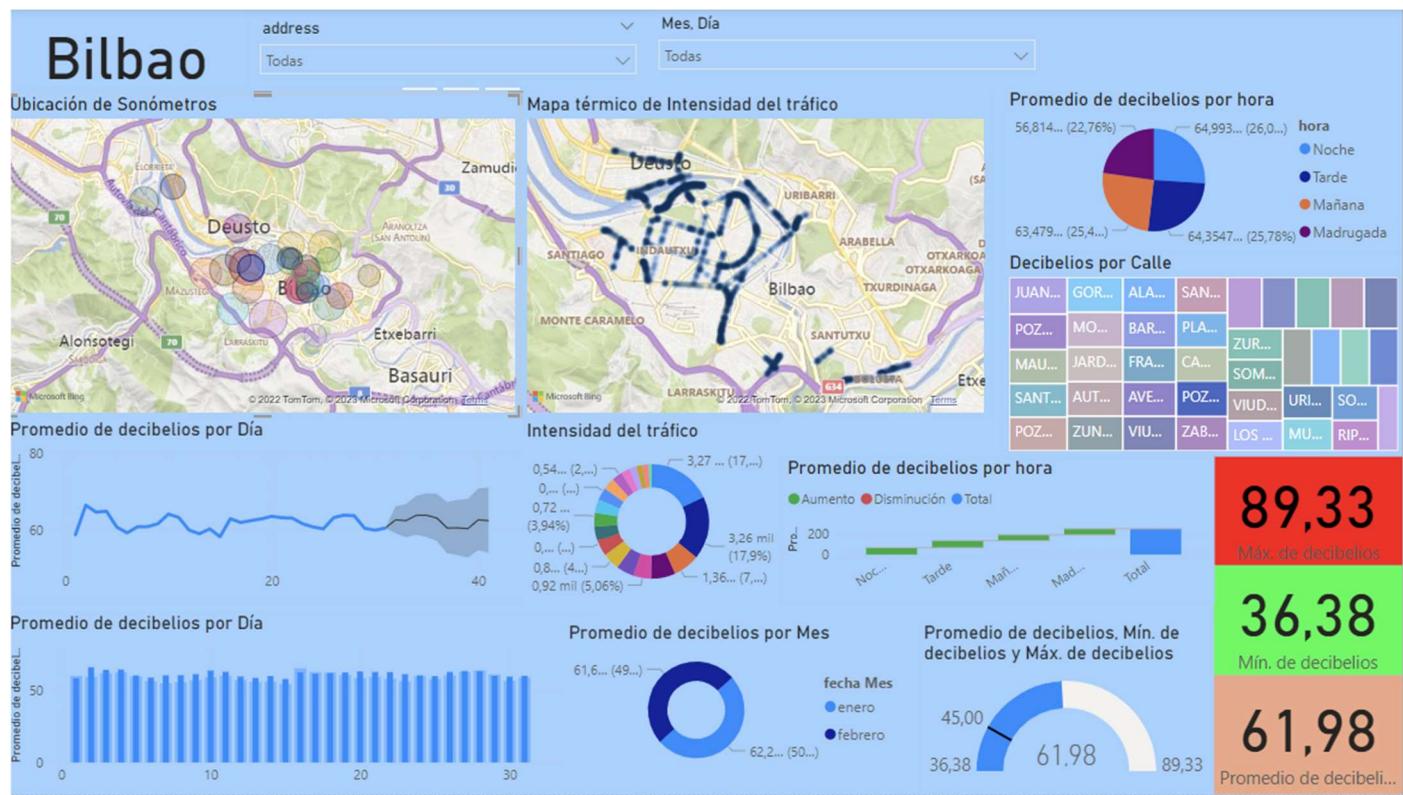
Power BI es una herramienta de análisis de datos que permite conectar y transformar datos de múltiples fuentes, crear visualizaciones interactivas y compartir informes y paneles de control para tomar decisiones basadas en datos.

- Hemos exportado los datos a través del CSV generado en Python, en Power Bi hemos generado el siguiente informe con estas características:
- Hemos creado 2 segmentaciones de datos para mostrar, las calles y los días del mes en la cabecera del Power BI
- Hemos creado un gráfico circular con el promedio de decibelios por franja horaria.
- Hemos situado 2 mapas en Bilbao mediante coordenadas, uno refleja la posición de los sonómetros y otro la intensidad del tráfico.
- Hemos hecho un promedio de los decibelios por día en un gráfico de líneas, creando una predicción.
- Hemos creado 4 tarjetas simples con el nombre de la ciudad, el máximo y mínimo de decibelios, y el promedio de ellos.
- Hemos creado un “Treemap” que ordena los decibelios por promedio total por calles.
- Hemos creado una gráfica en cascada en la que se pueden ver el aumento o disminución de decibelios por franja horaria.

Ejemplo con todo seleccionado:



Ejemplo con una calle seleccionada:



3. Programación de Inteligencia Artificial y Sistemas de aprendizaje automático

a. Selección de datos para su posterior análisis.

Para nuestro posterior análisis de los datos vamos a seleccionar 3 archivos para poder trabajar con ellos en Colab.

- concatenated_data.json
- ubicacionsonometros.geojson
- ubicaciontrafico.geojson

b. Google Colab

¿Qué es Google Colab?

Es un servicio gratuito de nube alojado por Google para fomentar la investigación sobre Aprendizaje de Máquina e inteligencia Artificial.

En el trabajaremos de forma estadística nuestros archivos, crearemos gráficas y algoritmos de predicción.

Primero de todo llamamos a las librerías necesarias para trabajar y cargamos los 3 archivos:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
import pickle
from sklearn.neighbors import NearestNeighbors
from sklearn.metrics import log_loss
from sklearn.metrics import mean_squared_error
from sklearn.metrics import f1_score
from google.colab import files

uploaded = files.upload()
```

Declaramos el archivo JSON o CSV de datos concatenados:

```
dfsonometros = pd.read_json('concatenated_data.json')
print(dfsonometros)
```

	nombre_dispositivo	decibelios	address	status	\	
0	BI-RUI-C003	51.599	MUELLE RIPA (RIA)	1		
1	BI-RUI-006	55.801	BARRENKALE 7	1		
2	BI-RUI-001	71.792	RODRIGUEZ ARIAS 71 BIS	1		
3	BI-RUI-004	58.390	PLAZA UNAMUNO FAROLA 13	1		
4	BI-RUI-019	62.662	ZUNZUNEGUI JUNTO BOCA METRO	1		
...	
168873	BI-RUI-006	54.549	BARRENKALE 7	1		
168874	BI-RUI-004	64.249	PLAZA UNAMUNO FAROLA 13	1		
168875	BI-RUI-011	72.635	JARDINES 6	1		
168876	BI-RUI-019	69.399	ZUNZUNEGUI JUNTO BOCA METRO	1		
168877	BI-RUI-C005	55.700	SOMERA 8	1		
	fecha	hora	ciudad	longitude	latitude	
0	2023-01-16	Madrugada	Bilbao	-2,924641	43,261251	
1	2023-01-16	Madrugada	Bilbao	-2,925136	43,256697	
2	2023-01-16	Madrugada	Bilbao	-2,944465	43,263875	
3	2023-01-16	Madrugada	Bilbao	-2,921691	43,258396	
4	2023-01-16	Madrugada	Bilbao	-2,948616	43,261587	
...	
168873	2023-02-15	Tarde	Bilbao	-2,925136	43,256697	
168874	2023-02-15	Tarde	Bilbao	-2,921691	43,258396	
168875	2023-02-15	Tarde	Bilbao	-2,925043	43,258266	
168876	2023-02-15	Tarde	Bilbao	-2,948616	43,261587	
168877	2023-02-15	Tarde	Bilbao	-2,922789	43,256406	

[168878 rows x 9 columns]

Ahora nos toca trabajar con el geoJson con la ubicación de los sonómetros, para ello debemos instalar la librería geopandas, así que cargamos la ubicación de los sonómetros.

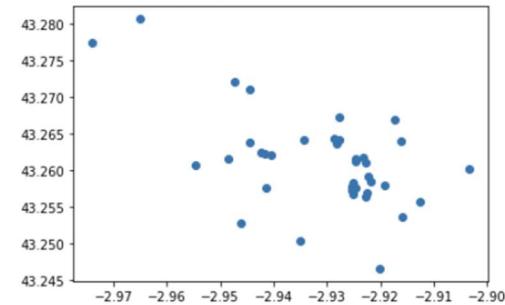
```
[5] import geopandas as gpd
[6]
[7] dfubicacion = gpd.read_file('ubicacionsonometros.geojson')
dfubicacion.head(3)
```

	name	serialNumber	status	address	deviceTypeId	longitude	latitude	geometry
0	BI-RUI-001	051999999991001	1	RODRIGUEZ ARIAS 71 BIS	1	-2.944465	43.263875	POINT (-2.944467 43.26388)
1	BI-RUI-004	051999999991004	1	PLAZA UNAMUNO FAROLA 13	1	-2.921691	43.258396	POINT (-2.92169 43.25840)
2	BI-RUI-006	051999999991006	1	BARRENKALE 7	1	-2.925136	43.256697	POINT (-2.92514 43.25670)

Al obtener el geojson podemos comprobar que tenemos un campo con coordenadas llamado “**Geometry**”, en el tenemos tres valores, POINT que es la definición de la geometría en este caso es un punto en un eje de coordenadas de posición, x e y, siendo x = longitud e y = latitud, el siguiente paso será graficar las coordenadas sobre un plano, una vez graficadas podemos ver la ubicación sobre plano de los sonómetros.

```
[ ] dfubicacion.plot()
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f38a766fd30>
```



El siguiente paso será hacer lo mismo con la densidad del tráfico, en este caso tenemos el área geométrica definida por polígonos en vez de puntos:

```
▶ print(dftrafico)
```

	CodigoSección	Ocupacion	Intensidad	Velocidad	FechaHora
0	248	0	68	23	2023-02-16 15:35:00
1	273	15	983	14	2023-02-16 15:35:00
2	274	54	0	0	2023-02-16 15:35:00
3	275	7	1041	32	2023-02-16 15:35:00
4	276	6	645	13	2023-02-16 15:35:00
..
76	392	11	1114	17	2023-02-16 15:35:00
77	393	10	2181	38	2023-02-16 15:35:00
78	396	6	819	24	2023-02-16 15:35:00
79	398	3	320	10	2023-02-16 15:35:00
80	399	0	-432	0	2023-02-16 15:35:00

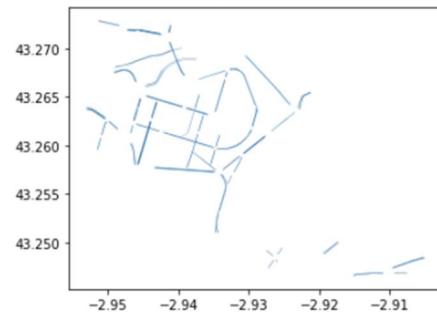
```
geometry
0  POLYGON ((-2.92908 43.26388, -2.92984 43.26387...
1  POLYGON ((-2.94672 43.26119, -2.94676 43.26120...
2  POLYGON ((-2.94624 43.26247, -2.94624 43.26247...
3  POLYGON ((-2.94567 43.26387, -2.94530 43.26475...
4  POLYGON ((-2.94547 43.26489, -2.94587 43.26392...
.  ...
76  POLYGON ((-2.95305 43.26367, -2.95283 43.26367...
77  POLYGON ((-2.95025 43.26284, -2.95102 43.26325...
78  POLYGON ((-2.95020 43.26252, -2.95020 43.26251...
79  POLYGON ((-2.95105 43.26080, -2.95154 43.25959...
80  POLYGON ((-2.95149 43.25954, -2.95126 43.26008...
```

[81 rows x 6 columns]

Obtenemos las coordenadas poligonales y serán más marcadas o menos marcadas, en nivel a la densidad de tráfico total:

```
▶ dftrafico.plot()
```

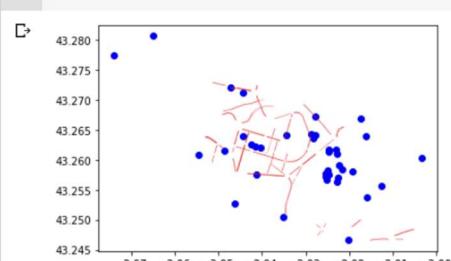
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f38a71d24c0>
```



Realizamos la fusión de los dos espacios en uno solo, y podemos ver los sonómetros que están cerca de los puntos con mayor intensidad de tráfico.

```
fig, ax = plt.subplots()
```

```
# Graficar el primer dataframe con color azul
dfubicacion.plot(ax=ax, color='blue')
# Graficar el segundo dataframe con color rojo
dftrafico.plot(ax=ax, color='red')
plt.show()
```

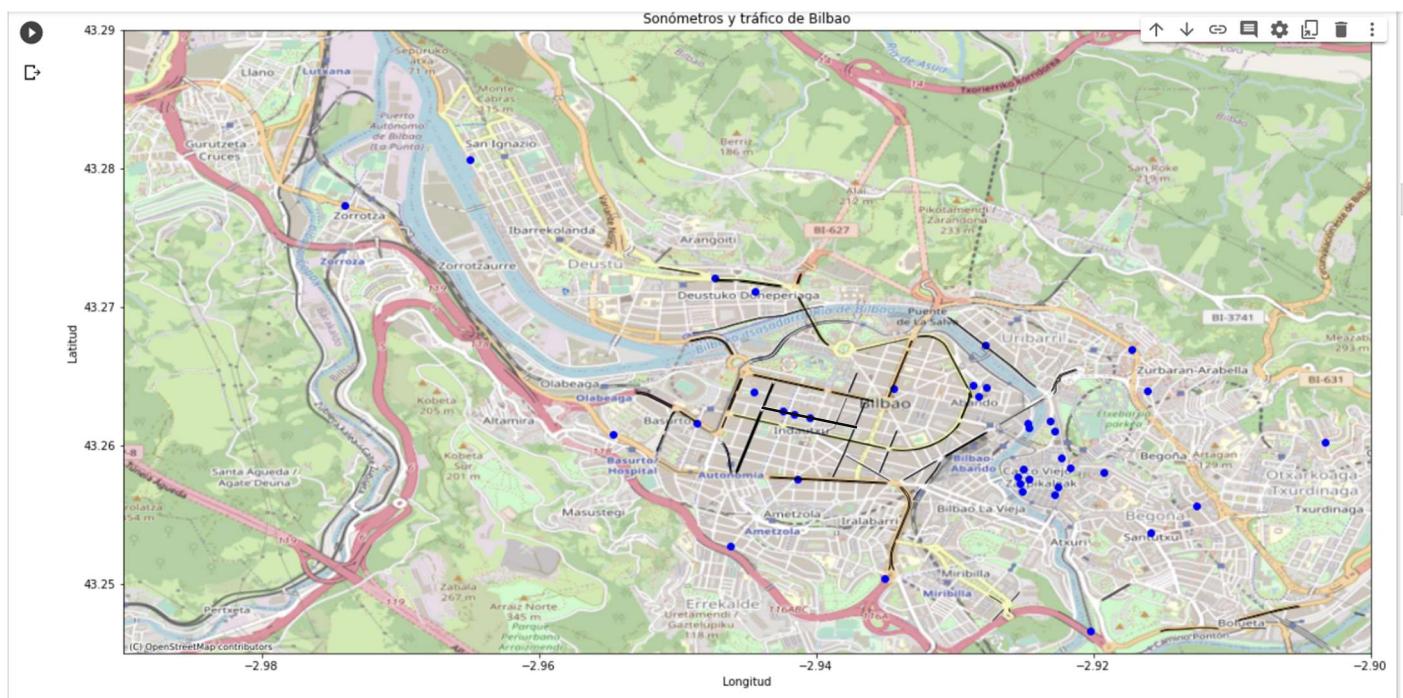


Para añadir un poco de realismo a la gráfica vamos a añadir un mapa de Bilbao con la librería contextily, que como su propio nombre indica crea un mosaico geográfico de contexto en Python.

Muy importante extraer las coordenadas anteriores, el máximo y el mínimo de latitud y longitud para que muestre correctamente la gráfica.

```
import matplotlib.pyplot as plt
import contextily as ctx

# Crear una figura y un conjunto de ejes para la gráfica
fig, ax = plt.subplots()
#coordenadas bilbao
ax.set_xlim(-2.99, -2.90)
ax.set_ylim(43.245, 43.290)
# Añadir un título
ax.set_title("Sonómetros y tráfico de Bilbao")
# Añadir una etiqueta para el eje X
ax.set_xlabel("Longitud")
# Añadir una etiqueta para el eje Y
ax.set_ylabel("Latitud")
# Graficar el primer dataframe con color azul
dfubicacion.plot(ax=ax, color='blue')
# Graficar el segundo dataframe con color rojo
dftrafico.plot(ax=ax, color='black')
# Agregar un mapa base de OpenStreetMap
ctx.add_basemap(ax, crs=dfubicacion.crs.to_string(), source=ctx.providers.OpenStreetMap.Mapnik)
# Mostrar la gráfica
plt.show()
```

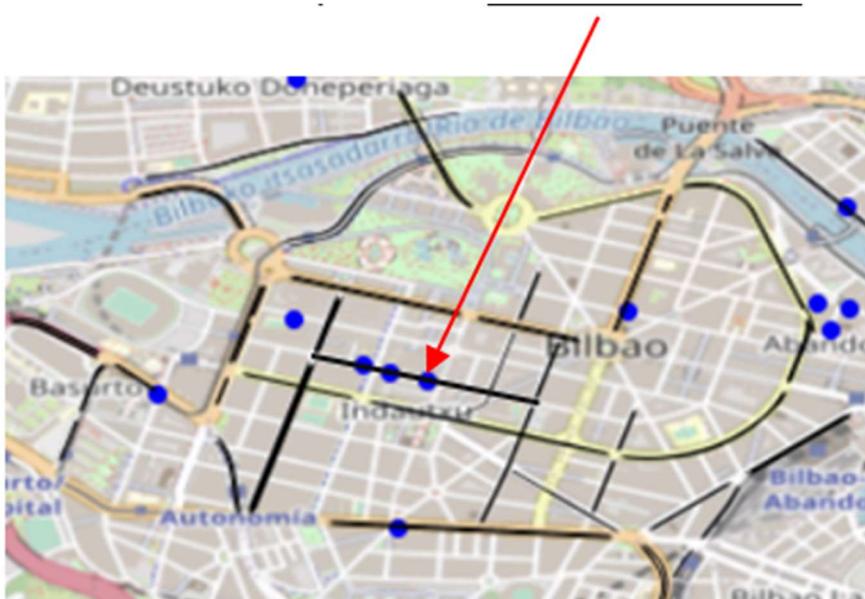


Entonces vamos a clasificar los decibelios más altos registrados por los sonómetros a partir del rango de 100 decibelios ordenados de mayor a menor usando el método `sort_values()` de Pandas.

