NOMBRE: Jonnathan Alexander Pérez Ochoa
Laboratorio No. 2. Exploración de librerías para cargue de datos.

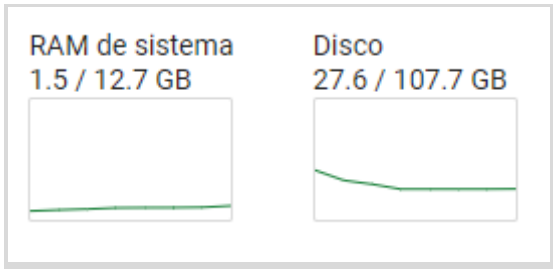
PANDAS

Primer experimento.

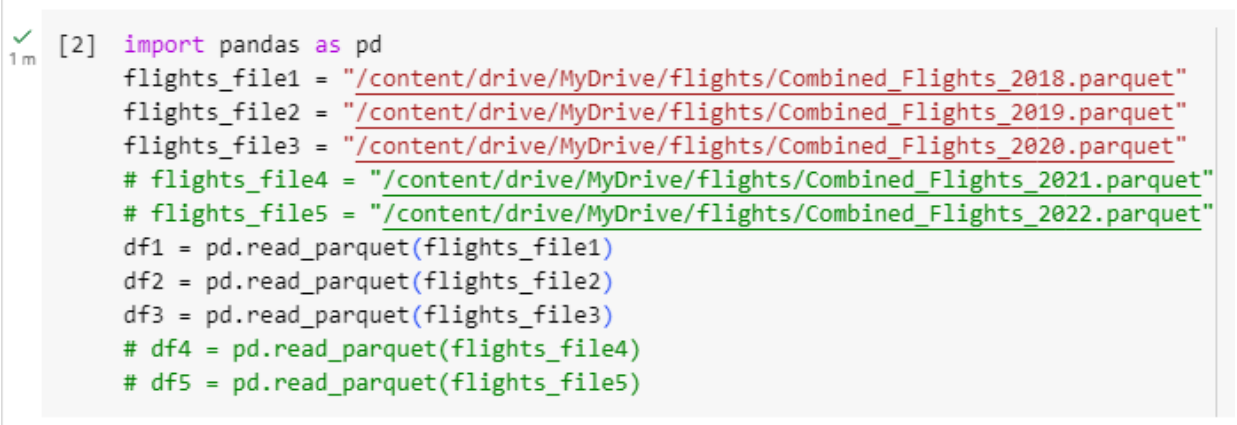
Se van a cargar los primeros tres (3) archivos del conjunto de datos, correspondientes a:

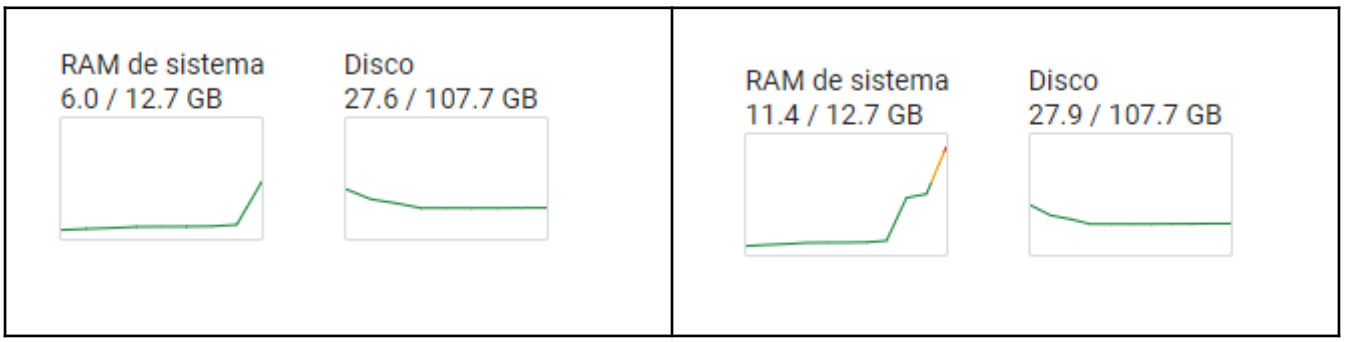
- /content/drive/MyDrive/flights/Combined_Flights_2018.parquet
- /content/drive/MyDrive/flights/Combined_Flights_2019.parquet
- /content/drive/MyDrive/flights/Combined_Flights_2020.parquet