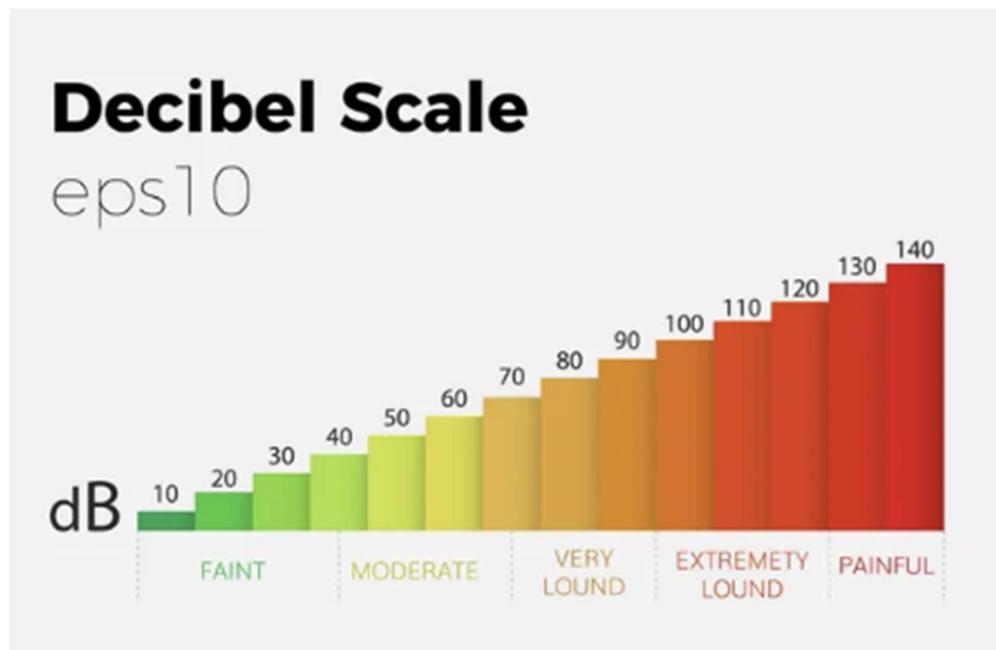
```
result = dfsonometros.loc[dfsonometros['decibelios'] > 100]
result = result.sort_values(by='decibelios', ascending=False)
print(result)
```

	nombre_dispositivo	decibelios	address	status	fecha	hora	ciudad	longitude	latitude
47265	BI-RUI-C005	117.6	SOMERA 8	1	47265	2023-01-24	Tarde	-2,922789	43,256406
118446	BI-RUI-C005	116.8	SOMERA 8	1	118446	2023-02-06	Tarde	-2,922789	43,256406
80431	BI-RUI-C006	115.1	VIUDA DE EPALZA 3	1	80431	2023-01-30	Tarde	-2,922792	43,261024
47263	BI-RUI-C005	113.8	SOMERA 8	1	47263	2023-01-24	Tarde	-2,922789	43,256406
163163	BI-RUI-021	113.0	POZA 53	1	163163	2023-02-14	Tarde	-2,941606	43,262206
48432	BI-RUI-021	112.9	POZA 53	1	48432	2023-01-24	Tarde	-2,941606	43,262206
151245	BI-RUI-021	112.5	POZA 53	1	151245	2023-02-12	Mañana	-2,941606	43,262206
137883	BI-RUI-021	112.4	POZA 53	1	137883	2023-02-10	Madrugada	-2,941606	43,262206
108232	BI-RUI-021	112.4	POZA 53	1	108232	2023-02-04	Tarde	-2,941606	43,262206
148125	BI-RUI-021	112.2	POZA 53	1	148125	2023-02-11	Noche	-2,941606	43,262206
32629	BI-RUI-021	112.0	POZA 53	1	32629	2023-01-21	Noche	-2,941606	43,262206
129173	BI-RUI-021	110.3	POZA 53	1	129173	2023-02-08	Mañana	-2,941606	43,262206
28154	BI-RUI-021	110.3	POZA 53	1	28154	2023-01-21	Madrugada	-2,941606	43,262206
13630	BI-RUI-021	110.2	POZA 53	1	13630	2023-01-18	Mañana	-2,941606	43,262206
14322	BI-RUI-021	110.1	POZA 53	1	14322	2023-01-18	Tarde	-2,941606	43,262206
143932	BI-RUI-021	109.8	POZA 53	1	143932	2023-02-11	Madrugada	-2,941606	43,262206
163402	BI-RUI-021	109.7	POZA 53	1	163402	2023-02-14	Tarde	-2,941606	43,262206
156488	BI-RUI-021	109.7	POZA 53	1	156488	2023-02-13	Mañana	-2,941606	43,262206
135786	BI-RUI-021	109.6	POZA 53	1	135786	2023-02-09	Tarde	-2,941606	43,262206
163745	BI-RUI-021	109.6	POZA 53	1	163745	2023-02-14	Tarde	-2,941606	43,262206
138213	BI-RUI-021	109.5	POZA 53	1	138213	2023-02-08	Tarde	-2,941606	43,262206
135375	BI-RUI-021	109.5	POZA 53	1	135375	2023-02-09	Tarde	-2,941606	43,262206
148301	BI-RUI-021	109.5	POZA 53	1	148301	2023-02-11	Noche	-2,941606	43,262206
153810	BI-RUI-021	109.5	POZA 53	1	153810	2023-02-12	Noche	-2,941606	43,262206
155554	BI-RUI-021	109.5	POZA 53	1	155554	2023-02-13	Madrugada	-2,941606	43,262206
135493	BI-RUI-021	109.4	POZA 53	1	135493	2023-02-09	Tarde	-2,941606	43,262206

- Observando estos resultados vemos que la mayoría de las incidencias se producen fuera del fin de semana y la franja horaria más habitual es la de la tarde.
- La mayoría de los sonidos por encima de **100 decibelios** son extremadamente molestos, lo cual nos hace deducir que ha sido a consecuencia del tráfico.
- **Poza 53** es una calle con mucho tráfico a lo largo del día, hay múltiples zonas de carga y descarga.
- Para verlo más claro en el mapa **Poza 53** corresponde a [este sonómetro](#):



Para saber un poco la magnitud del tipo de sonido de los que estamos hablando, vamos a ver una tabla con la escala de decibelios.



Ahora incluiremos una función de Python para categorizar los datos en función de la escala:

```

def categorize_sound(decibels):
    conditions = [decibels < 20, (decibels >= 20) & (decibels < 40), (decibels >= 40) & (decibels < 60), (decibels >= 60) & (decibels < 80),
                  (decibels >= 80) & (decibels < 100), (decibels >= 100) & (decibels < 120), decibels >= 120]
    choices = ['Suave', (variable) condiciones: list fuerte', 'Molesto', 'Extremadamente Molesto', 'Doloroso']
    return np.select(conditions, choices, default='Invalido')

result['categoría_sonido'] = result['decibelios'].apply(categorize_sound)

print(result)

```

nombre_dispositivo	decibelios	address	status	categoría_sonido
47265	BI-RUI-C005	117.6	SOMERA 8	1 Extremadamente Molesto
118446	BI-RUI-C005	116.8	SOMERA 8	1 Extremadamente Molesto
80431	BI-RUI-C006	115.1	VIUDA DE EPALZA 3	1 Extremadamente Molesto
47263	BI-RUI-C005	113.8	SOMERA 8	1 Extremadamente Molesto
163163	BI-RUI-021	113.0	POZA 53	1 Extremadamente Molesto
48432	BI-RUI-021	112.9	POZA 53	1 Extremadamente Molesto
151245	BI-RUI-021	112.5	POZA 53	1 Extremadamente Molesto
137883	BI-RUI-021	112.4	POZA 53	1 Extremadamente Molesto
108232	BI-RUI-021	112.4	POZA 53	1 Extremadamente Molesto
148125	BI-RUI-021	112.2	POZA 53	1 Extremadamente Molesto
32629	BI-RUI-021	112.0	POZA 53	1 Extremadamente Molesto
129173	BI-RUI-021	110.3	POZA 53	1 Extremadamente Molesto
28154	BI-RUI-021	110.3	POZA 53	1 Extremadamente Molesto
13630	BI-RUI-021	110.2	POZA 53	1 Extremadamente Molesto
14322	BI-RUI-021	110.1	POZA 53	1 Extremadamente Molesto
143932	BI-RUI-021	109.8	POZA 53	1 Extremadamente Molesto
163402	BI-RUI-021	109.7	POZA 53	1 Extremadamente Molesto
156488	BI-RUI-021	109.7	POZA 53	1 Extremadamente Molesto

Como resultado obtenemos que se da un ruido extremadamente molesto, en las mediciones más altas.

En el caso de la calle **Somera 8** puede ser la alta concentración de personas y en **Viuda de Epalza 3** y **Poza 53**, son debido al tráfico(claxon, ruidos de motor) o obras en la calle.

En los siguientes pasos vamos a ver la descripción de los datos y el nombre de las columnas para saber con que tipos de datos vamos a trabajar, así que visualizaremos con .dtypes(), .columns e info().

```
[ ] dfsonometros.dtypes
[ ] dfsonometros.info()
[ ] dfubicacion.columns
```

	Column	Non-Null Count	Dtype
0	nombre_dispositivo	168878	non-null object
1	decibelios	168878	non-null float64
2	address	168878	non-null object
3	status	168878	non-null int64
4	fecha	168878	non-null object
5	hora	168878	non-null object
6	ciudad	168878	non-null object
7	longitude	168878	non-null object
8	latitude	168878	non-null object

dtypes: float64(1), int64(1), object(7)
memory usage: 11.6+ MB

Podemos ver el **recuento total de registros** desde la fecha inicial hasta la final, y son un **total de 168878**, registros, puede haber una variación si se vuelven a recoger los datos.

Visualizamos con describe(), una tabla con los valores máximo y mínimos registrados, los percentiles, la media aritmética y la desviación standard, de los decibelios y el status de los sonómetros.

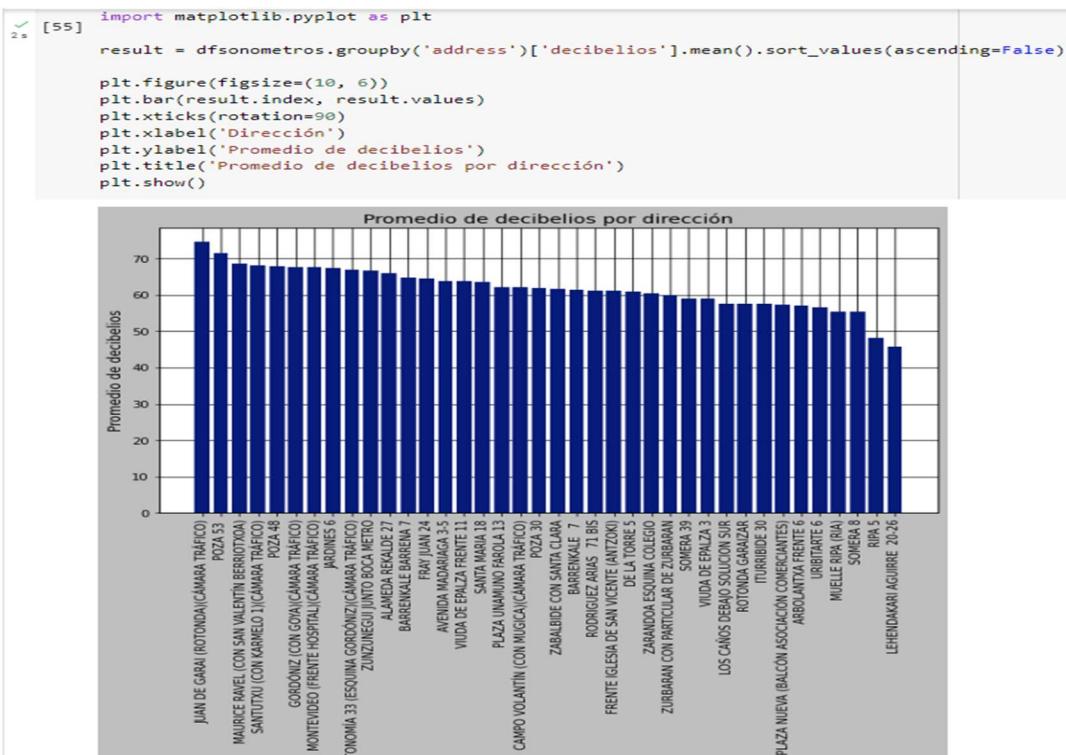
dfsonometros.describe()		
	decibelios	status
count	168878.000000	168878.000000
mean	59.732676	1.00106
std	10.631793	0.03254
min	22.400000	1.00000
25%	53.829250	1.00000
50%	60.700000	1.00000
75%	67.100000	1.00000
max	117.600000	2.00000

El paso siguiente es sacar los promedios y después una graficarlo.

```
result = dfsonometros.groupby('address')['decibelios'].mean().sort_values(ascending=False)
print(result)
```

address	decibelios
JUAN DE GARAI (ROTONDA)(CÁMARA TRÁFICO)	74.668539
POZA 53	71.598443
MAURICE RAVEL (CON SAN VALENTÍN BERRIOTXOA)	68.700855
SANTUTXU (CON KARMELO 1)(CÁMARA TRÁFICO)	68.095066
POZA 48	67.893588
GORDÓNIZ (CON GOYA)(CÁMARA TRÁFICO)	67.756653
MONTEVIDEO (FRENTE HOSPITAL)(CÁMARA TRÁFICO)	67.616795
JARDINES 6	67.324929
AUTONOMÍA 33 (ESQUINA GORDÓNIZ)(CÁMARA TRÁFICO)	66.860846
ZUNZUNEGUI JUNTO BOCA METRO	66.774696
ALAMEDA REKALDE 27	65.954069
BARRENKALE BARRENA 7	64.707358
FRAY JUAN 24	64.539956
AVENIDA MADARIAGA 3-5	63.907209
VIUDA DE EPALZA FRENTE 11	63.828937
SANTA MARÍA 18	63.491959
PLAZA UNAMUNO FAROLA 13	62.171258
CAMPO VOLANTÍN (CON MUGICA)(CÁMARA TRÁFICO)	62.068386
POZA 30	61.980763
ZABALBIDE CON SANTA CLARA	61.645962
BARRENKALE 7	61.356144
RODRIGUEZ ARIAS 71 BIS	61.060533
FRENTE IGLESIA DE SAN VICENTE (ANTZOKI)	61.051086
DE LA TORRE 5	60.833356
ZARANDOA ESQUINA COLEGIO	60.480782

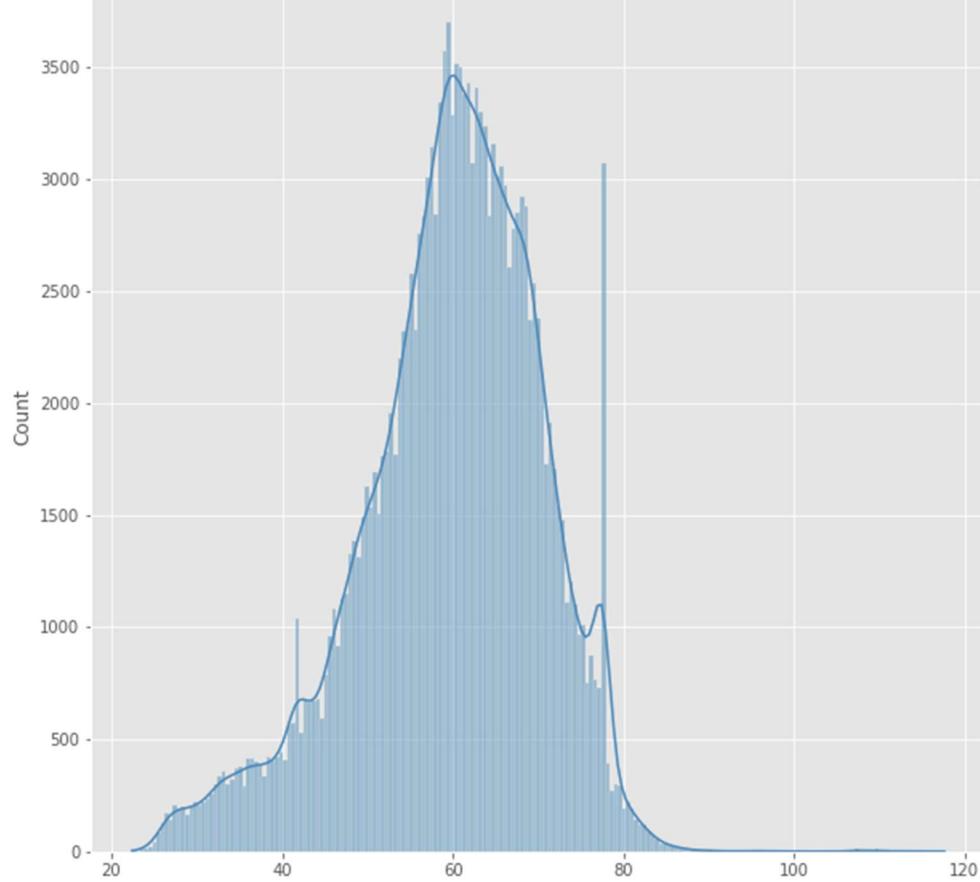
Los resultados obtenidos son iguales a la consulta hecha a MongoDB, en el apartado de Big Data, ahora mostraremos el resultado gráficamente.



La siguiente gráfica que vamos a mostrar es todas las veces que ha recogido un valor en el eje Y, y los decibelios recogidos en el eje X.

```
import matplotlib.pyplot as plt
import seaborn as sns
plt.figure(figsize=(10,10))
plt.style.use('ggplot')
sns.histplot(dfsonometros.decibelios, kde=True)
```

<matplotlib.axes._subplots.AxesSubplot at 0x7f38a3bdaa60>



Podemos ver la predominancia que hay en la gráfica en las muestras recogidas de 40 a 80 decibelios.

El siguiente paso es comprobar cuantas veces ha sido efectiva el valor de la muestra, contando los valores de status, que son las veces que funcionaba un sonómetro o no, siendo 1 muestra ok y 2 muestra no ok.

```
[ ] dfsonometros['status'].value_counts()
```

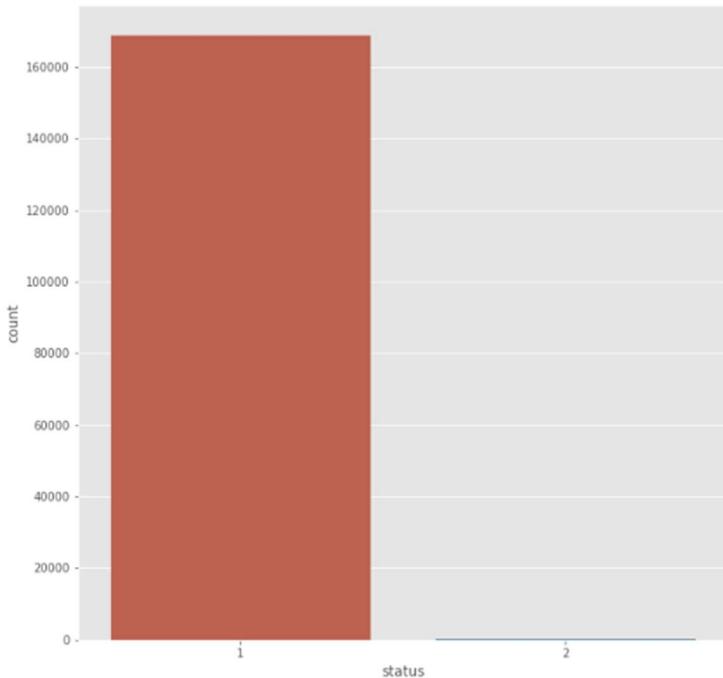
1	168699
2	179
Name: status, dtype: int64	

De las **168878** muestras podemos ver que OK estarán 168699 y NO OK 179, así que podremos trabajar con ellos con total integridad.

Representamos gráficamente el valor de los status:

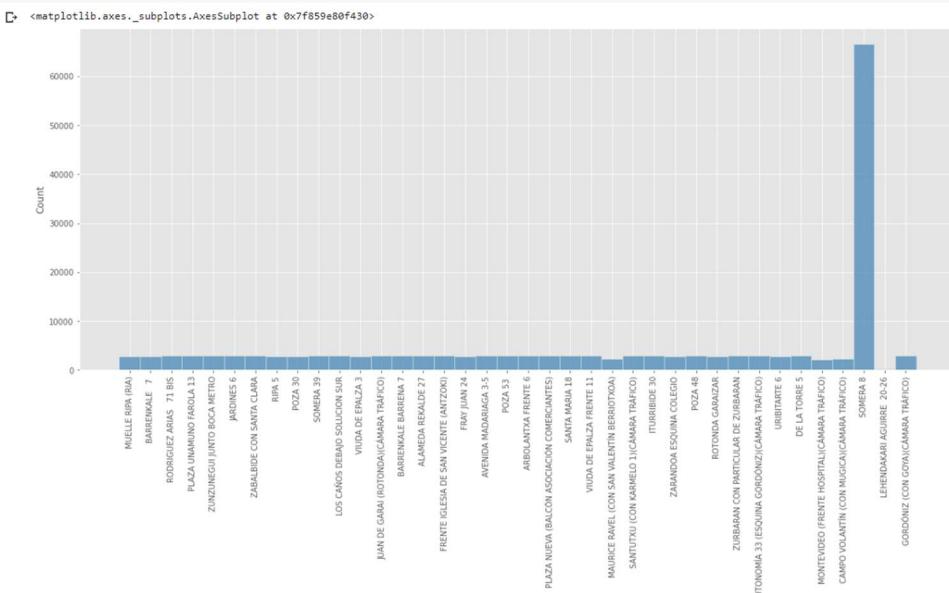
```
plt.figure(figsize=(10,10))
plt.style.use('ggplot')
sns.countplot(dfsonometros.status)

/usr/local/lib/python3.8/dist-packages/seaborn/_decorators.py:36: FutureWarning: Pass the foll
warnings.warn(
<matplotlib.axes._subplots.AxesSubplot at 0x7f38a6ec1580>
```



Ahora comprobamos por referencia del dispositivo, si recogen todos los sonómetros el mismo número de datos:

```
plt.figure(figsize=(80,80))
plt.style.use('ggplot')
sns.histplot(dfsonometros.nombre_dispositivo)
```

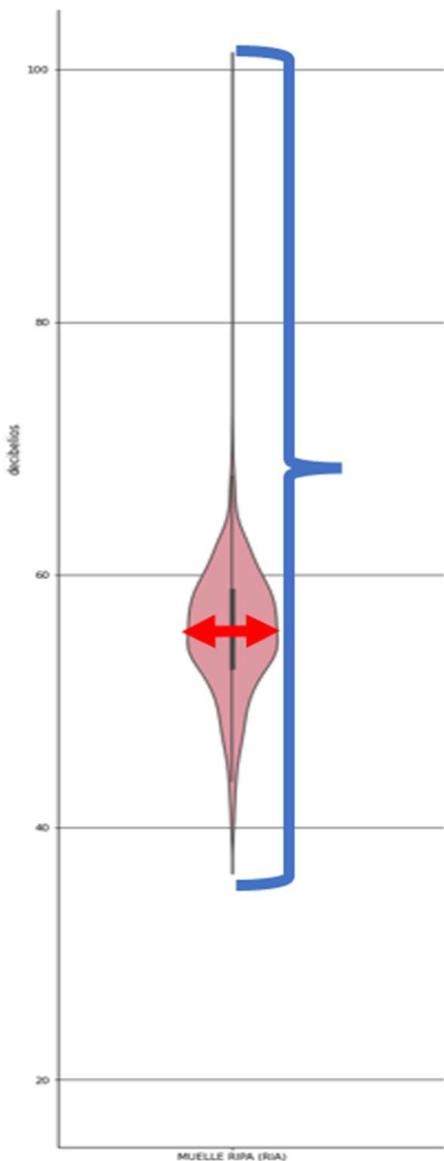


Se puede ver que el de **Somera 8** coge datos cada 5 minutos, en vez de los 15 minutos que recogen los demás y hay uno que solo coge **una muestra cada hora** que pertenece a **Lehendakari Agirre** en **Deusto**.

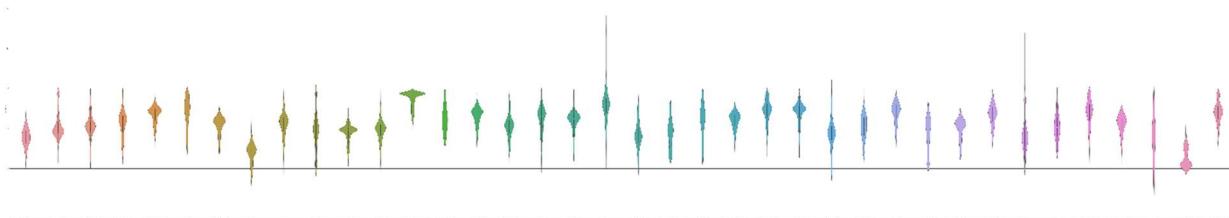
Los demás más o menos están a la par en recogida de datos.

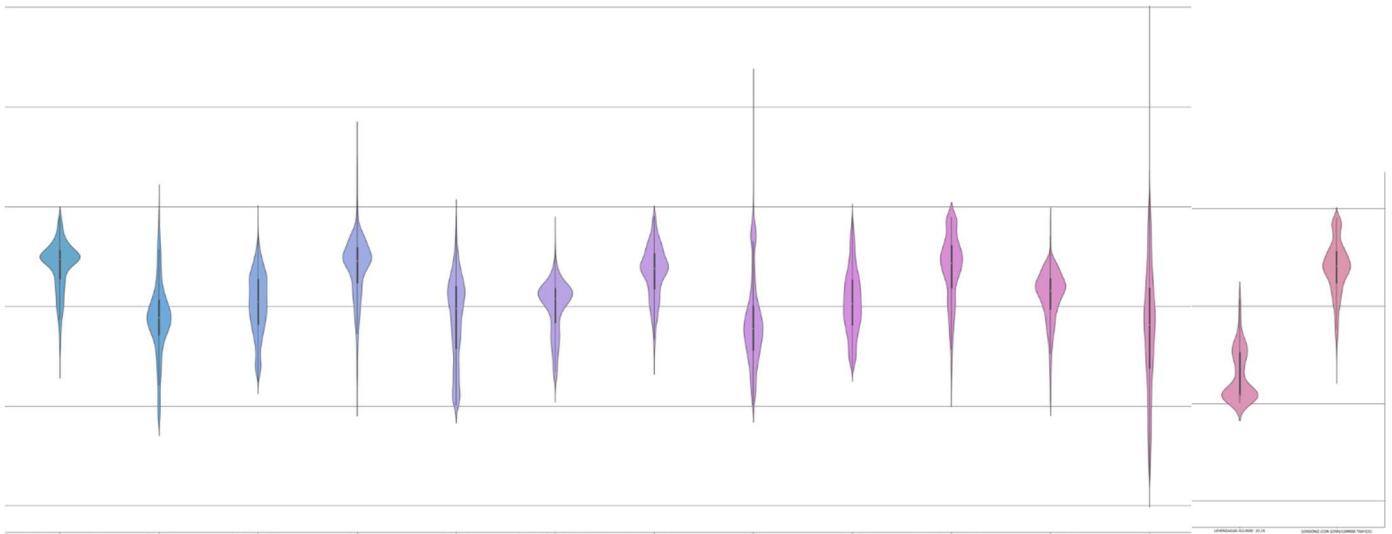
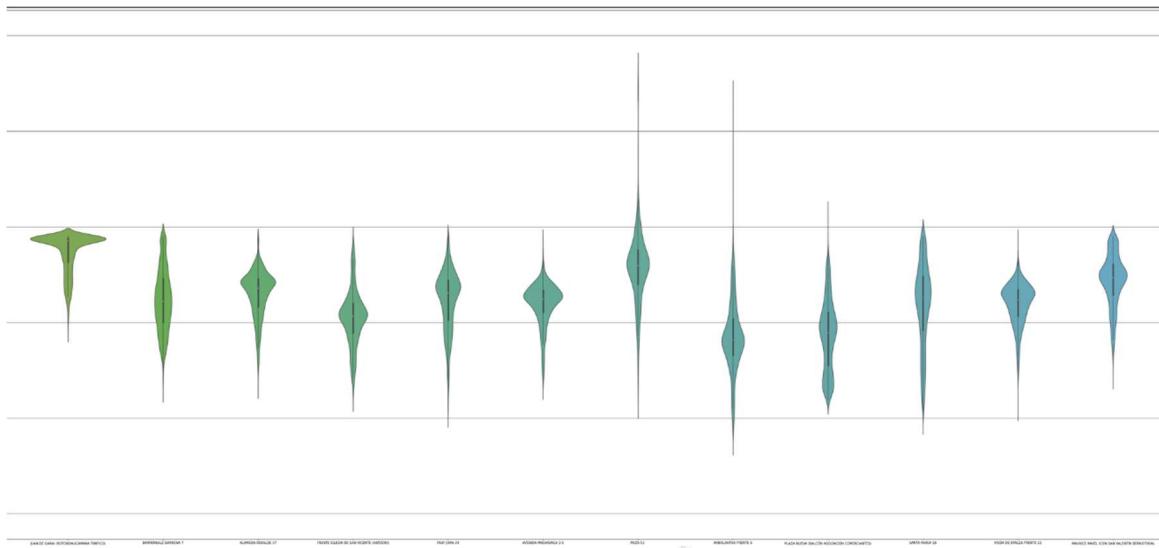
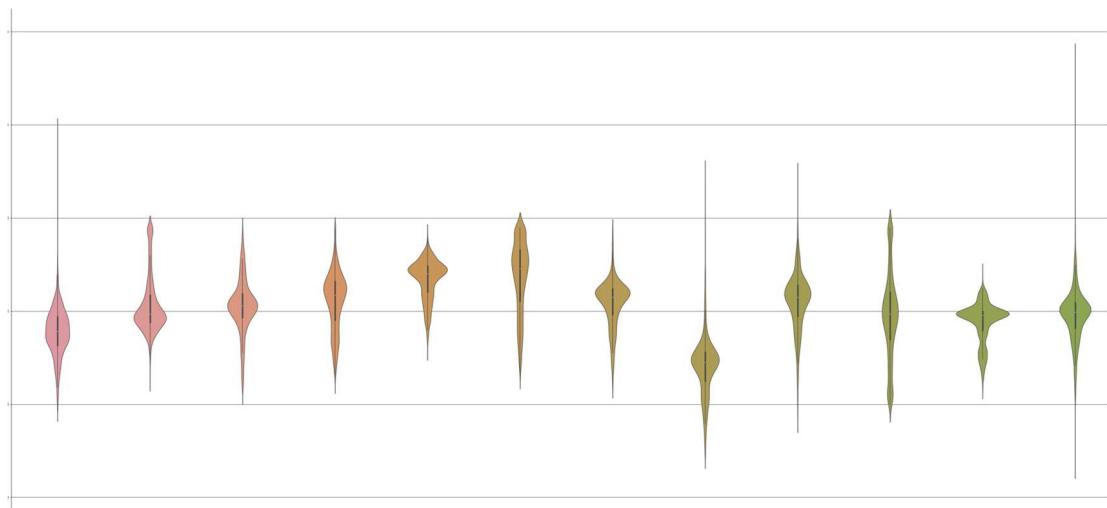
A continuación, vamos a verificar la gráfica utilizando la vista de violín, el violinplot es un tipo de gráfico que se utiliza en visualización de datos y que muestra la distribución de una variable numérica o cuantitativa a través de un diagrama de violín. Es una combinación de un histograma y un gráfico de densidad.

Como entender esta vista aplicada a nuestras muestras:



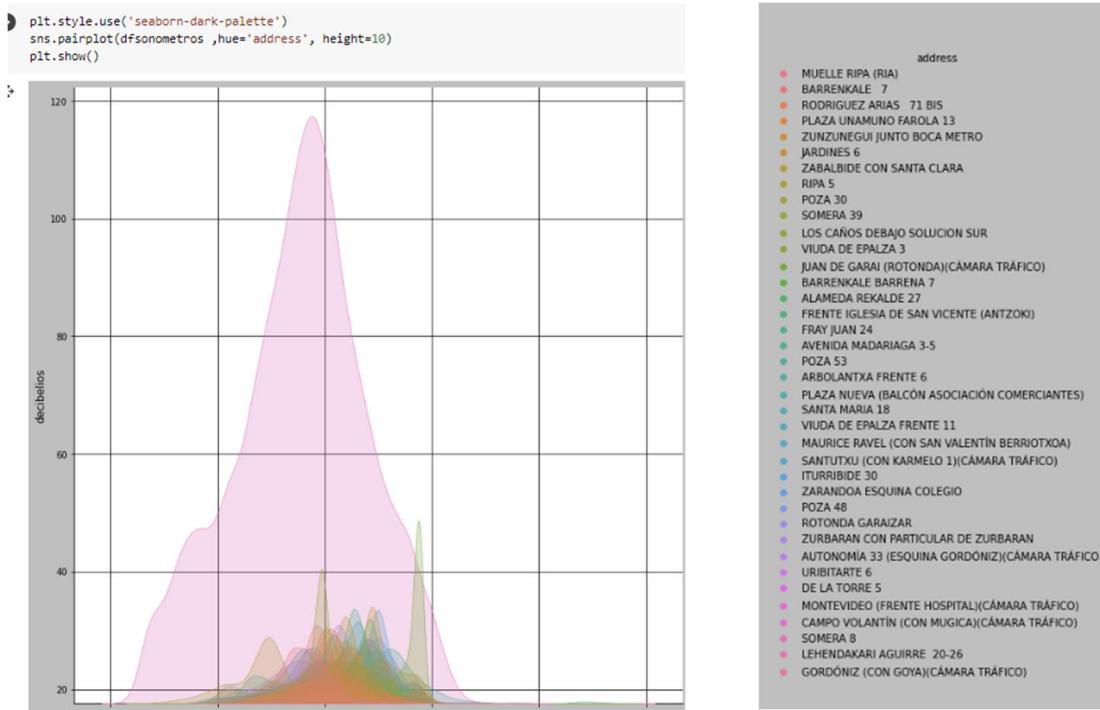
- En el eje Y estarán representados los decibelios.
- En el eje X estarán representadas las calles.
- La largora del violín representa el rango de decibelios que se han contado en esa calle.
- La anchura de la figura del violín es el número de veces que se ha contado por decibelio en esa cuenta.





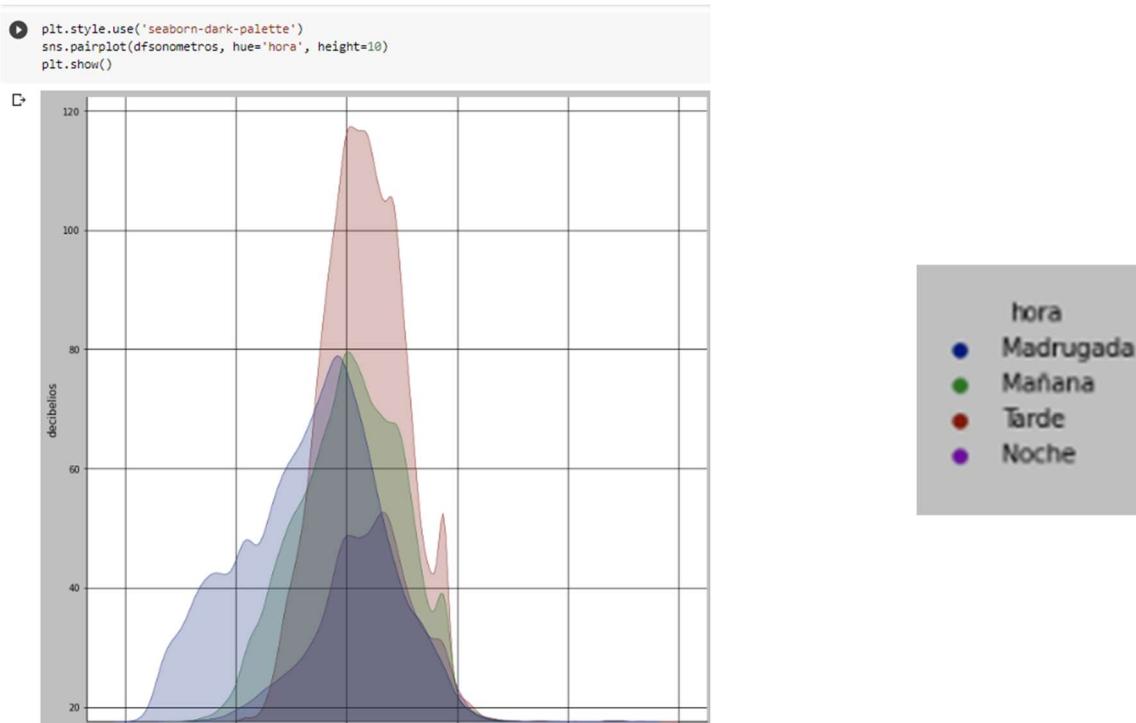
En las gráficas se puede ver que Juan de Garai, Poza 53 y Avenida de Montevideo suelen ser las que tienen los decibelios al alza.

Ahora representemos esto con una matriz de gráficos de dispersión (scatterplots) e histogramas utilizando el método pairplot de la biblioteca Seaborn.



En ella predomina el muestreo de Somera 8 porque tiene una gran cantidad de muestras, pero en el muestreo normal, baja la cantidad.