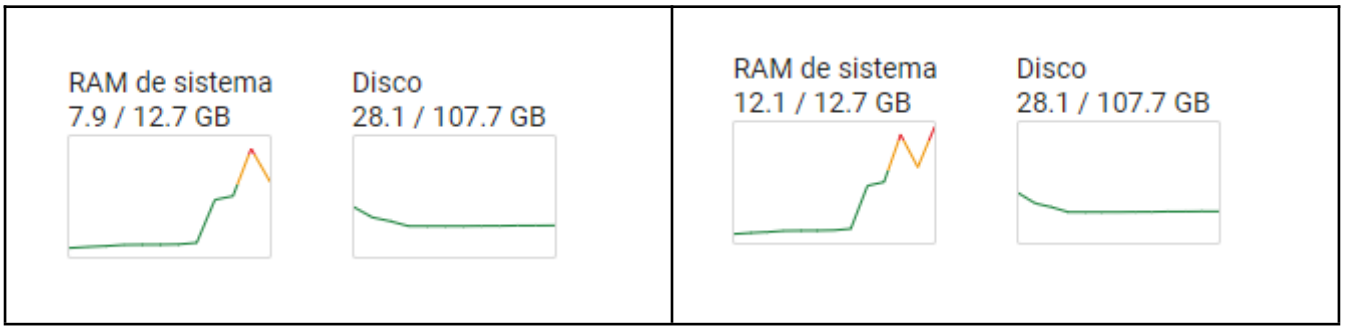
Uso de los recursos de RAM y Disco antes de iniciar las pruebas:



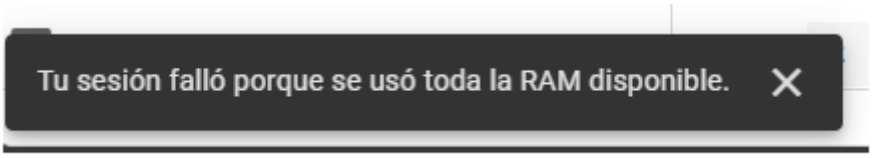
1. En la primera parte, vemos como el valor de la memoria RAM va aumentando a medida que se va ejecutando la línea de código, encontramos picos de consumo de la memoria RAM que llegan hasta los **12.1 GB** de los **12.7 GB** que tenemos disponibles. Por otro lado, el uso del Disco no presenta un aumento significativo en el transcurso de la prueba. La ejecución tardó aproximadamente **un minuto** y finalizó exitosamente.

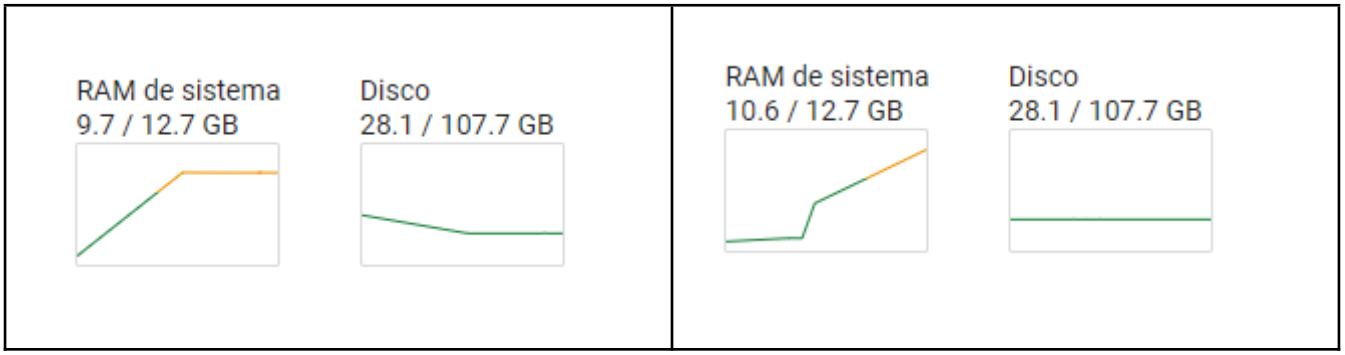






2. En la segunda parte, la ejecución se ha detenido por sobrepasar el límite de la memoría RAM, no ha sido posible concatenar los 3 archivos que habíamos cargado previamente. El resultado es fallido para esta prueba.