Ahora haremos lo mismo, pero por franja horaria:



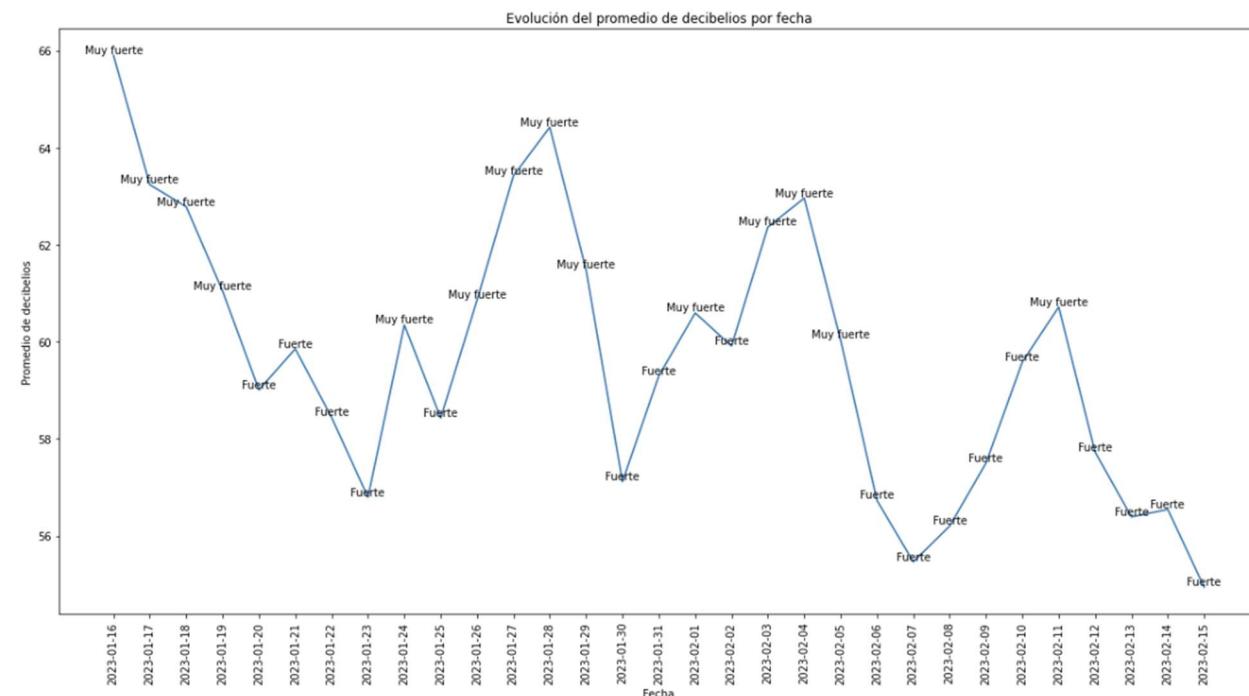
Predomina la tarde, siendo la más ruidosa, seguida de la mañana y la madrugada que están casi a la par.

Promedio real de todos los sonómetros de Bilbao.

Promedio de todos los sonómetros de Bilbao

```
▶ promedio = dfsonometros['decibelios'].mean()
categoria_promedio = categorize_sound(round(promedio, 2))
print(f"El promedio total de decibelios es {promedio:.2f}, que corresponde a la categoría de sonido '{categoria_promedio}'.")

El promedio total de decibelios es 59.73, que corresponde a la categoría de sonido 'Fuerte'.
```



c. Predicciones con Colab

La primera predicción que vamos a realizar es muy sencilla, de hecho, está documentada como curiosidad, hemos elegido el algoritmo del árbol de decisión para predecir si los sonómetros están funcionando o no en función de los valores de decibelios medidos por los mismos, y lo que se calcula es la precisión del modelo.

1. Árbol de decisiones

La precisión es del 0,99 sobre 1, esta relación la podemos hacer con la gráfica del status de los sonómetros que hemos plasmado en este documento anteriormente.

```
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

# Separar las características (X) y la etiqueta (y)
X = dfsonometros[['decibelios']]
y = dfsonometros['status']

# Dividir los datos en conjuntos de entrenamiento y prueba
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Crear el modelo de árbol de decisión
clf = DecisionTreeClassifier()

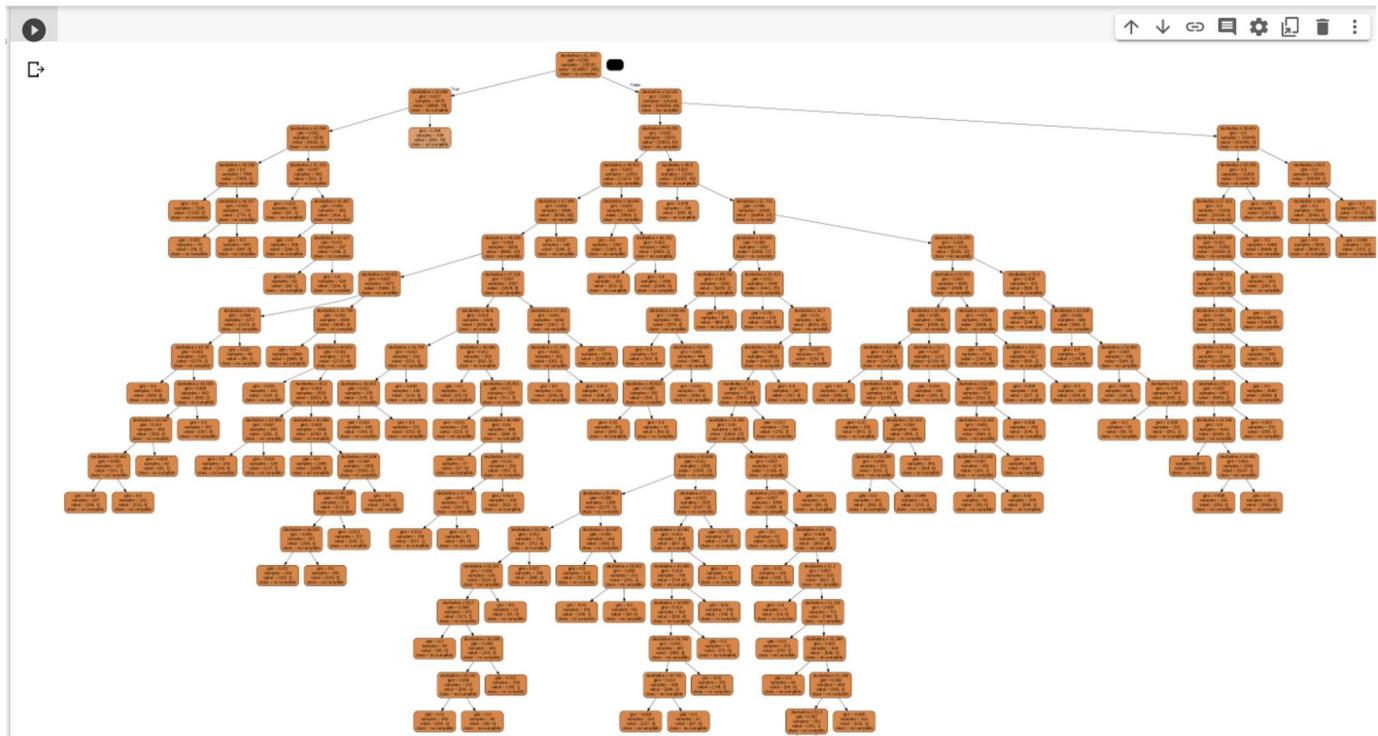
# Entrenar el modelo con los datos de entrenamiento
clf.fit(X_train, y_train)

# Realizar predicciones en los datos de prueba
y_pred = clf.predict(X_test)

# Calcular la precisión del modelo
accuracy = accuracy_score(y_test, y_pred)
print('Precisión:', accuracy)
```

Precisión: 0.9989933680720038

Como curiosidad dejaremos la gráfica de este árbol de decisiones:



2. Regresión lineal

La siguiente predicción vamos a utilizar una regresión lineal, y en este caso vamos a utilizarlo para predecir los niveles de decibelios en función de la ubicación geográfica (la longitud y latitud).

Para conseguirlo, vamos a hacer los siguientes pasos:

- Cargamos un conjunto de datos en un DataFrame y realizamos transformaciones de datos, como convertir una columna de cadenas a números y reemplazar comas por puntos en dos columnas.
- Graficamos los datos en un diagrama de dispersión, utilizando la longitud y latitud como coordenadas y la dirección de la dirección como un factor que se refleja en el color de los puntos.
- Dividimos los datos en un conjunto de entrenamiento y en un conjunto de prueba, luego entrenamos un modelo de regresión lineal utilizando la longitud y latitud como variables de entrada y los decibelios como la variable de salida.
- El código luego realiza una predicción de los decibelios utilizando los datos de prueba y calcula el promedio de las predicciones para cada dirección. Luego, imprime los promedios y los valores reales de los decibelios para cada dirección.

```
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression

# Carga los datos en un DataFrame
df2sonometros = dfsonometros.copy()

# Convierte la columna "decibelios" de cadenas a números
df2sonometros['decibelios'] = pd.to_numeric(df2sonometros['decibelios'])

# Reemplaza comas por puntos en las columnas "longitude" y "latitude"
df2sonometros['longitude'] = df2sonometros['longitude'].str.replace(',', '.').astype(float)
df2sonometros['latitude'] = df2sonometros['latitude'].str.replace(',', '.').astype(float)

# Añade la columna "Address" al DataFrame df2sonometros
df2sonometros['address'] = dfsonometros['address']

# Grafica los datos en un diagrama de dispersión
fig, ax = plt.subplots()
colors = df2sonometros['address'].astype('category').cat.codes
scatter = ax.scatter(df2sonometros['longitude'], df2sonometros['latitude'], c=colors, cmap='tab20', alpha=0.8)

# Elimina los nombres de las calles del gráfico
ax.set_xticklabels([])
ax.set_yticklabels([])
ax.set_xticks([])
ax.set_yticks([])

# Añade título al gráfico
ax.set_title('Decibelios en función de la ubicación')
plt.show()

# Dividimos los datos en conjunto de entrenamiento y conjunto de prueba
train_data = df2sonometros.sample(frac=0.8, random_state=1)
test_data = df2sonometros.drop(train_data.index)

# Entrenamos un modelo de regresión lineal utilizando la columna "decibelios" como
variable de entrada
# y las columnas "longitude" y "latitude" como variables de salida
model = LinearRegression()
model.fit(train_data[['longitude', 'latitude']], train_data['decibelios'])

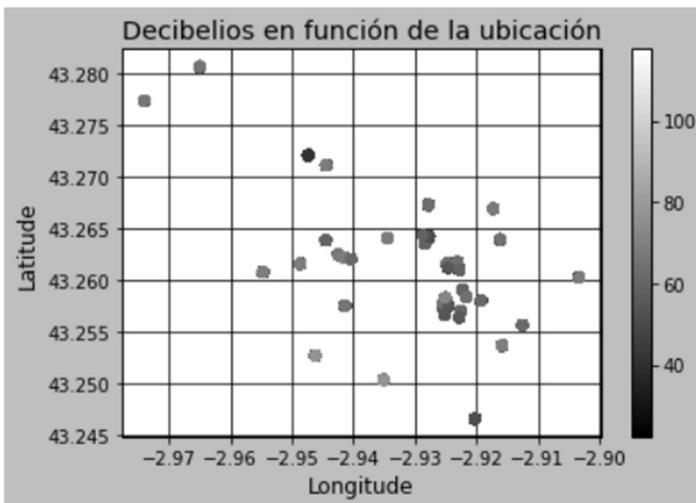
for address in test_data['address'].unique():
    filtered_data = test_data[test_data['address'] == address]
    # Calcular las predicciones para el conjunto de prueba
    predictions = model.predict(filtered_data[['longitude', 'latitude']])
    # Calcular el promedio de las predicciones para cada dirección
```

```

average = predictions.mean()
# Mostrar el resultado promediado
print(f"Promedio de las predicciones para {address}: {average:.2f}")
print(f"Valores reales para {address}:")
print(filtered_data['decibelios'].head())
print()

```

Salida del programa:



```

Promedio de las predicciones para MUELLE RIPA (RIA): 58.67
Valores reales para MUELLE RIPA (RIA):
0      51.599
130    55.143
190    52.352
475    55.321
590    54.003
Name: decibelios, dtype: float64

Promedio de las predicciones para JARDINES 6: 59.20
Valores reales para JARDINES 6:
5      62.623
425   58.031
593   64.161
995   61.585
1284  67.785
Name: decibelios, dtype: float64

Promedio de las predicciones para RIPA 5: 58.65
Valores reales para RIPA 5:
7      42.6
42     44.4
443   47.8
1611  48.3
1669  51.2
Name: decibelios, dtype: float64

```

Salen todas las calles promediadas, en la figura solo he puesto 3 pero salen el total de calles.

Ahora ordenamos el promedio de decibelios por calle, y el resultado nos da una predicción de cuales serán las calles más ruidosas en función de los datos.

```

promedios = []

for address in test_data['address'].unique():
    filtered_data = test_data[test_data['address'] == address]
    # Agrupar los datos por dirección y hora
    grouped_data = filtered_data.groupby(['address', 'hora'])
    # Calcular las predicciones para cada grupo de datos
    predictions = grouped_data.apply(lambda x: model.predict(x[['longitude', 'latitude']]).mean())
    # Obtener la hora en la que se producen más decibelios por calle
    max_hour = predictions.idxmax()[1]
    # Obtener el promedio de las predicciones para cada dirección
    average = predictions.mean()
    # Agregar la dirección, el promedio y la hora en la que se producen más decibelios a la lista de promedios
    promedios.append((address, average, max_hour))

# Ordenar los promedios de mayor a menor

```

```
promedios_ordenados = sorted(promedios, key=lambda x: x[1], reverse=True)

# Mostrar los resultados
for direccion, promedio, hora in promedios_ordenados:
    print("Dirección: ", direccion)
    print("Hora en la que se producen más decibelios: ", hora)
    print("Promedio de las predicciones: ", promedio)
    print()
```

Salida del programa:

Dirección: FRAY JUAN 24
Hora en la que se producen más decibelios: Madrugada
Promedio de las predicciones: 68.50061666659713

Dirección: MONTEVIDEO (FRENTE HOSPITAL) (CÁMARA TRÁFICO)
Hora en la que se producen más decibelios: Madrugada
Promedio de las predicciones: 66.12491039686483

Dirección: ZARANDOA ESQUINA COLEGIO
Hora en la que se producen más decibelios: Mañana
Promedio de las predicciones: 65.7951990307838

Dirección: GORDÓNIZ (CON GOYA) (CÁMARA TRÁFICO)
Hora en la que se producen más decibelios: Madrugada
Promedio de las predicciones: 65.19105959761328

Dirección: ZUNZUNEGUI JUNTO BOCA METRO
Hora en la que se producen más decibelios: Madrugada
Promedio de las predicciones: 64.51624326713136

Dirección: AUTONOMÍA 33 (ESQUINA GORDÓNIZ) (CÁMARA TRÁFICO)
Hora en la que se producen más decibelios: Madrugada
Promedio de las predicciones: 63.3072915884386

Dirección: RODRIGUEZ ARIAS 71 BIS
Hora en la que se producen más decibelios: Madrugada
Promedio de las predicciones: 63.16737797217138

Dirección: POZA 48
Hora en la que se producen más decibelios: Madrugada
Promedio de las predicciones: 62.86616625571696

Dirección: JUAN DE GARAI (ROTONDA) (CÁMARA TRÁFICO)
Hora en la que se producen más decibelios: Madrugada
Promedio de las predicciones: 62.797016665457704

Dirección: POZA 53
Hora en la que se producen más decibelios: Madrugada
Promedio de las predicciones: 62.70419620016855

d. Red neuronal

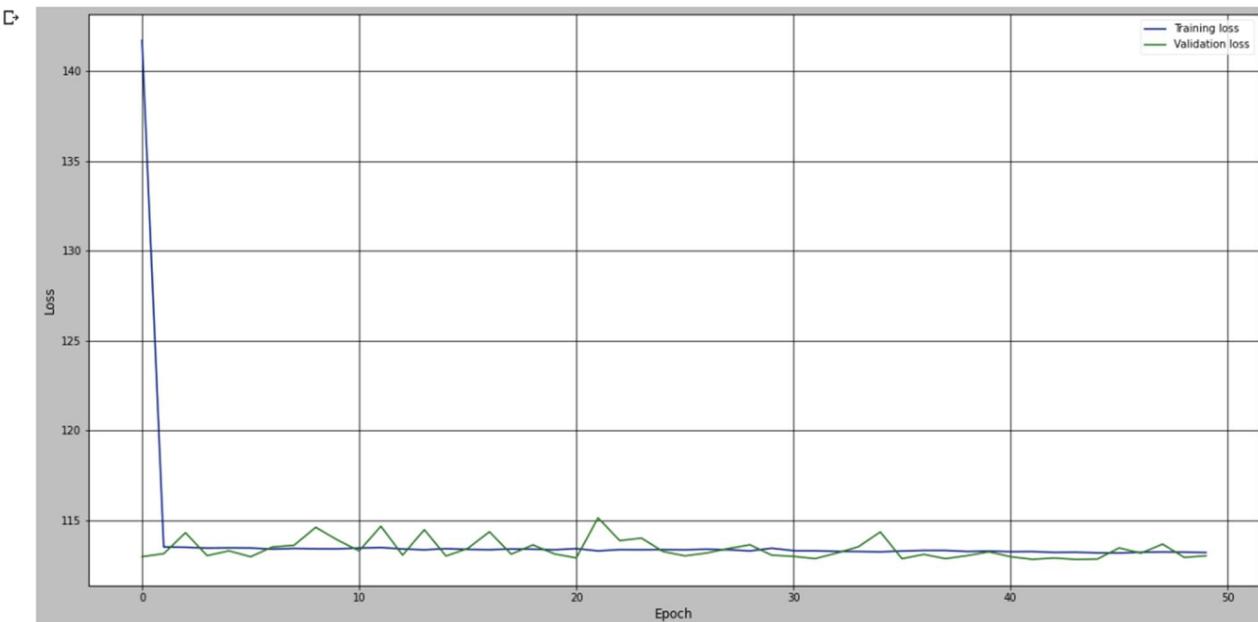
La red neuronal tiene como objetivo predecir el nivel de decibelios en función de la ubicación geográfica. Específicamente, se utiliza la columna "decibelios" como variable de entrada y las columnas "longitud" y "latitud" como variables de salida.

La función de pérdida que se utiliza para entrenar la red neuronal es el error cuadrático medio (MSE), que mide la diferencia entre las predicciones del modelo y los valores reales de los datos. El objetivo del entrenamiento es minimizar la pérdida, para realizar futuras predicciones con datos nuevos.

```
Epoch 38/50
3378/3378 [=====] - 9s 3ms/step - loss: 113.2583 - val_loss: 112.8739
Epoch 39/50
3378/3378 [=====] - 10s 3ms/step - loss: 113.2929 - val_loss: 113.1268
Epoch 40/50
3378/3378 [=====] - 8s 2ms/step - loss: 113.2095 - val_loss: 112.8341
Epoch 41/50
3378/3378 [=====] - 10s 3ms/step - loss: 113.2588 - val_loss: 112.8318
Epoch 42/50
3378/3378 [=====] - 11s 3ms/step - loss: 113.2451 - val_loss: 112.8342
Epoch 43/50
3378/3378 [=====] - 10s 3ms/step - loss: 113.2217 - val_loss: 112.8625
Epoch 44/50
3378/3378 [=====] - 8s 2ms/step - loss: 113.2472 - val_loss: 113.2578
Epoch 45/50
3378/3378 [=====] - 10s 3ms/step - loss: 113.2041 - val_loss: 112.8732
Epoch 46/50
3378/3378 [=====] - 10s 3ms/step - loss: 113.2753 - val_loss: 112.8124
Epoch 47/50
3378/3378 [=====] - 8s 2ms/step - loss: 113.1882 - val_loss: 112.9959
Epoch 48/50
3378/3378 [=====] - 11s 3ms/step - loss: 113.1571 - val_loss: 112.9688
Epoch 49/50
3378/3378 [=====] - 10s 3ms/step - loss: 113.1839 - val_loss: 113.0129
Epoch 50/50
3378/3378 [=====] - 9s 3ms/step - loss: 113.2052 - val_loss: 113.2521
1056/1056 [=====] - 2s 2ms/step - loss: 112.8660
Pérdida en el conjunto de prueba: 112.87
```

En este caso, el valor de la pérdida obtenido en el conjunto de entrenamiento es 113.2052, mientras que en el conjunto de prueba es 112.8660. Estos valores indican que la red neuronal está logrando ajustarse bien a los datos de entrenamiento y a los datos de prueba.

Gráfica del entrenamiento de la red neuronal:



e. BigML

BigML es una plataforma de aprendizaje automático en la nube que permite a los usuarios construir y desplegar modelos de aprendizaje automático de manera rápida y sencilla. La plataforma se enfoca en hacer que el aprendizaje automático sea accesible para cualquier persona, independientemente de su nivel de habilidad o experiencia en programación o estadísticas.

BigML ofrece una interfaz intuitiva basada en la nube que facilita la carga de datos, la selección de variables y la construcción de modelos de aprendizaje automático.

Para empezar en BigML vamos a cargar el archivo CSV en el dashboard, hemos seleccionado el archivo CSV debido a su poco peso, unos 14 megas, ya que en formato Json pesa aproximadamente 53 megas.

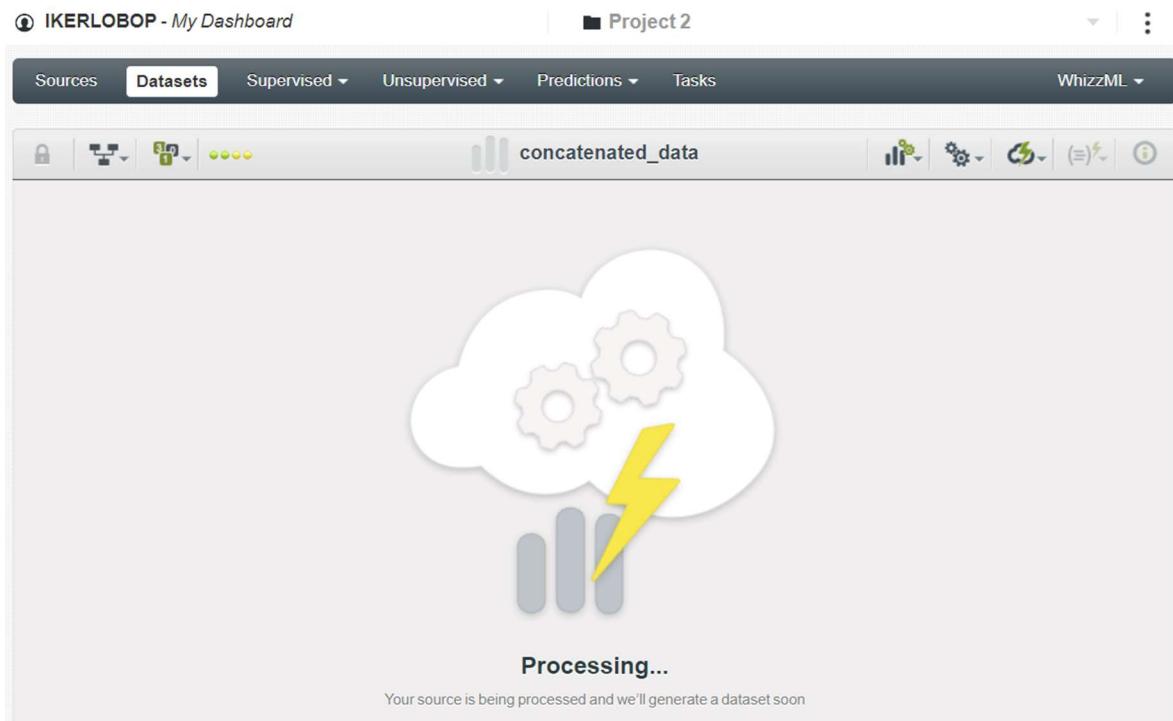
The screenshot shows the BigML dashboard interface. At the top, there's a navigation bar with links for PRODUCT, GETTING STARTED, PRICING, SUPPORT, IKERLOBOP, and Dashboard. Below the navigation is a header bar with tabs for Sources, Datasets, Supervised, Unsupervised, Predictions, Tasks, and WhizzML. The main area is titled 'Sources' and shows a list of files. One file, 'concatenated_data.csv', is highlighted. A context menu is open over this file, with options: UPLOAD A LOCALFILE (CSV, TSV, JSON, ARFF, XLS, GZ, BZ2, PNG, JPG, GIF, TIF, BMP, WEBP AND MORE), CREATE A SOURCE FROM A URL, and CREATE AN INLINE SOURCE.

Al cargar los datos podemos ver las variables, con sus respectivos campos:

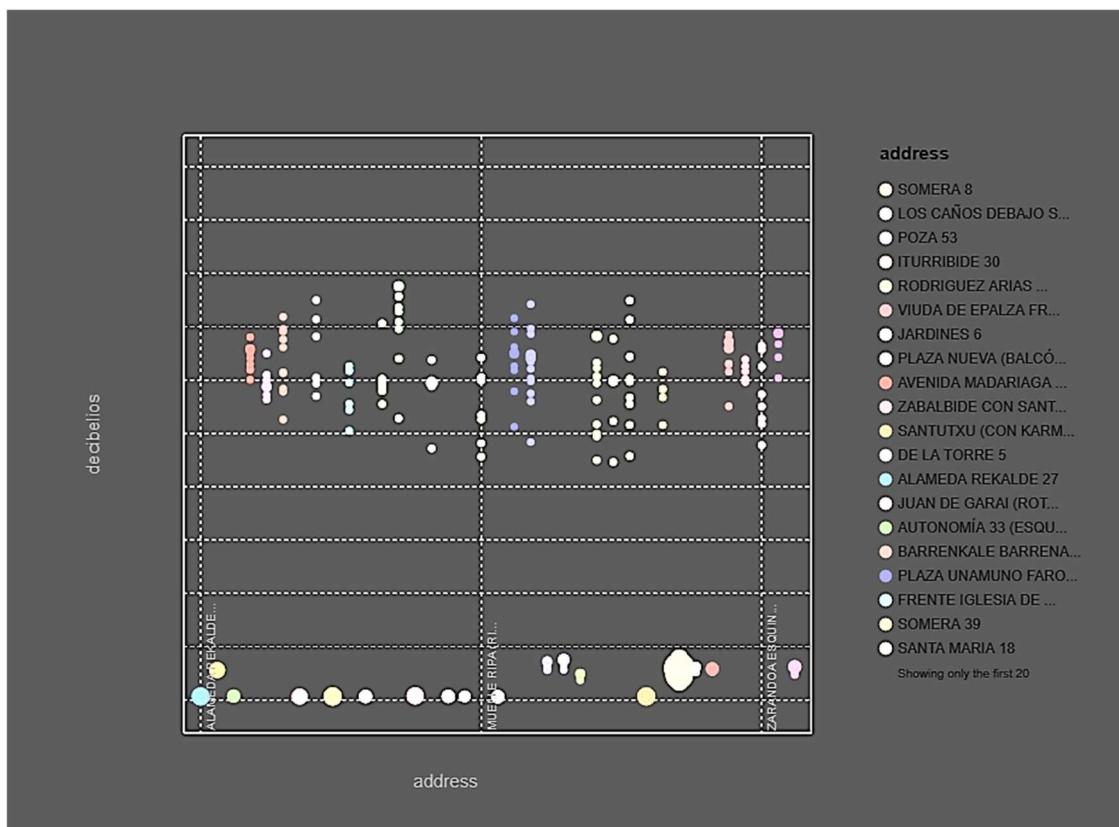
The screenshot shows the BigML dashboard interface with the 'concatenated_data.csv' dataset selected. The top navigation bar and tabs are identical to the previous screenshot. The main area displays the dataset details. The dataset name is 'concatenated_data.csv'. Below it is a table with columns: Name, Type, Instance 1, Instance 2, and Instance 3. The table rows represent different variables and their values across three instances. The variables and their values are:

Name	Type	Instance 1	Instance 2	Instance 3
nombre_dispositivo	ABC	BI-RUI-C003	BI-RUI-006	BI-RUI-001
decibelios	123	51.599	55.801	71.792
address	ABC	MUELLE RIPA (RIA)	BARRENKALE 7	RODRIGUEZARIAS 71
status	123	1	1	1
fecha	YYYY-MM-DD	2023-01-16	2023-01-16	2023-01-16
hora	ABC	Madrugada	Madrugada	Madrugada
ciudad	ABC	Bilbao	Bilbao	Bilbao
longitude	123	-2,924641	-2,925136	-2,944465
latitude	123	43,261251	43,256697	43,263875
fecha.year	YYYY-MM-DD	2023	2023	2023

Entonces cargamos todo y creamos un nuevo dataset:



Una vez cargado, en esta imagen podemos ver como ya nos segmenta los datos y nos hace unos gráficos con los campos de decibelios y direcciones.



A partir de aquí vamos a aplicar 2 algoritmos, uno supervisado y otro no supervisado.

- Algoritmo supervisado

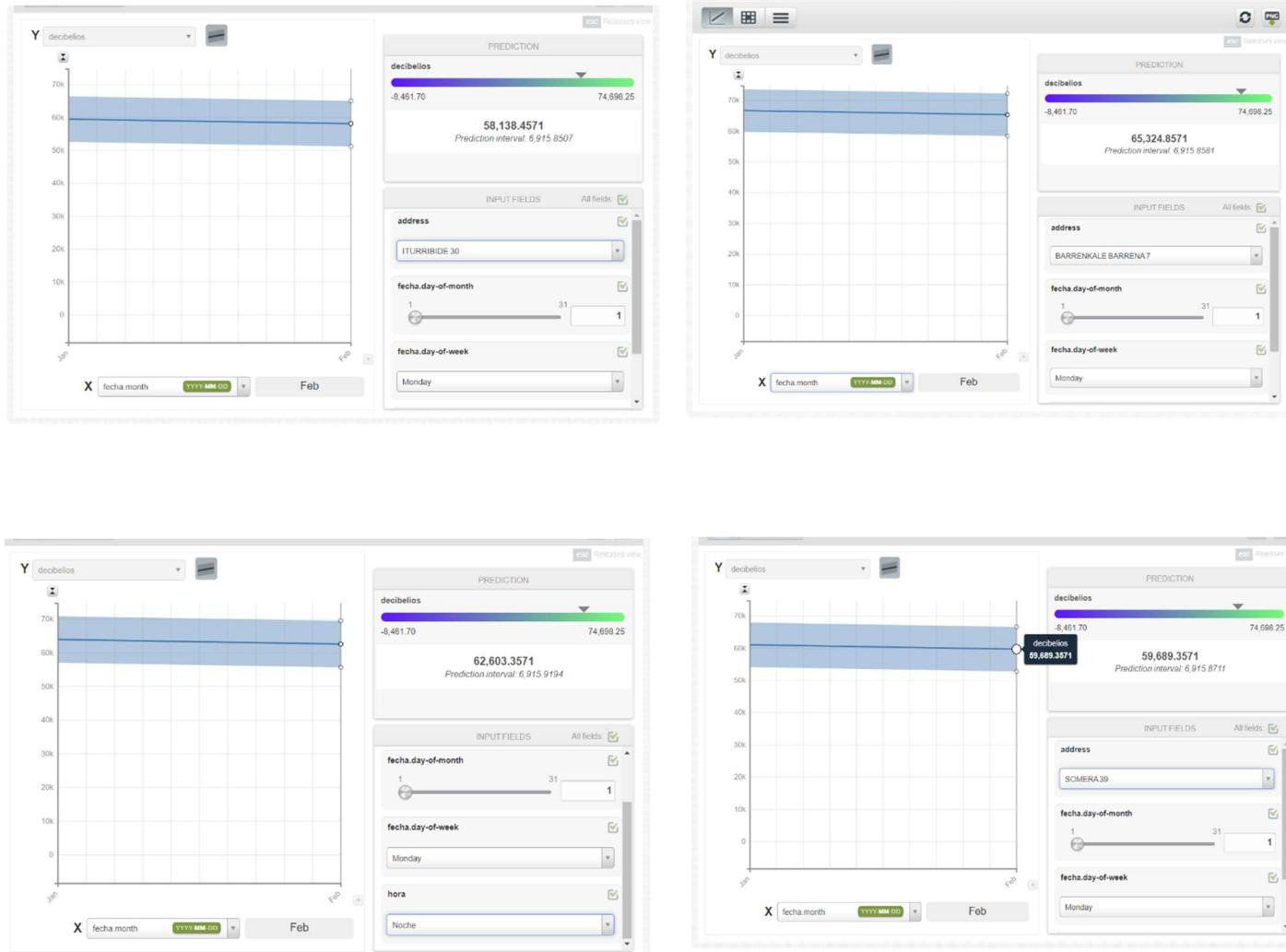
Un algoritmo de aprendizaje supervisado utiliza datos de entrenamiento etiquetados para predecir la salida de nuevas observaciones.

Utilizamos la regresión lineal para analizar la relación entre estos dos conjuntos de datos.

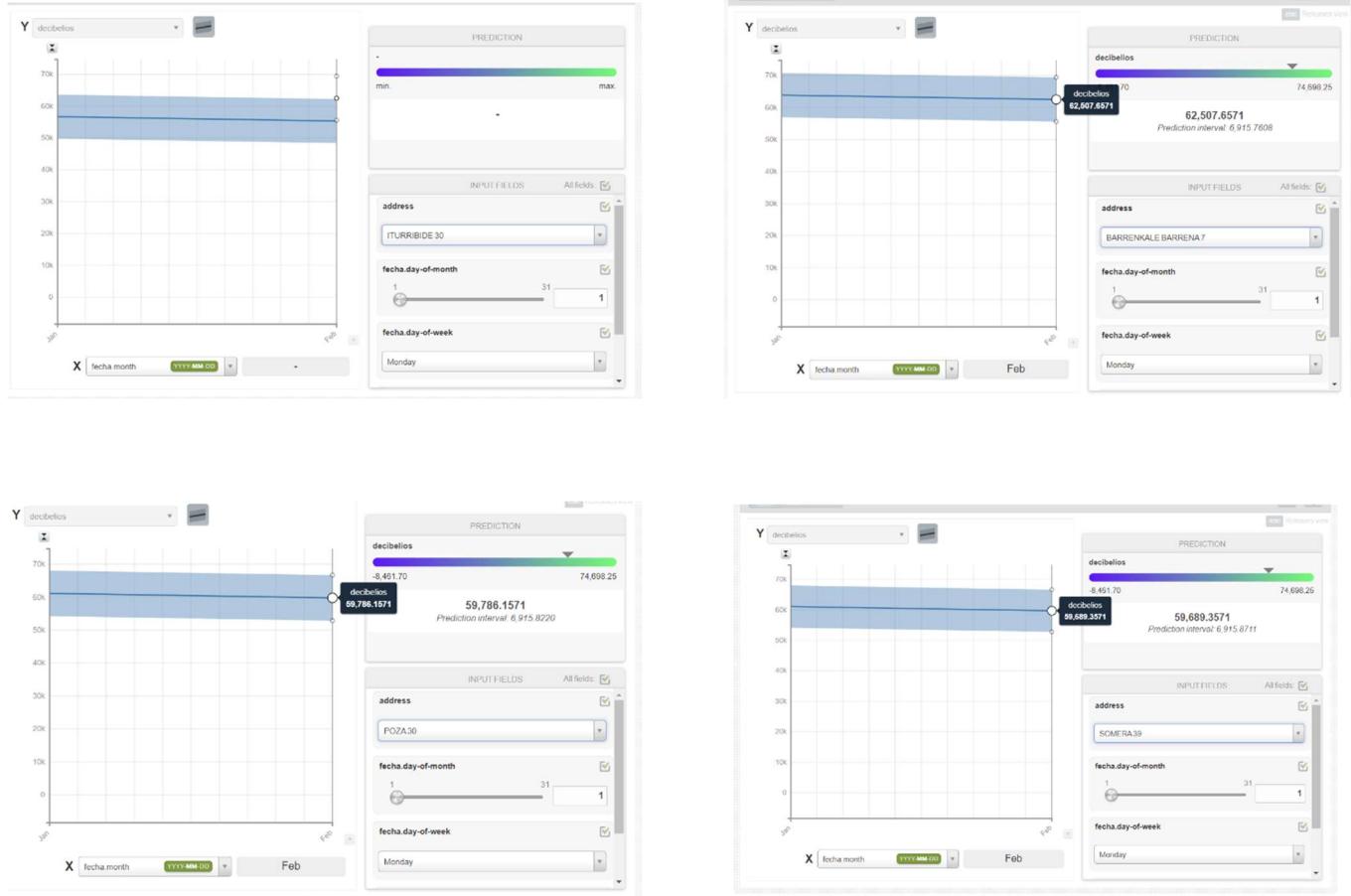
En este caso analizamos las calles principales de Ocio donde hay un sonómetro cerca, las franjas horarias se sitúan en la noche y en la madrugada.

- Iturribide
- Barrenkale Barrena
- Poza
- Somera

Noche:



Madrugada:

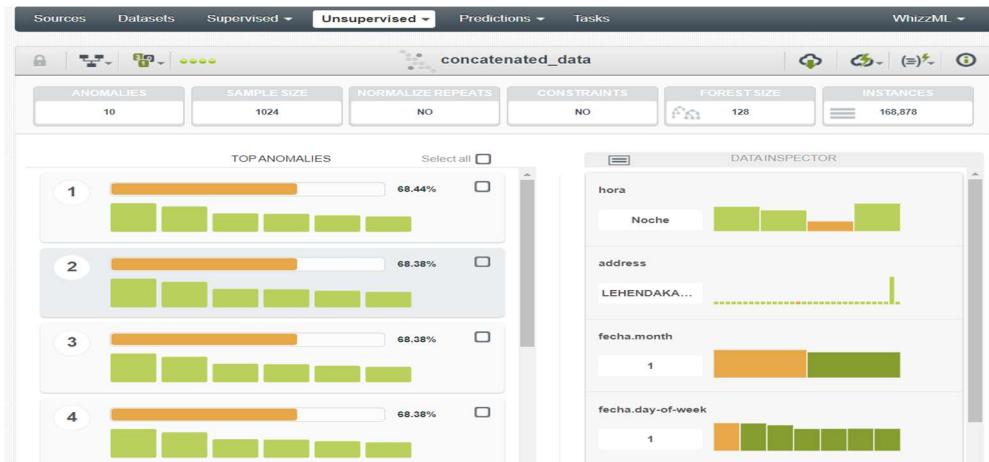


En todos los gráficos la línea de regresión resultante tiene una pendiente negativa, lo que indicaría que los niveles de ruido están disminuyendo a medida que pasa el tiempo, por lo cual la tendencia es a la baja, aunque el ruido es molesto.