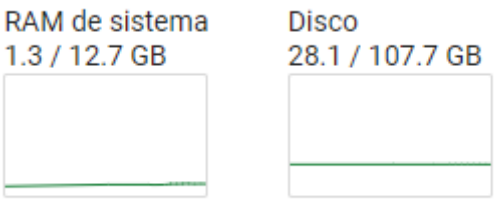
Polars

Primer experimento.

Se van a cargar los cinco (5) archivos del conjunto de datos, correspondientes a:

- /content/drive/MyDrive/flights/Combined_Flights_2018.parquet
- /content/drive/MyDrive/flights/Combined_Flights_2019.parquet
- /content/drive/MyDrive/flights/Combined_Flights_2020.parquet
- /content/drive/MyDrive/flights/Combined_Flights_2021.parquet
- /content/drive/MyDrive/flights/Combined_Flights_2022.parquet

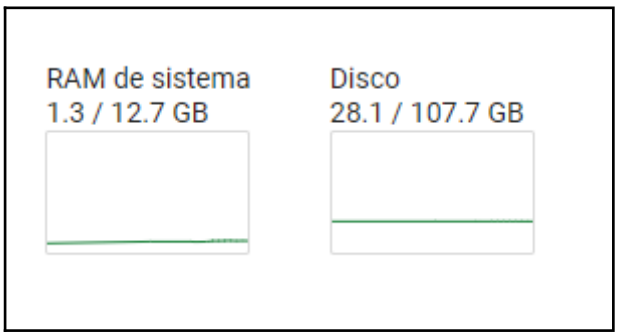
Uso de los recursos de RAM y Disco antes de iniciar las pruebas:



1. En la primera parte, la ejecución se realiza de manera exitosa, el uso de recursos se mantuvo en el mismo rango de valores y ha terminado la ejecución rápidamente sin mayor percance.

```
[1]  import polars as pl
```

```
[2]  flights_file1 = "/content/drive/MyDrive/flights/Combined_Flights_2018.parquet"
     flights_file2 = "/content/drive/MyDrive/flights/Combined_Flights_2019.parquet"
     flights_file3 = "/content/drive/MyDrive/flights/Combined_Flights_2020.parquet"
     flights_file4 = "/content/drive/MyDrive/flights/Combined_Flights_2021.parquet"
     flights_file5 = "/content/drive/MyDrive/flights/Combined_Flights_2022.parquet"
     df1 = pl.scan_parquet(flights_file1)
     df2 = pl.scan_parquet(flights_file2)
     df3 = pl.scan_parquet(flights_file3)
     df4 = pl.scan_parquet(flights_file4)
     df5 = pl.scan_parquet(flights_file5)
```

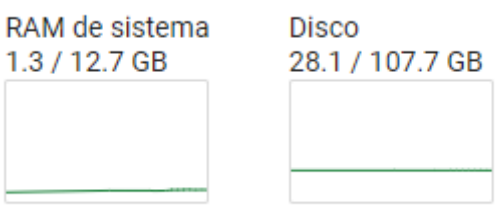


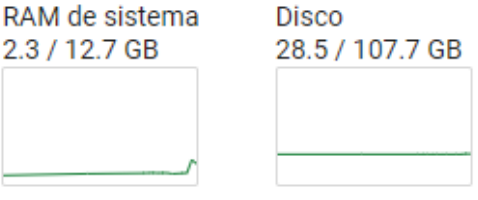
2. En la segunda parte, la ejecución ha tenido una duración de 1 minuto y 27 segundos aproximadamente. Durante la ejecución, el uso de memoria RAM variaba tanto de subida y bajada sin llegar a un tope máximo con respecto a la memoria total disponible. La ejecución termina exitosamente.