- Algoritmo No Supervisado

El algoritmo no supervisado busca patrones y estructuras en datos no etiquetados sin una salida predefinida.

Hemos utilizado el buscador de anomalías:



En las anomalías hemos detectado que:

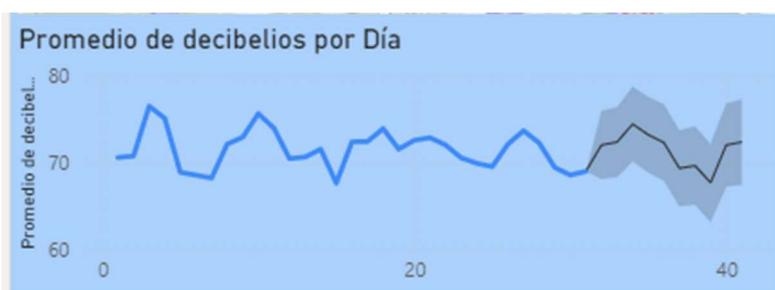
- El menor número de decibelios se da por la noche.
- Lehendakari Agirre registra el menor número de muestras recogidas, mientras Somera 8 el mayor.
- En enero se han recogido más muestras que en febrero.
- El día de la semana donde se han recogido más decibelios es el lunes.

f. Power BI

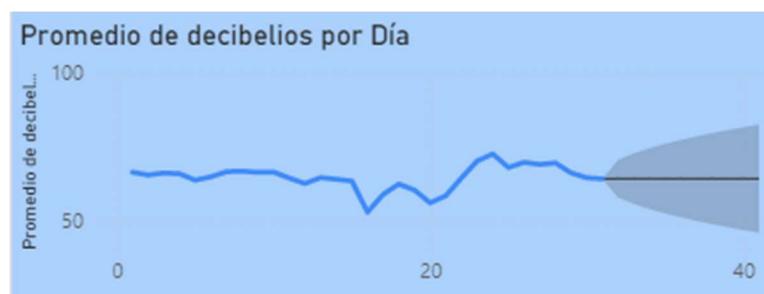
Predicción por calle en Power BI.

La predicción en una gráfica se refiere a la capacidad de predecir valores futuros basados en patrones identificados en los datos históricos, vamos a tomar de ejemplo una calle de ocio, otra que es una arteria principal de tráfico y la general.

Predicción de Poza 43 para los siguientes 9 días:



Predicción de Fray Juan 24 para los siguientes 9 días:



Promedio de todo Bilbao para los siguientes 9 días:



4. Conclusión general, problemas y Scrum

Conclusiones

Las calles que mayoritariamente tienen mayor número de decibelios son las que tienen mayor tráfico en el mapa, con el promedio sabemos que la mayoría de los ruidos son de tarde.

La predicción nos dice que la calle más ruidosa es Fray Juan 24 en Zorroza y las siguientes para tener en cuenta son de la misma artería principal de tráfico en Bilbao que son Zabalburu, Autonomía, Avenida Montevideo, etc.

Poza es una calle que tiene ruido sobre todo de tarde, en esa misma calle hay zonas de carga y descarga y mucho movimiento.

Las zonas de ocio tienen una tendencia a la baja en ruido, aunque el promedio de decibelios en escala sigue siendo molesto.

Los ruidos de madrugada en arterias principales registrados no pueden ser de ocio nocturno, ya que alcanzan una sonoridad muy fuerte incluso molesta, más bien se deben al camión de la basura o las barredoras.

A pesar de que Somera es una calle en la que los sonómetros cogen más muestras de lo normal, en los promedios de las muestras recogidas y de la predicción, no esta entre las más ruidosas.

Problemas

Uno de los principales problemas era la elección de los datos, al principio se eligió OpenSkyNetwork pero el volumen de datos era bastante escaso, ya que solo se podía acceder a los últimos 7 días, para por ejemplo hacer seguimientos de vuelos había que pagar un extra.

La máquina virtual ha tenido varios problemas y se optó por hacerlo con Windows 10.

Otro problema ha sido la conversión de datos y su concatenación, ya que la peculiar estructura del geojson no puede trabajar en cualquier programa, por ejemplo, se hace muy complicada en Power BI. La estrategia por abordar ha sido modificar un CSV, cambiando los puntos de los decimales por comas, y hacer la separación de todos los campos por punto y coma.

El último problema surge en la elección de los datos de la red neuronal, ya que había que transformar datos para poder trabajarlos.

Scrum(TRELLO)

Scrum es una metodología ágil de gestión de proyectos que se enfoca en la entrega temprana de productos funcionales y en la colaboración cercana entre los miembros del equipo.

En Scrum, el proyecto se divide en ciclos de desarrollo de corta duración llamados sprints, en los que el equipo trabaja para producir un entregable completo y funcional.

Cada sprint comienza con una reunión de planificación en la que se establecen los objetivos y se seleccionan las tareas prioritarias para el sprint. Durante el sprint, el equipo realiza reuniones diarias para monitorear el progreso y hacer ajustes según sea necesario. Al final del sprint, se realiza una revisión y una retrospectiva para evaluar el progreso y planificar el siguiente sprint.

El monitoreo se ha realizado de manera semanal, con Ainhoa e Iban, intentando cumplir los objetivos semanales, las primeras semanas han sido un poco lentas por la elección de los datos.

Aquí un ejemplo de seguimiento de las primeras semanas del reto, realizado en Trello:

Lista de tareas	En proceso	Hecho
Proyecto IABD	Iniciar una sesión en IntelliJ con SparkSession instalando dependencias	AEVIT OpenData
Crear Power Bi	probar apis y crear archivos JSON para abrir en Spark	image.png
+ Añada una tarjeta	Recopilar datos de dos fuentes mediante las APIS	Se ha seleccionado las fuentes y apis de datos correspondientes
	limpieza de datos	Fallos en máquina virtual
	+ Añada una tarjeta	 image.png
		Empezando de nuevo en Windows xDDD
		Optimismo y alegría :D
		Cambio de proyecto a Sonometros para contaminación acústica.
		+ Añada una tarjeta

5. Webgrafía

<https://opendata.euskadi.eus/>

https://www.w3schools.com/python/module_requests.asp

https://www.w3schools.com/python/python_json.asp

<https://pypi.org/project/pymongo/>

<https://www.freecodecamp.org/espanol/news/como-usar-pip-install-en-python/>

<https://colab.research.google.com/>

<https://es.stackoverflow.com/questions/128897/hacer-un-json-de-dos-directorios-en-python>

<https://phoenixnap.com/kb/install-spark-on-windows-10>

<https://openwebinars.net/blog/como-utilizar-spark-en-windows/>

<http://blog.josemarianoalvarez.com/2018/09/09/instalar-apache-spark-en-windows-10/>

<https://bigml.com/>

6. Anexo

a. Código Spark-Scala

```
import java.net.URL
import javax.net.ssl.HttpsURLConnection
import javax.net.ssl.SSLContext
import javax.net.ssl.TrustManager
import javax.net.ssl.X509TrustManager
import java.security.cert.X509Certificate

// Clase para deshabilitar la verificación del certificado SSL
class TrustAllX509TrustManager extends X509TrustManager {
    def getAcceptedIssuers(): Array[X509Certificate] = null
    def checkClientTrusted(certs: Array[X509Certificate], authType: String): Unit = {}
    def checkServerTrusted(certs: Array[X509Certificate], authType: String): Unit = {}
}

// Deshabilitar temporalmente la verificación del certificado SSL
val trustAllCerts = Array[TrustManager](new TrustAllX509TrustManager())
val sslContext = SSLContext.getInstance("SSL")
sslContext.init(null, trustAllCerts, new java.security.SecureRandom())
HttpsURLConnection.setDefaultSSLSocketFactory(sslContext.getSocketFactory())

// Conectar a la URL
val url = new
URL("https://www.bilbao.eus/aytoonline/jsp/opendata/movilidad/od_sonometro_mediciones.j
sp?idioma=c&formato=json")
val connection = url.openConnection().asInstanceOf[HttpsURLConnection]
val response = scala.io.Source.fromInputStream(connection.getInputStream).mkString
println(response)

// Reestablecer la verificación del certificado SSL
sslContext.init(null, null, null)
HttpsURLConnection.setDefaultSSLSocketFactory(sslContext.getSocketFactory())

//Guardamos el resultado en un fichero
import java.io._
val pw = new PrintWriter(new
File("C:\\\\Users\\\\Iker\\\\Desktop\\\\retoApi\\\\sonometro3.json"))
pw.write(response)
pw.close
```

b. Código Python

```
import requests
import json

# archivo JSON sonometros bilbao
with open("C://Users/Iker/Desktop/retoApi/sonometro3.json") as f:
    contenido = json.load(f)
# URL GEOJSON sonometros bilbao
url_geo =
"https://www.bilbao.eus/aytoonline/jsp/opendata/movilidad/od_sonometro_ubicacion.jsp?id
ioma=c&formato=geojson"

# URL GEOJSON trafico bilbao
url_geo_trafico = "https://www.bilbao.eus/aytoonline/srvDatasetTrafico?formato=geojson"

#verificamos si es correcta la conexion es correcta a Json
try:
    #conectar a carpetalocal
    respuesta= requests.Response()
    respuesta._content = json.dumps(contenido).encode('utf-8')
    respuesta.status_code = 200
    respuesta.headers[ 'content-type' ] = 'application/json'

    print("*****")
    print("*Conectado a datos JSON LOCALES*")
    print("*****")

    from tqdm import tqdm
    import time
    for i in tqdm(range(100)):
        time.sleep(0.01)

except requests.exceptions.HTTPError as err:
    print(err)
    exit(1)

#verificamos si es correcta la conexion es correcta a url_geo
try:
    r_geo = requests.get(url_geo)
    r_geo.raise_for_status()
    print("*****")
    print("*Conectado a datos GEOJSON UBICACIÓN SONOMETROS*")
    print("*****")

    from tqdm import tqdm
    import time
```

```
for i in tqdm(range(100)):
    time.sleep(0.01)

except requests.exceptions.HTTPError as err:
    print(err)
    exit(1)

#verificamos si es correcta la conexion es correcta a url_geo_trafico
try:
    r_geo_trafico = requests.get(url_geo_trafico)
    r_geo_trafico.raise_for_status()
    print("*****")
    print("*Conectado a datos GEOJSON UBICACIÓN TRÁFICO*")
    print("*****")

    from tqdm import tqdm
    import time
    for i in tqdm(range(100)):
        time.sleep(0.01)

except requests.exceptions.HTTPError as err:
    print(err)
    exit(1)

#datos JSON
with open('sonometros.json', 'wb') as f:
    f.write(respuesta.content)

#datos GEOJSON SONOMETROS en json
with open('sonometros1.json', 'w') as f:
    json.dump(r_geo.json(), f, indent=4)

#datos GEOJSON SONOMETROS en geojson
with open('ubicacionsonometros.geojson', 'w') as f:
    json.dump(r_geo.json(), f, indent=4)

#datos GEOJSON TRÁFICO
with open('trafico2.json', 'w') as f:
    json.dump(r_geo_trafico.json(), f, indent=4)

#datos GEOJSON TRÁFICO en geojson
with open('ubicaciontrafico.geojson', 'w') as f:
    json.dump(r_geo_trafico.json(), f, indent=4)
```

```

#infile
with open("sonometros.json", "rb") as infile:
    r = json.load(infile)

#infile
with open("sonometros1.json", "rb") as infile:
    r_geo = json.load(infile)

#infile
with open("trafico2.json", "rb") as infile:
    r_geo_trafico = json.load(infile)

#infile
with open("ubicaciontrafico.geojson", "rb") as infile:
    r_geo_trafico_geojson = json.load(infile)

#infile
with open("ubicacionsonometros.geojson", "rb") as infile:
    r_geo_geojson = json.load(infile)

# Crea un nuevo diccionario que combina los datos de ambos diccionarios
concatenated_data = []
for item_json in r:
    name = item_json["nombre_dispositivo"]
    for item_geojson in r_geo["features"]:
        if item_geojson["properties"]["name"] == name:
            concatenated_item = item_json
            concatenated_item["address"] = item_geojson["properties"]["address"]
            concatenated_item["status"] = item_geojson["properties"]["status"]

            #separar fecha_medicion creando hora y fecha por separado
            fecha_medicion = item_json["fecha_medicion"]
            fecha_medicion = fecha_medicion.split(" ")
            fecha = fecha_medicion[0]
            hora = fecha_medicion[1]
            concatenated_item["fecha"] = fecha
            concatenated_item["hora"] = hora

            #formato de hora
            hora = hora.split(":")
            hora = hora[0]
            hora = int(hora)

            if hora >= 0 and hora <= 6:
                concatenated_item["hora"] = "Madrugada"
            elif hora >= 7 and hora <= 12:
                concatenated_item["hora"] = "Mañana"

```

```

        elif hora >= 13 and hora <= 20:
            concatenated_item["hora"] = "Tarde"
        elif hora >= 21 and hora <= 23:
            concatenated_item["hora"] = "Noche"

#quitamos fecha_mediccion
del concatenated_item["fecha_mediccion"]

#longitud con dos decimales
longitude = float(item_geojson["properties"]["longitude"])

#latitud con dos decimales
latitude = float(item_geojson["properties"]["latitude"])

#añadir un campo nuevo para definir Ciudad = "Bilbao"
concatenated_item["ciudad"] = "Bilbao"
concatenated_item["longitude"] = longitude
concatenated_item["latitude"] = latitude
#sustituir . por , en longitude y latitud
concatenated_item["longitude"] =
str(concatenated_item["longitude"]).replace(".", ",")
concatenated_item["latitude"] =
str(concatenated_item["latitude"]).replace(".", ",")

concatenated_data.append(concatenated_item)
break

# Escribe el nuevo diccionario en un archivo JSON
with open("concatenated_data.json", "w") as outfile:
    json.dump(concatenated_data, outfile, indent=4)

#importar concatenated_data.json con pymongo al espacio de mongoDB
from pymongo import MongoClient
client = MongoClient('localhost', 27017)
db = client['sonometros']
collection = db['sonometros']
with open('concatenated_data.json') as f:
    file_data = json.load(f)

    for doc in file_data:
        doc['decibelios'] = float(doc['decibelios'])

    collection.insert_many(file_data)

#importamos concatenated_data.json a Power Bi
import pandas as pd
import json

```

```
with open('concatenated_data.json') as f:
    data = json.load(f)
df = pd.DataFrame(data)
df.to_csv('concatenated_data.csv', index=False)
```

c. Consultas a MongoDB con PyMongo

```
import time
from pymongo import MongoClient

#conexion a mongoDB
client = MongoClient('localhost', 27017)
db = client['sonometros']
collection = db['sonometros']

#comprobamos si conectamos a mongoDB
print(client.list_database_names())

#comprobamos si conectamos a laleccion
print(db.list_collection_names())

#menu consultas
print("1. ¿Cuántos registros hay en la base de datos de los sonómetros de Bilbao?")
print("2. Mostrar calles en Bilbao que cuentan con sonómetros")
print("3. ¿Cuántas veces ha contado un sonómetro en cada calle?")
print("4. ¿Cuál es el promedio de la calle con los decibelios más altos")
print("5. ¿Cuál es el promedio de la calle con los decibelios más bajos")
print("6. ¿Cuál es el registro más alto en decibelios de un sonómetro")
print("7. ¿Cuál es el registro más bajo en decibelios de un sonómetro")
print("8. Imprimir calles ordenadas de mayor a menor por promedio de decibelios")
print("9. ¿En cuál franja horaria(hora) se producen más decibelios?")
print("10. ¿En cuál franja horaria(hora) se producen menos decibelios?")
print("11. ¿En qué fecha se han producido más decibelios?")
print("12. ¿En qué fecha se han producido menos decibelios?")
print("13. ¿En qué fecha y hora se han producido más decibelios?")
print("14. ¿En qué fecha y hora se han producido menos decibelios?")
print("15. Promedio de decibelios de todo Bilbao")
print("16. Salir")

#while menu
while True:
    #selecionar consulta
    consulta = int(input("Selecciona una consulta: "))
    if consulta == 1:
```

```

#¿Cuántos sonómetros hay en la ciudad de Bilbao?
    print("hay un total de : ", collection.count_documents({"ciudad": "Bilbao"}),
"registros")

elif consulta == 2:

#¿Qué calles hay en Bilbao?
    print(collection.distinct("address"))

elif consulta == 3:
#¿Cuántas veces ha contado un sonómetro en cada calle?
    for i in collection.distinct("address"):
        print(i, collection.count_documents({"address": i}))

elif consulta == 4:
#Agregación para encontrar la calle con el promedio de decibelios más altos
    pipeline = [
        {"$group": {"_id": "$address", "promedio_decibelios": {"$avg": "$decibelios"}}, 
        {"$sort": {"promedio_decibelios": -1}},
        {"$limit": 1}
    ]

# Ejecutar la agregación y guardar los resultados en la variable "result"
result = list(collection.aggregate(pipeline))

# Imprimir el resultado de la agregación
print("La dirección con el promedio más alto en decibelios es:",
result[0]['_id'])
print("El promedio de decibelios es: ",result[0]['promedio_decibelios'])

elif consulta == 5:
# Agregación para encontrar la calle con el promedio de decibelios más bajos
    pipeline1 = [
        {"$group": {"_id": "$address", "promedio_decibelios": {"$avg": "$decibelios"}}, 
        {"$sort": {"promedio_decibelios": 1}},
        {"$limit": 1}
    ]

# Ejecutar la agregación y guardar los resultados en la variable "result"
result = list(collection.aggregate(pipeline1))

# Imprimir el resultado de la agregación
print("La dirección con el promedio más bajo en decibelios es:",
result[0]['_id'])
print("El promedio de decibelios es: ",result[0]['promedio_decibelios'])

```

```

elif consulta == 6:

    pipeline2 = [
        {"$group": {"_id": "$address", "max_decibelios": {"$max": "$decibelios"}},,
        {"$sort": {"max_decibelios": -1}},
        {"$limit": 1}
    ]

    # Ejecutar la agregación y guardar los resultados en la variable "result"
    result = list(collection.aggregate(pipeline2))

    # Imprimir el resultado de la agregación
    print("La dirección con el registro más alto en decibelios es:",
result[0]['_id'])
    print("Los decibelios totales son: ",result[0]['max_decibelios'])

elif consulta == 7:

    pipeline3 = [
        {"$group": {"_id": "$address", "min_decibelios": {"$min": "$decibelios"}},,
        {"$sort": {"min_decibelios": 1}},
        {"$limit": 1}
    ]

    # Ejecutar la agregación y guardar los resultados en la variable "result"
    result = list(collection.aggregate(pipeline3))

    # Imprimir el resultado de la agregación
    print("La dirección con el registro más bajo en decibelios es:",
result[0]['_id'])
    print("Los decibelios totales son: ",result[0]['min_decibelios'])

elif consulta == 8:
    #Promedio de decibelios de todos los sonómetros mostrando su calle al lado, el
    #valor de decibelios para a ser int en vez de string
    pipeline5 = [
        {"$group": {"_id": "$address", "promedio_decibelios": {"$avg": {"$toInt": "$decibelios"}},},
        {"$sort": {"promedio_decibelios": -1}}
    ]

    # Ejecutar la agregación y guardar los resultados en la variable "result"
    result = list(collection.aggregate(pipeline5))

    # Imprimir el resultado de la agregación
    for i in result:
        print(i['_id'], i['promedio_decibelios'])