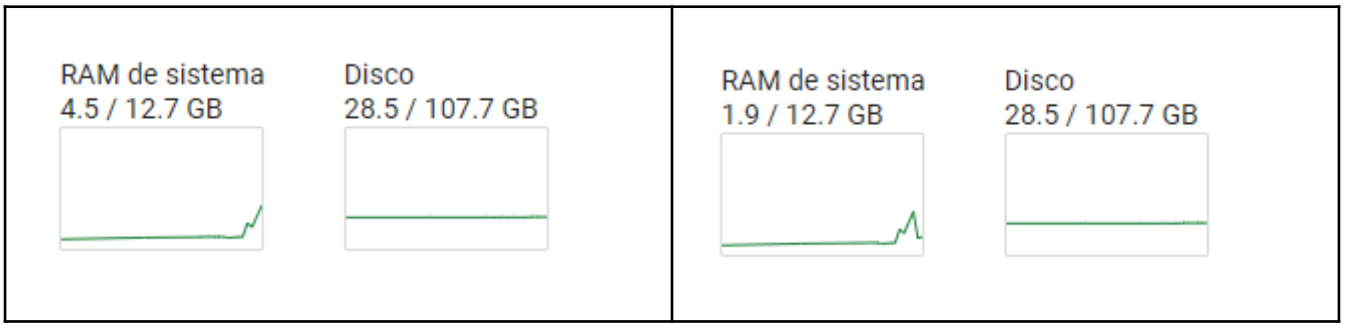
```
[3]  %%timeit

     df_polars = (
         pl.concat([df1, df2, df3, df4, df5])
         .groupby(['Airline', 'Year'])
         .agg([
             pl.col("DepDelayMinutes").mean().alias("avg_dep_delay"),
             pl.col("DepDelayMinutes").sum().alias("sum_dep_delay"),
             pl.col("DepDelayMinutes").max().alias("max_dep_delay"),
             pl.col("ArrDelayMinutes").mean().alias("avg_arr_delay"),
             pl.col("ArrDelayMinutes").sum().alias("sum_arr_delay"),
             pl.col("ArrDelayMinutes").max().alias("max_arr_delay"),
           ])
     ).collect()

     df_polars.write_parquet('temp_polars.parquet')
```







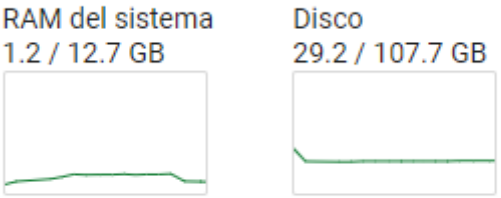
PySpark

Primer experimento.

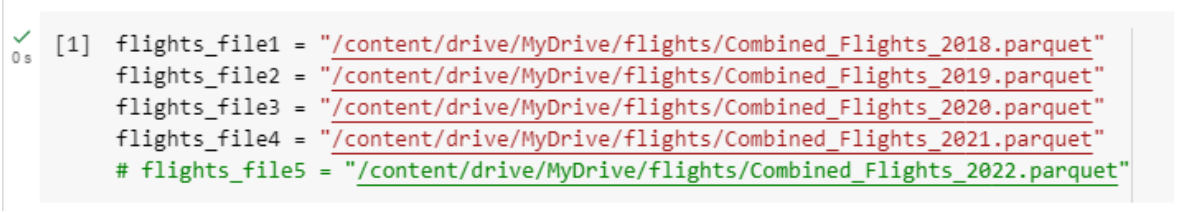
Se van a cargar los primeros cuatro (4) archivos del conjunto de datos, correspondientes a:

- /content/drive/MyDrive/flights/Combined_Flights_2018.parquet
- /content/drive/MyDrive/flights/Combined_Flights_2019.parquet
- /content/drive/MyDrive/flights/Combined_Flights_2020.parquet
- /content/drive/MyDrive/flights/Combined_Flights_2021.parquet

Uso de los recursos de RAM y Disco antes de iniciar las pruebas:



1. En la primera parte, el consumo de memoria RAM y espacio del disco es mínima, se cargan los archivos de manera correcta y el tiempo de ejecución no supera un (1) segundo.

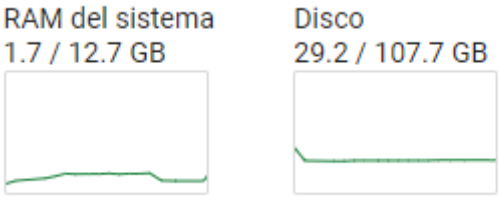


2. En la segunda parte, la lectura de los archivos se realiza en un tiempo de **ocho (8) segundos** aproximadamente, el consumo de RAM para este proceso se ve ligeramente incrementado:

- Ejecución de la línea para lectura de los archivos.

```
[6]  df_spark1 = spark.read.parquet(flights_file1)
     df_spark2 = spark.read.parquet(flights_file2)
     df_spark3 = spark.read.parquet(flights_file3)
     df_spark4 = spark.read.parquet(flights_file4)
     # df_spark5 = spark.read.parquet(flights_file5)
```

- Recursos del sistema al momento de finalizar la prueba.

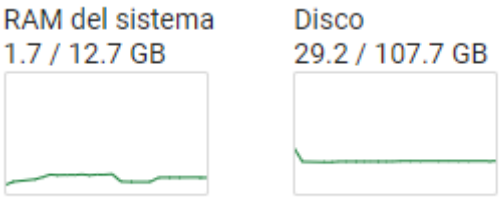


3. En la tercera parte, el proceso de unión entre los archivos se realiza rápidamente, el uso de los recursos es mínimo y no se aprecia incremento en la memoria RAM y el disco.

- Ejecución para la unión de los archivos

```
[7]  df_spark = df_spark1.union(df_spark2)
     df_spark = df_spark.union(df_spark3)
     df_spark = df_spark.union(df_spark4)
     # df_spark = df_spark.union(df_spark5)
```

- Recursos del sistema al momento de finalizar la prueba.



4. En la cuarta parte, la ejecución tuvo una duración de **un minuto y 15 segundos** arrojando un resultado exitoso. El consumo de memoria RAM tuvo un aumento de **0.2 GB** con respecto a la gráfica del paso anterior.

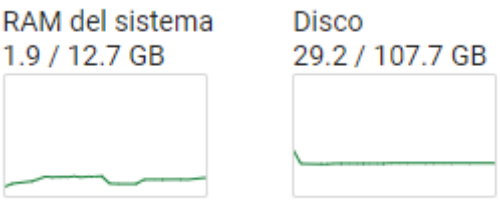
-

```
%%timeit

df_spark_agg = df_spark.groupby("Airline", "Year").agg(
    avg("ArrDelayMinutes").alias('avg_arr_delay'),
    sum("ArrDelayMinutes").alias('sum_arr_delay'),
    max("ArrDelayMinutes").alias('max_arr_delay'),
    avg("DepDelayMinutes").alias('avg_dep_delay'),
    sum("DepDelayMinutes").alias('sum_dep_delay'),
    max("DepDelayMinutes").alias('max_dep_delay'),
)
df_spark_agg.write.mode('overwrite').parquet('temp_spark.parquet')
```

```
8.32 s ± 1.12 s per loop (mean ± std. dev. of 7 runs, 1 loop each)
```

- Recursos del sistema al momento de finalizar la prueba.



Dask

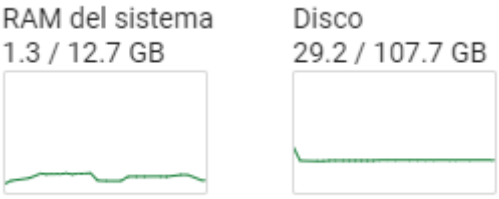
Primer experimento.

Antes de iniciar con el proceso de pruebas en la librería **Dask, es necesario reiniciar la sesión en el entorno de ejecución**

Se van a cargar los cinco (5) archivos del conjunto de datos, correspondientes a:

- /content/drive/MyDrive/flights/Combined_Flights_2018.parquet
- /content/drive/MyDrive/flights/Combined_Flights_2019.parquet
- /content/drive/MyDrive/flights/Combined_Flights_2020.parquet
- /content/drive/MyDrive/flights/Combined_Flights_2021.parquet
- /content/drive/MyDrive/flights/Combined_Flights_2021.parquet

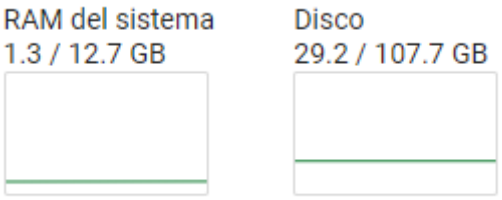
Uso de los recursos de RAM y Disco antes de iniciar las pruebas:



1. En la primera parte se cargaron y leyeron los archivos, con un tiempo de ejecución de aproximadamente **2 segundos.** El uso de memoria RAM no presenta cambios con respecto a las cifras de la gráfica inicial.

```
✓ 2 s  [1]  import pandas as pd
            import dask.dataframe as dd
            flights_file1 = "/content/drive/MyDrive/flights/Combined_Flights_2018.parquet"
            flights_file2 = "/content/drive/MyDrive/flights/Combined_Flights_2019.parquet"
            flights_file3 = "/content/drive/MyDrive/flights/Combined_Flights_2020.parquet"
            flights_file4 = "/content/drive/MyDrive/flights/Combined_Flights_2021.parquet"
            flights_file5 = "/content/drive/MyDrive/flights/Combined_Flights_2022.parquet"
            df1 = dd.read_parquet(flights_file1)
            df2 = dd.read_parquet(flights_file2)
            df3 = dd.read_parquet(flights_file3)
            df4 = dd.read_parquet(flights_file4)
            df5 = dd.read_parquet(flights_file5)
```

- Recursos del sistema al momento de finalizar la prueba.

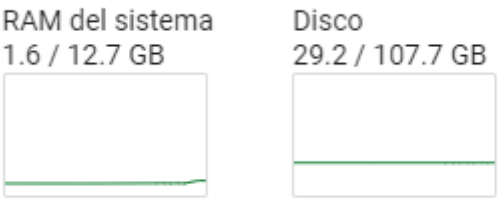


2. En la segunda parte, la concatenación de los cinco (5) archivos se realiza en un tiempo de ejecución de aproximadamente **1 segundo** con un resultado es exitoso. El uso de recursos se ve reflejado en un aumento de la memoria RAM en **0.3 GB** con respecto a la gráfica anterior.

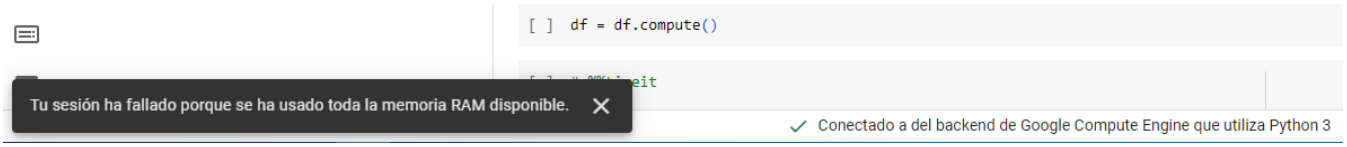
- Realizar la concatenación mediante la siguiente línea

```
[2]  df = dd.concat([df1, df2, df3, df4, df5])
```

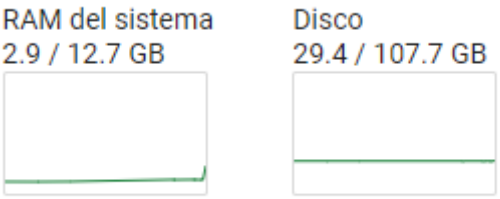
- Recursos del sistema al momento de finalizar la prueba.



3. En la tercera parte, vemos como la ejecución sobrepasa la capacidad de la memoria RAM y termina por matar el proceso.



- Recursos del sistema al momento de iniciar la prueba, se ve inicialmente un pico hasta los **2.9GB** y luego el proceso falla y se detiene la ejecución por falta de recursos.



CONCLUSIONES

El ejercicio realizado nos sirvió para comparar el procesamiento de cada una de las librerías frente a un conjunto de datos establecidos, allí observamos que cada una tiene declaradas sus funciones y métodos de maneras distintas, algunas con un tiempo de respuestas mucho menor con respecto a las anteriores, como en el caso de **PySpark** que en mi opinión y según las evidencias fue quien procesó la información en un menor tiempo, y con un consumo de recursos mucho menor que Pandas por ejemplo. Mientras que **Polars**, fue en mi caso la librería que más tiempo tardó para realizar la concatenación de los archivos, sin contar que en la prueba realizada con Pandas no fue posible concatenar la información sin que se detuviera la ejecución del proceso por falta de recursos de memoria RAM.

Está comparación entre una librería y otra nos sirve como referencia para cuando necesitemos procesar grandes volúmenes de información en un ambiente como Google Colab, independientemente el proyecto que se requiera, puesto que se debe primero identificar cuál de aquellas librerías se ajusta mejor a nuestras necesidades y recursos.