```

```

print

elif consulta == 9:

    pipeline6 = [
        {"$group": {"_id": "$hora", "promedio_decibelios": {"$avg": {"$toInt": "$decibelios"}}, },
        {"$sort": {"promedio_decibelios": -1}},
        {"$limit": 1}
    ]

    # Ejecutar la agregación y guardar los resultados en la variable "result"
    result = list(collection.aggregate(pipeline6))

    # Imprimir el resultado de la agregación
    print("La franja horaria con el promedio más alto en decibelios es:",
result[0]['_id'])
    print("El promedio de decibelios es: ",result[0]['promedio_decibelios'])

elif consulta == 10:

    pipeline7 = [
        {"$group": {"_id": "$hora", "promedio_decibelios": {"$avg": {"$toInt": "$decibelios"}}, },
        {"$sort": {"promedio_decibelios": 1}},
        {"$limit": 1}
    ]

    # Ejecutar la agregación y guardar los resultados en la variable "result"
    result = list(collection.aggregate(pipeline7))

    # Imprimir el resultado de la agregación
    print("La franja horaria con el promedio más bajo en decibelios es:",
result[0]['_id'])
    print("El promedio de decibelios es: ",result[0]['promedio_decibelios'])

elif consulta == 11:

    pipeline8 = [
        {"$group": {"_id": "$fecha", "promedio_decibelios": {"$avg": {"$toInt": "$decibelios"}}, },
        {"$sort": {"promedio_decibelios": -1}},
        {"$limit": 1}
    ]

    # Ejecutar la agregación y guardar los resultados en la variable "result"
    result = list(collection.aggregate(pipeline8))

    # Imprimir el resultado de la agregación

```

```

        print("La fecha con el promedio más alto en decibelios es:",
result[0]['_id'])
        print("El promedio de decibelios es: ",result[0]['promedio_decibelios'])

    elif consulta == 12:

        pipeline9 = [
            {"$group": {"_id": "$fecha", "promedio_decibelios": {"$avg": "$toInt": "$decibelios"}},},
            {"$sort": {"promedio_decibelios": 1}},
            {"$limit": 1}
        ]

        # Ejecutar la agregación y guardar los resultados en la variable
"result"
        result = list(collection.aggregate(pipeline9))

        # Imprimir el resultado de la agregación
        print("La fecha con el promedio más bajo en decibelios es:",
result[0]['_id'])
        print("El promedio de decibelios es:
",result[0]['promedio_decibelios'])

    elif consulta == 13:
        # Agregación para encontrar la calle con el promedio de decibelios más altos
        # por fecha y franja horaria(hora)
        pipeline10 = [
            {"$group": {"_id": {"fecha": "$fecha", "hora": "$hora"}, "promedio_decibelios": {"$avg": {"$toInt": "$decibelios"}},}},
            {"$sort": {"promedio_decibelios": -1}},
            {"$limit": 1}
        ]

        # Ejecutar la agregación y guardar los resultados en la variable "result"
        result = list(collection.aggregate(pipeline10))

        # Imprimir el resultado de la agregación
        print("La fecha y franja horaria con el promedio más alto en decibelios son:",
result[0]['_id'])
        print("El promedio de decibelios es: ",result[0]['promedio_decibelios'])

    elif consulta == 14:
        # Agregación para encontrar la calle con el promedio de decibelios más bajos
        # por fecha y franja horaria(hora)
        pipeline11 = [
            {"$group": {"_id": {"fecha": "$fecha", "hora": "$hora"}, "promedio_decibelios": {"$avg": {"$toInt": "$decibelios"}},}},
            {"$sort": {"promedio_decibelios": 1}},
            {"$limit": 1}
        ]

```

```

# Ejecutar la agregación y guardar los resultados en la variable "result"
result = list(collection.aggregate(pipeline11))

# Imprimir el resultado de la agregación
print("La fecha y franja horaria con el promedio más bajo en decibelios son:",
result[0]['_id'])
print("El promedio de decibelios es: ",result[0]['promedio_decibelios'])

if consulta == 14:
    #Agregación para saber el promedio general de la suma de todas las calles

    pipeline12 = [
        {"$group": {"_id": "$address", "promedio_decibelios": {"$avg": {"$toInt": "$decibelios"}}}},
        {"$group": {"_id": "promedio general", "promedio_decibelios": {"$avg": "$promedio_decibelios"}}}
    ]

    # Ejecutar la agregación y guardar los resultados en la variable "result"
    result = list(collection.aggregate(pipeline12))

    # Imprimir el resultado de la agregación
    print("El promedio general de decibelios es:
",result[0]['promedio_decibelios'])

elif consulta == 15:
    break
else:
    print("Selecciona una opción válida")

```

7.

Salida del programa:

1. ¿Cuántos registros hay en la base de datos de los sonómetros de Bilbao?
2. Mostrar calles en Bilbao que cuentan con sonómetros
3. ¿Cuántas veces ha contado un sonómetro en cada calle?
4. ¿Cuál es el promedio de la calle con los decibelios más altos
5. ¿Cuál es el promedio de la calle con los decibelios más bajos
6. ¿Cuál es el registro más alto en decibelios de un sonómetro
7. ¿Cuál es el registro más bajo en decibelios de un sonómetro
8. Imprimir calles ordenadas de mayor a menor por promedio de decibelios

9. ¿En cuál franja horaria(hora) se producen más decibelios?
10. ¿En cuál franja horaria(hora) se producen menos decibelios?
11. ¿En qué fecha se han producido más decibelios?
12. ¿En qué fecha se han producido menos decibelios?
13. ¿En qué fecha y hora se han producido más decibelios?
14. ¿En qué fecha y hora se han producido menos decibelios?
15. Promedio de decibelios de todo Bilbao
16. Salir.

Selecciona una consulta: 1

hay un total de : 168878 registros

Selecciona una consulta: 2

'ALAMEDA REKALDE 27', 'ARBOLANTXA FRENTE 6', 'AUTONOMÍA 33 (ESQUINA GORDÓNIZ)(CÁMARA TRÁFICO)', 'AVENIDA MADARIAGA 3-5', 'BARRENKALE 7', 'BAR

RENKALE BARRENA 7', 'CAMPO VOLANTÍN (CON MUGICA)(CÁMARA TRÁFICO)', 'DE LA TORRE 5', 'FRAY JUAN 24', 'FRENTE IGLESIA DE SAN VICENTE (ANTZOKI)', 'G

ORDÓNIZ (CON GOYA)(CÁMARA TRÁFICO)', 'ITURRIBIDE 30', 'JARDINES 6', 'JUAN DE GARAI (ROTONDA)(CÁMARA TRÁFICO)', 'LEHENDAKARI AGUIRRE 20-26', 'LOS

CAÑOS DEBAJO SOLUCION SUR', 'MAURICE RAVEL (CON SAN VALENTÍN BERRIOTXOA)', 'MONTEVIDEO (FRENTE HOSPITAL)(CÁMARA TRÁFICO)', 'MUELLE RIPA (RIA)',

'PLAZA NUEVA (BALCÓN ASOCIACIÓN COMERCIANTES)', 'PLAZA UNAMUNO FAROLA 13', 'POZA 30', 'POZA 48', 'POZA 53', 'RIPA 5', 'RODRIGUEZ ARIAS 71 BIS'

, 'ROTONDA GARAIZAR', 'SANTA MARÍA 18', 'SANTUTXU (CON KARMELO 1)(CÁMARA TRÁFICO)', 'SOMERA 39', 'SOMERA 8', 'URIBITARTE 6', 'VIUDA DE EPALZA 3',

'VIUDA DE EPALZA FRENTE 11', 'ZABALBIDE CON SANTA CLARA', 'ZARANDOA ESQUINA COLEGIO', 'ZUNZUNEGUI JUNTO BOCA METRO', 'ZURBARAN CON PARTICULAR DE

ZURBARAN']

Selecciona una consulta: 3

ALAMEDA REKALDE 27 2937

ARBOLANTXA FRENTE 6 2890

AUTONOMÍA 33 (ESQUINA GORDÓNIZ)(CÁMARA TRÁFICO) 2932

AVENIDA MADARIAGA 3-5 2940

BARRENKALE 7 2858

BARRENKALE BARRENA 7 2932

CAMPO VOLANTÍN (CON MUGICA)(CÁMARA TRÁFICO) 2249

DE LA TORRE 5 2938

FRAY JUAN 24 2738

FRENTE IGLESIA DE SAN VICENTE (ANTZOKI) 2914

GORDÓNIZ (CON GOYA)(CÁMARA TRÁFICO) 2886

ITURRIBIDE 30 2950

JARDINES 6 2941

JUAN DE GARAI (ROTONDA)(CÁMARA TRÁFICO) 2937

LEHENDAKARI AGUIRRE 20-26 179

LOS CAÑOS DEBAJO SOLUCION SUR 3002

MAURICE RAVEL (CON SAN VALENTÍN BERRIOTXOA) 2339

MONTEVIDEO (FRENTE HOSPITAL)(CÁMARA TRÁFICO) 2203

MUELLE RIPA (RIA) 2770

PLAZA NUEVA (BALCÓN ASOCIACIÓN COMERCIANTES) 2941

PLAZA UNAMUNO FAROLA 13 2928

POZA 30 2797

POZA 48 2885

POZA 53 2955

RIPA 5 2783

RODRIGUEZ ARIAS 71 BIS 2944

ROTONDA GARAIZAR 2804

SANTA MARIA 18 2903

SANTUTXU (CON KARMELO 1)(CÁMARA TRÁFICO) 2939

SOMERA 39 2904

SOMERA 8 66484

URIBITARTE 6 2855

VIUDA DE EPALZA 3 2825

VIUDA DE EPALZA FRENTE 11 2942

ZABALBIDE CON SANTA CLARA 2940

ZARANDOA ESQUINA COLEGIO 2720

ZUNZUNEGUI JUNTO BOCA METRO 2897

ZURBARAN CON PARTICULAR DE ZURBARAN 2897

Selecciona una consulta: 4

La dirección con el promedio más alto en decibelios es: JUAN DE GARAI (ROTONDA)(CÁMARA TRÁFICO)

El promedio de decibelios es: 74.6685393258427

Selecciona una consulta: 5

La dirección con el promedio más bajo en decibelios es: LEHENDAKARI AGUIRRE 20-26

El promedio de decibelios es: 45.75698324022346

Selecciona una consulta: 6

La dirección con el registro más alto en decibelios es: SOMERA 8

Los decibelios totales son: 117.6

Selecciona una consulta: 7

La dirección con el registro más bajo en decibelios es: SOMERA 8

Los decibelios totales son: 22.4

Selecciona una consulta: 8

JUAN DE GARAI (ROTONDA)(CÁMARA TRÁFICO) 74.15083418454205

POZA 53 71.15499153976312

MAURICE RAVEL (CON SAN VALENTÍN BERRIOTXOA) 68.2389910218042

SANTUTXU (CON KARMELO 1)(CÁMARA TRÁFICO) 67.63661109220823

POZA 48 67.44055459272097

GORDÓNIZ (CON GOYA)(CÁMARA TRÁFICO) 67.2955647955648

MONTEVIDEO (FRENTE HOSPITAL)(CÁMARA TRÁFICO) 67.14525646845212

JARDINES 6 66.81944916695002

AUTONOMÍA 33 (ESQUINA GORDÓNIZ)(CÁMARA TRÁFICO) 66.3993860845839

ZUNZUNEGUI JUNTO BOCA METRO 66.27856403175699

ALAMEDA REKALDE 27 65.5110657133129

BARRENKALE BARRENA 7 64.20804911323329

FRAY JUAN 24 64.08838568298027
AVENIDA MADARIAGA 3-5 63.403401360544215
VIUDA DE EPALZA FRENTE 11 63.3256288239293
SANTA MARIA 18 62.99414398897692
PLAZA UNAMUNO FAROLA 13 61.67691256830601
CAMPO VOLANTÍN (CON MUGICA)(CÁMARA TRÁFICO) 61.614939973321476
POZA 30 61.491598140865214
ZABALBIDE CON SANTA CLARA 61.14591836734694
BARRENKALE 7 60.860391882435266
RODRIGUEZ ARIAS 71 BIS 60.55672554347826
FRENTE IGLESIA DE SAN VICENTE (ANTZOKI) 60.54392587508579
DE LA TORRE 5 60.33492171545269
ZARANDOA ESQUINA COLEGIO 59.97095588235294
ZURBARAN CON PARTICULAR DE ZURBARAN 59.40283051432517
VIUDA DE EPALZA 3 58.618761061946905
SOMERA 39 58.554752066115704
LOS CAÑOS DEBAJO SOLUCION SUR 57.14990006662225
ITURRIBIDE 30 57.03254237288136
ROTONDA GARAIZAR 57.023537803138375
PLAZA NUEVA (BALCÓN ASOCIACIÓN COMERCIANTES) 56.8884733083985
ARBOLANTXA FRENTE 6 56.58685121107266
URIBITARTE 6 56.2199649737303
SOMERA 8 55.02400577582576
MUELLE RIPA (RIA) 54.98628158844765
RIPA 5 47.74164570607258
LEHENDAKARI AGUIRRE 20-26 45.19553072625698

Selecciona una consulta: 9

La franja horaria con el promedio más alto en decibelios es: Noche

El promedio de decibelios es: 62.9722477396896

Selecciona una consulta: 10

La franja horaria con el promedio más bajo en decibelios es: Madrugada

El promedio de decibelios es: 52.94246647065931

Selecciona una consulta: 11

La fecha con el promedio más alto en decibelios es: 2023-01-16

El promedio de decibelios es: 65.42502736227654

Selecciona una consulta: 12

La fecha con el promedio más bajo en decibelios es: 2023-02-15

El promedio de decibelios es: 54.4737412019491

Selecciona una consulta: 13

La fecha y franja horaria con el promedio más alto en decibelios son: {'fecha': '2023-01-28', 'hora': 'Noche'}

El promedio de decibelios es: 72.07445255474452

Selecciona una consulta: 14

La fecha y franja horaria con el promedio más bajo en decibelios son: {'fecha': '2023-02-07', 'hora': 'Madrugada'}

El promedio de decibelios es: 45.554054054054056

Selecciona una consulta: 15

El promedio general de decibelios es: 61.439795915389524

Selecciona una consulta: 16

PS C:\Users\Iker\Desktop\retoApi>

a. Colab

```
# -*- coding: utf-8 -*-
"""Copia de Retofinal.ipynb
```

Automatically generated by Colaboratory.

Original file is located at

https://colab.research.google.com/drive/1HM6jIISCT_nqybD0eICmawG2n4WtJBV6

Importamos las librerías necesarias para trabajar, y los Json y geoJson necesarios:

```
concatenated_data.json
```

```
ubicacionsonometros.geojson
```

```
ubicaciontrafico.geojson
```

```
"""
```

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
import pickle
from sklearn.neighbors import NearestNeighbors
from sklearn.metrics import log_loss
from sklearn.metrics import mean_squared_error
from sklearn.metrics import f1_score
from google.colab import files
```

```
uploaded = files.upload()
```

```
"""Creamos un DataFrame con concatenated_data.json e imprimimos en pantalla."""
dfsonometros = pd.read_json('concatenated_data.json')
print(dfsonometros)
```

```
"""Instalación e importación de Geopandas para poder manipular los geojson."""
!pip install geopandas
```

```
import geopandas as gpd
```

```
"""Convertimos a DataFrame el geojson que contiene la ubicación de los sonómetros e
imprimimos los 3 primeros resultados para comprobar."""
dfubicacion = gpd.read_file('ubicacionsonometros.geojson')
dfubicacion.head(3)
```

```
print(dfubicacion)
```

```
"""Para trabajar la geometría del geoJson vamos a sacar los campos de Geometry, se
establece que la geometría de ubicación son puntos y sus respectivas coordenadas."""
dfubicacion ['geometry']
```

```
"""Imprimimos en pantalla el resultado con plot(), esto nos ubicará los puntos en una serie de coordenadas."""

dfubicacion.plot()

"""Lo mismo hacemos con la ubicación del tráfico."""

dftráfico = gpd.read_file('ubicaciontráfico.geojson')
dftráfico.head(3)

print(dftráfico)

dftráfico['geometry']

dftráfico.plot()

"""Hacemos la unión del primer gráfico del geojson con el segundo geojson, podremos ver el la incidencia del tráfico y los sonómetros en el mismo gráfico."""

fig, ax = plt.subplots()
# Graficar el primer dataframe de la ubicación con color azul
dfubicacion.plot(ax=ax, color='blue')
# Graficar el segundo dataframe del tráfico con color rojo
dftráfico.plot(ax=ax, color='black')
plt.show()

"""Queremos que esta representación sea más gráfica y debemos añadirle un mapa con contextily, instalamos e importamos."""

pip install contextily

"""Ahora podemos ver un mapa fusionando la ubicación de los sonómetros y la incidencia del tráfico."""

import matplotlib.pyplot as plt
import contextily as ctx

# Crear una figura y un conjunto de ejes para la gráfica
fig, ax = plt.subplots(figsize=(20, 20))
#coordenadas bilbao
ax.set_xlim(-2.99, -2.90)
ax.set_ylim(43.245, 43.290)
# Añadir un título
ax.set_title("Sonómetros y tráfico de Bilbao")
# Añadir una etiqueta para el eje X
ax.set_xlabel("Longitud")
# Añadir una etiqueta para el eje Y
ax.set_ylabel("Latitud")
```

```

# Graficar el primer dataframe con color azul
dfubicacion.plot(ax=ax, color='blue')
# Graficar el segundo dataframe con color rojo
dftrafico.plot(ax=ax, color='black')
# Agregar un mapa base de OpenStreetMap
ctx.add_basemap(ax, crs=dfubicacion.crs.to_string(),
source=ctx.providers.OpenStreetMap.Mapnik)
# Mostrar la gráfica
plt.show()

"""Empezamos a limpiar datos y a ordenar"""

result = dfsonometros.loc[dfsonometros['decibelios'] > 100]
result = result.sort_values(by='decibelios', ascending=False)
print(result)

"""Ahora buscamos las calles con el mayor promedio de decibelios ordenados de mayor a menor."""

result1 =
dfsonometros.groupby('address')['decibelios'].mean().sort_values(ascending=False)
print(result1)

"""Hacemos un gráfico con el promedio."""

import matplotlib.pyplot as plt

result2 =
dfsonometros.groupby('address')['decibelios'].mean().sort_values(ascending=False)

plt.figure(figsize=(10, 6))
plt.bar(result2.index, result2.values)
plt.xticks(rotation=90)
plt.xlabel('Dirección')
plt.ylabel('Promedio de decibelios')
plt.title('Promedio de decibelios por dirección')
plt.show()

import numpy as np

def categorize_sound(decibels):
    conditions = [decibels < 20, (decibels >= 20) & (decibels < 40), (decibels >= 40) &
    (decibels < 60), (decibels >= 60) & (decibels < 80),
                  (decibels >= 80) & (decibels < 100), (decibels >= 100) & (decibels <
    120), decibels >= 120]
    choices = ['Suave', 'Moderado', 'Fuerte', 'Muy fuerte', 'Molesto', 'Extremadamente
    Molesto', 'Doloroso']
    return np.select(conditions, choices, default='Invalido')

```

```
result['categoría_sonido'] = result['decibelios'].apply(categorize_sound)

print(result)

"""Promedio de todos los sonómetros de Bilbao"""

promedio = dfsonometros['decibelios'].mean()
categoria_promedio = categorize_sound(round(promedio, 2))
print(f"El promedio total de decibelios es {promedio:.2f}, que corresponde a la categoría de sonido '{categoria_promedio}'.")

"""Para trabajar tenemos que saber que tipos y que columnas tenemos en cada DataFrame."""

dfsonometros.dtypes

dfubicacion.columns

dfsonometros.columns

dfsonometros.info()

dfsonometros.value_counts()

dfsonometros.shape

dfsonometros.describe

dfsonometros.describe()

"""Veces que han sido contados la cantidad exacta de Decibelios, situando los más frecuentes entre 50 y 75 decibelios."""

import matplotlib.pyplot as plt
import seaborn as sns
plt.figure(figsize=(10,10))
plt.style.use('ggplot')
sns.histplot(dfsonometros.decibelios, kde=True)

"""Recogemos el Status de los sonómetros podemos ver que de las 169430 veces que ha recogido el sonido:

* 1 representan las veces que funcionan
* 2 representan las veces que no recogen sonido

"""

dfsonometros['status'].value_counts()
```

```

plt.figure(figsize=(10,10))
plt.style.use('ggplot')
sns.countplot(dfsonometros.status)

"""
```
Tiene formato de código
```

```

El que más muestras ha recogido corresponde al dispositivo BI-RUI-C005 que corresponde a Somera 8"""

```

plt.figure(figsize=(20,8))
plt.style.use('ggplot')
plt.xticks(rotation=90)
sns.histplot(dfsonometros.address)

```

"""Figuras en forma de violín, su altura representa el rango de decibelios, su anchura las veces que ha sido contado ese número dentro del rango."""

```

plt.figure(figsize=(20,10))
plt.style.use('grayscale')
plt.xticks(rotation=90)
sns.violinplot(dfsonometros.address, dfsonometros.decibelios)

```

```

plt.style.use('seaborn-dark-palette')
sns.pairplot(dfsonometros ,hue='address', height=10)
plt.show()

plt.style.use('seaborn-dark-palette')
sns.pairplot(dfsonometros, hue='hora', height=10)
plt.show()

```

"""Previsiones"""

```

from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

# Separar las características (X) y la etiqueta (y)
X = dfsonometros[['decibelios']]
y = dfsonometros['status']

# Dividir los datos en conjuntos de entrenamiento y prueba
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Crear el modelo de árbol de decisión

```

```
clf = DecisionTreeClassifier()

# Entrenar el modelo con los datos de entrenamiento
clf.fit(X_train, y_train)

# Realizar predicciones en los datos de prueba
y_pred = clf.predict(X_test)

# Calcular la precisión del modelo
accuracy = accuracy_score(y_test, y_pred)
print('Precisión:', accuracy)

pip install graphviz

pip install pydotplus

from sklearn.tree import export_graphviz
from IPython.display import Image
import pydotplus

dot_data = export_graphviz(clf, out_file=None,
                           feature_names=['decibelios'],
                           class_names=['no cumplido', 'cumplido'],
                           filled=True, rounded=True,
                           special_characters=True)
graph = pydotplus.graph_from_dot_data(dot_data)
Image(graph.create_png())

import pandas as pd
from sklearn.linear_model import LinearRegression

# Cargamos los datos del DataFrame dfsonometros en df2sonometros
df2sonometros = dfsonometros.copy()

# Convertimos la columna "decibelios" de cadenas a números
df2sonometros['decibelios'] = pd.to_numeric(df2sonometros['decibelios'])

# Reemplazamos comas por puntos en las columnas "longitude" y "latitude"
df2sonometros['longitude'] = df2sonometros['longitude'].str.replace(',',
'.').astype(float)
df2sonometros['latitude'] = df2sonometros['latitude'].str.replace(',', '.').astype(float)

# Dividimos los datos en conjunto de entrenamiento y conjunto de prueba
train_data = df2sonometros.sample(frac=0.8, random_state=1)
test_data = df2sonometros.drop(train_data.index)

# Entrenamos un modelo de regresión lineal utilizando la columna "decibelios" como
# variable de entrada
# y las columnas "longitude" y "latitude" como variables de salida
model = LinearRegression()
```

```
model.fit(train_data[['longitude', 'latitude']], train_data['decibelios'])

# Hacemos predicciones en el conjunto de prueba
predictions = model.predict(test_data[['longitude', 'latitude']])

# Imprimimos las primeras 10 predicciones y los valores reales correspondientes
print(predictions[:10])
print(test_data['decibelios'][:10])

import matplotlib.pyplot as plt

plt.figure(figsize=(20,10))
df_promedio = dfsonometros.groupby('fecha')['decibelios'].mean().reset_index()
df_promedio['categoria_sonido'] = df_promedio['decibelios'].apply(categorize_sound)

plt.plot(df_promedio['fecha'], df_promedio['decibelios'])

for i, row in df_promedio.iterrows():
    plt.text(row['fecha'], row['decibelios'], row['categoria_sonido'], ha='center',
va='bottom')

plt.xlabel('Fecha')
plt.ylabel('Promedio de decibelios')
plt.title('Evolución del promedio de decibelios por fecha')
plt.xticks(rotation=90)
plt.show()

import pandas as pd
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression

# Cargar los datos en un DataFrame
df2sonometros = dfsonometros.copy()

# Convertir la columna "decibelios" de cadenas a números
df2sonometros['decibelios'] = pd.to_numeric(df2sonometros['decibelios'])

# Reemplazar comas por puntos en las columnas "longitude" y "latitude"
df2sonometros['longitude'] = df2sonometros['longitude'].str.replace(',',
' .').astype(float)
df2sonometros['latitude'] = df2sonometros['latitude'].str.replace(',',
' .').astype(float)

# Graficar los datos en un diagrama de dispersión
plt.scatter(df2sonometros['longitude'], df2sonometros['latitude'],
c=df2sonometros['decibelios'])
plt.xlabel('Longitude')
plt.ylabel('Latitude')
plt.title('Decibelios en función de la ubicación')
plt.colorbar()
```

```
plt.show()

# Dividir los datos en conjunto de entrenamiento y conjunto de prueba
train_data = df2sonometros.sample(frac=0.8, random_state=1)
test_data = df2sonometros.drop(train_data.index)

# Entrenar un modelo de regresión lineal utilizando la columna "decibelios" como variable
# de entrada
# y las columnas "longitude" y "latitude" como variables de salida
model = LinearRegression()
model.fit(train_data[['longitude', 'latitude']], train_data['decibelios'])

# Hacer predicciones en el conjunto de prueba
predictions = model.predict(test_data[['longitude', 'latitude']])

# Imprimir las primeras 20 predicciones y los valores reales correspondientes
print(predictions[:20])
print(test_data['decibelios'][:20])

import pandas as pd
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression

# Cargar los datos en un DataFrame
df2sonometros = dfsonometros.copy()

# Convertir la columna "decibelios" de cadenas a números
df2sonometros['decibelios'] = pd.to_numeric(df2sonometros['decibelios'])

# Reemplazar comas por puntos en las columnas "longitude" y "latitude"
df2sonometros['longitude'] = df2sonometros['longitude'].str.replace(',', '.').astype(float)
df2sonometros['latitude'] = df2sonometros['latitude'].str.replace(',', '.').astype(float)

# Añadir la columna "Address" al DataFrame df2sonometros
df2sonometros['address'] = dfsonometros['address']

# Graficar los datos en un diagrama de dispersión
fig, ax = plt.subplots()
colors = df2sonometros['address'].astype('category').cat.codes
scatter = ax.scatter(df2sonometros['longitude'], df2sonometros['latitude'], c=colors, cmap='tab20', alpha=0.8)

# Eliminar los nombres de las calles del gráfico
ax.set_xticklabels([])
ax.set_yticklabels([])
ax.set_xticks([])
ax.set_yticks([])

# Añadir título al gráfico
```

```

ax.set_title('Decibelios en función de la ubicación')

plt.show()

# Dividir los datos en conjunto de entrenamiento y conjunto de prueba
train_data = df2sonometros.sample(frac=0.8, random_state=1)
test_data = df2sonometros.drop(train_data.index)

# Entrenar un modelo de regresión lineal utilizando la columna "decibelios" como variable
# de entrada
# y las columnas "longitude" y "latitude" como variables de salida
model = LinearRegression()
model.fit(train_data[['longitude', 'latitude']], train_data['decibelios'])

for address in test_data['address'].unique():
    filtered_data = test_data[test_data['address'] == address]
    # Calcular las predicciones para el conjunto de prueba
    predictions = model.predict(filtered_data[['longitude', 'latitude']])
    # Calcular el promedio de las predicciones para cada dirección
    average = predictions.mean()
    # Mostrar el resultado promediado
    print(f"Promedio de las predicciones para {address}: {average:.2f}")
    print(f"Valores reales para {address}:")
    print(filtered_data['decibelios'].head())
    print()

promedios = []

for address in test_data['address'].unique():
    filtered_data = test_data[test_data['address'] == address]
    # Calcular las predicciones para el conjunto de prueba
    predictions = model.predict(filtered_data[['longitude', 'latitude']])
    # Calcular el promedio de las predicciones para cada dirección
    average = predictions.mean()
    # Agregar el promedio a la lista de promedios
    promedios.append((address, average))

# Ordenar los promedios de mayor a menor
promedios_ordenados = sorted(promedios, key=lambda x: x[1], reverse=True)

# Mostrar los resultados
for direccion, promedio in promedios_ordenados:
    print("Dirección: ", direccion)
    print("Promedio de las predicciones: ", promedio)
    print()

promedios = []

for address in test_data['address'].unique():
    filtered_data = test_data[test_data['address'] == address]

```

```

# Agrupar los datos por dirección y hora
grouped_data = filtered_data.groupby(['address', 'hora'])
# Calcular las predicciones para cada grupo de datos
predictions = grouped_data.apply(lambda x: model.predict(x[['longitude',
'latitude']]).mean())
# Obtener la hora en la que se producen más decibelios por calle
max_hour = predictions.idxmax()[1]
# Obtener el promedio de las predicciones para cada dirección
average = predictions.mean()
# Agregar la dirección, el promedio y la hora en la que se producen más decibelios a
la lista de promedios
promedios.append((address, average, max_hour))

# Ordenar los promedios de mayor a menor
promedios_ordenados = sorted(promedios, key=lambda x: x[1], reverse=True)

# Mostrar los resultados
for direccion, promedio, hora in promedios_ordenados:
    print("Dirección: ", direccion)
    print("Hora en la que se producen más decibelios: ", hora)
    print("Promedio de las predicciones: ", promedio)
    print()

```

b. Red neuronal

Código:

```

import pandas as pd
import tensorflow as tf
from sklearn.model_selection import train_test_split

df2sonometros = dfsonometros.copy()
df2sonometros['decibelios'] = pd.to_numeric(df2sonometros['decibelios'])
df2sonometros['longitude'] = df2sonometros['longitude'].str.replace(',', '.').astype(
float)
df2sonometros['latitude'] = df2sonometros['latitude'].str.replace(',', '.').astype(
float)

train_data, test_data = train_test_split(df2sonometros, test_size=0.2, random_state
=1)

model = tf.keras.Sequential([
    tf.keras.layers.Dense(64, activation='relu', input_shape=[2]), # Capa oculta con 64 neuronas
    tf.keras.layers.Dense(1) # Capa de salida con una neurona
])

```

```
model.compile(optimizer=tf.optimizers.Adam(), loss='mean_squared_error')

history = model.fit(train_data[['longitude', 'latitude']], train_data['decibelios']
,
validation_split=0.2, epochs=50)

# Evalua el modelo en el conjunto de prueba
loss = model.evaluate(test_data[['longitude', 'latitude']], test_data['decibelios'])
)
print(f"Pérdida en el conjunto de prueba: {loss:.2f}")
```

Salida:

```
Epoch 1/50
3378/3378 [=====] - 17s 5ms/step - loss: 141.7113 - val_loss:
112.9816
Epoch 2/50
3378/3378 [=====] - 16s 5ms/step - loss: 113.5241 - val_loss:
113.1497
Epoch 3/50
3378/3378 [=====] - 9s 3ms/step - loss: 113.5076 - val_loss:
114.3136
Epoch 4/50
3378/3378 [=====] - 9s 3ms/step - loss: 113.4613 - val_loss:
113.0379
Epoch 5/50
3378/3378 [=====] - 9s 3ms/step - loss: 113.4746 - val_loss:
113.3092
Epoch 6/50
3378/3378 [=====] - 10s 3ms/step - loss: 113.4670 - val_loss:
112.9780
Epoch 7/50
3378/3378 [=====] - 9s 3ms/step - loss: 113.4054 - val_loss:
113.5246
Epoch 8/50
3378/3378 [=====] - 8s 2ms/step - loss: 113.4436 - val_loss:
113.6161
Epoch 9/50
3378/3378 [=====] - 9s 3ms/step - loss: 113.4252 - val_loss:
114.6170
Epoch 10/50
3378/3378 [=====] - 12s 4ms/step - loss: 113.4220 - val_loss:
113.9100
Epoch 11/50
3378/3378 [=====] - 19s 6ms/step - loss: 113.4637 - val_loss:
113.3031
Epoch 12/50
3378/3378 [=====] - 10s 3ms/step - loss: 113.4930 - val_loss:
114.6783
Epoch 13/50
3378/3378 [=====] - 9s 3ms/step - loss: 113.4128 - val_loss:
113.0704
Epoch 14/50
3378/3378 [=====] - 10s 3ms/step - loss: 113.3653 - val_loss:
114.4843
Epoch 15/50
```

```
3378/3378 [=====] - 9s 3ms/step - loss: 113.4306 - val_loss:  
113.0191  
Epoch 16/50  
3378/3378 [=====] - 10s 3ms/step - loss: 113.3891 - val_loss:  
113.4418  
Epoch 17/50  
3378/3378 [=====] - 10s 3ms/step - loss: 113.3715 - val_loss:  
114.3683  
Epoch 18/50  
3378/3378 [=====] - 10s 3ms/step - loss: 113.4123 - val_loss:  
113.1217  
Epoch 19/50  
3378/3378 [=====] - 9s 3ms/step - loss: 113.3983 - val_loss:  
113.6387  
Epoch 20/50  
3378/3378 [=====] - 10s 3ms/step - loss: 113.3677 - val_loss:  
113.1346  
Epoch 21/50  
3378/3378 [=====] - 9s 3ms/step - loss: 113.4326 - val_loss:  
112.9117  
Epoch 22/50  
3378/3378 [=====] - 10s 3ms/step - loss: 113.3063 - val_loss:  
115.1446  
Epoch 23/50  
3378/3378 [=====] - 11s 3ms/step - loss: 113.3771 - val_loss:  
113.8756  
Epoch 24/50  
3378/3378 [=====] - 10s 3ms/step - loss: 113.3721 - val_loss:  
114.0154  
Epoch 25/50  
3378/3378 [=====] - 9s 3ms/step - loss: 113.3790 - val_loss:  
113.2673  
Epoch 26/50  
3378/3378 [=====] - 8s 2ms/step - loss: 113.3665 - val_loss:  
113.0266  
Epoch 27/50  
3378/3378 [=====] - 10s 3ms/step - loss: 113.3994 - val_loss:  
113.1870  
Epoch 28/50  
3378/3378 [=====] - 9s 3ms/step - loss: 113.3677 - val_loss:  
113.4376  
Epoch 29/50  
3378/3378 [=====] - 10s 3ms/step - loss: 113.3071 - val_loss:  
113.6437  
Epoch 30/50  
3378/3378 [=====] - 9s 3ms/step - loss: 113.4546 - val_loss:  
113.0657  
Epoch 31/50  
3378/3378 [=====] - 10s 3ms/step - loss: 113.3180 - val_loss:  
112.9993  
Epoch 32/50  
3378/3378 [=====] - 10s 3ms/step - loss: 113.3122 - val_loss:  
112.8733  
Epoch 33/50  
3378/3378 [=====] - 10s 3ms/step - loss: 113.2761 - val_loss:  
113.1863  
Epoch 34/50  
3378/3378 [=====] - 12s 3ms/step - loss: 113.2706 - val_loss:  
113.5405  
Epoch 35/50  
3378/3378 [=====] - 9s 3ms/step - loss: 113.2522 - val_loss:  
114.3599
```

```

Epoch 36/50
3378/3378 [=====] - 11s 3ms/step - loss: 113.2982 - val_loss:
112.8685
Epoch 37/50
3378/3378 [=====] - 10s 3ms/step - loss: 113.3330 - val_loss:
113.1101
Epoch 38/50
3378/3378 [=====] - 10s 3ms/step - loss: 113.3314 - val_loss:
112.8706
Epoch 39/50
3378/3378 [=====] - 11s 3ms/step - loss: 113.2717 - val_loss:
113.0394
Epoch 40/50
3378/3378 [=====] - 9s 3ms/step - loss: 113.2934 - val_loss:
113.2528
Epoch 41/50
3378/3378 [=====] - 10s 3ms/step - loss: 113.2614 - val_loss:
112.9791
Epoch 42/50
3378/3378 [=====] - 10s 3ms/step - loss: 113.2691 - val_loss:
112.8333
Epoch 43/50
3378/3378 [=====] - 8s 2ms/step - loss: 113.2202 - val_loss:
112.9047
Epoch 44/50
3378/3378 [=====] - 10s 3ms/step - loss: 113.2295 - val_loss:
112.8284
Epoch 45/50
3378/3378 [=====] - 10s 3ms/step - loss: 113.1984 - val_loss:
112.8491
Epoch 46/50
3378/3378 [=====] - 11s 3ms/step - loss: 113.1910 - val_loss:
113.4716
Epoch 47/50
3378/3378 [=====] - 10s 3ms/step - loss: 113.2270 - val_loss:
113.1795
Epoch 48/50
3378/3378 [=====] - 10s 3ms/step - loss: 113.2467 - val_loss:
113.6813
Epoch 49/50
3378/3378 [=====] - 9s 3ms/step - loss: 113.2316 - val_loss:
112.9411
Epoch 50/50
3378/3378 [=====] - 10s 3ms/step - loss: 113.2138 - val_loss:
113.0298
1056/1056 [=====] - 2s 2ms/step - loss: 112.9355
Pérdida en el conjunto de prueba: 112.94

```

Gráfica:

```

import matplotlib.pyplot as plt
plt.figure(figsize=(20,10))
# Graficando la evolución de la pérdida en el conjunto de entrenamiento y de prueba
plt.plot(history.history['loss'], label='Training loss')
plt.plot(history.history['val_loss'], label='Validation loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()
plt.show()

```

