

INFORME LABORATORIO 2

1. LIBRERÍA PANDAS

La idea es cargar la mayor cantidad de archivo y ver cuando puede consumir la RAM.

LIBRERÍA	ARCHIVOS CARGADOS	RAM UTILIZADA
Pandas Prueba 1	Archivo 2 y 3	7.5 RAM / Concatenando Saturado
Pandas Prueba 2	Archivo 4 y 5	6.1 RAM / Concatenando 10.9 RAM
Pandas Prueba 3	Archivo 2 y 5	5.9 RAM / Concatenando 12.1 RAM
Pandas Prueba 4	Archivos 3,4 y 5	10.8 RAM / Concatenando Saturado
Pandas Prueba 5	Archivos 2,3 y 4	Saturado Cargando los Archivos no se alcanza concatenar
Pandas Prueba 6	Archivos 3 y 5	4.8 RAM / Concatenado 9.6 RAM
Pandas Prueba 7	Archivo 2,3,4 y 5	Saturado Cargando los Archivos no se alcanza concatenar
Pandas Prueba 8	Archivos 1 y 2	7.9 RAM / Concatenando Saturado
Pandas Prueba 9	Archivo 1 y 3	6.9 RAM / Concatenado 11.1 AM

Hicimos varias pruebas con la librería de Pandas y podemos apreciar las siguientes conclusiones:

- Se pueden procesar de a dos (2) archivos dependiendo la cantidad de datos que exista en los mismos, como podemos apreciar en la prueba 6 se cargaron 2 archivos 3 y 5 el los cuales consumió 9.6 RAM en total.
- Si Cargamos de ha 3 archivos se satura completamente en la carga de los mismos y dependiendo el peso no se pueden Concatenar.
- Como se puede apreciar en la prueba 2 se cargaron 2 archivos, pero dependiendo la cantidad de datos ah procesar la maquina se saturo al momento de concatenar.

Prueba 2



Prueba 5

```
[ ] df = pd.concat([df2, df3, df4])
# df = df2

[ ] %timeit

df_agg = df.groupby(['Airline', 'Year'])[['DepDelayMinutes', 'ArrD
['mean', 'sum', 'max']]

Tu sesión ha fallado porque se ha usado toda la memoria RAM disponible. X
ackend t
```

Prueba 6



2. LIBRERÍA POLARS

A pesar que con Polars gastamos mucho menos recursos con la maquina puede que sea un poco más complicado la agrupación y concatenación de los datos a diferencia de Pandas.

```
[3] flights_file1 = "/content/drive/MyDrive/bigData/flights/Combined_Flights_2018.parquet"
    flights_file2 = "/content/drive/MyDrive/bigData/flights/Combined_Flights_2019.parquet"
    flights_file3 = "/content/drive/MyDrive/bigData/flights/Combined_Flights_2020.parquet"
    flights_file4 = "/content/drive/MyDrive/bigData/flights/Combined_Flights_2021.parquet"
    flights_file5 = "/content/drive/MyDrive/bigData/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)
```

Con POLARS podemos procesar en 1 solo script los 5 archivos de datos sin problema alguno y solo utilizamos un total de 3.7 de RAM



```
%%timeit
df_polars = (
    pl.concat([df1, df2, df3, df4, df5])
    .group_by(['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')
```

```
[ ] !ls -GFlash temp_polars.parquet
```

```
8.0K -rw-r--r-- 1 root 6.4K Jun 19 20:07 temp_polars.parquet
```

RAM del sistema
3.7 / 12.7 GB

3. LIBRERÍA PYSPARK

Si con Polars podíamos procesar los 5 archivos en uno solo script, en pySpark también podemos hacerlo a diferencia que su script es un poco mas complicado en la llamada de los datos.

```
✓ 10 s [9] spark = SparkSession.builder.master("local[1]").appName("airline-example").getOrCreate()
```

```
✓ 0 s [10] flights_file1 = "/content/drive/MyDrive/bigData/flights/Combined_Flights_2018.parquet"
flights_file2 = "/content/drive/MyDrive/bigData/flights/Combined_Flights_2019.parquet"
flights_file3 = "/content/drive/MyDrive/bigData/flights/Combined_Flights_2020.parquet"
flights_file4 = "/content/drive/MyDrive/bigData/flights/Combined_Flights_2021.parquet"
flights_file5 = "/content/drive/MyDrive/bigData/flights/Combined_Flights_2022.parquet"
```

```
✓ 8 s [11] 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)
```

```
✓ 1 s [12] 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)
```

Si con Polars usamos 3.7 de RAM en recursos de la maquina con pySpark usamos solo 1.9 de RAM de hecho no utilizamos casi recursos, pero si debemos hacer más largo el script.

```
✓ 16 s [13] # %%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')
```

¿Quieres más memori

del backend de Google
Mostrando recursos d

RAM del sistema
1.9 / 12.7 GB



4. LIBRERÍA DASK

Con dask utilizamos la misma estructura de Pandas a diferencia que importamos un nuevo parámetro llamado **dask.dataframe** para ellos haremos las mismas pruebas que hicimos con la librería pandas a ver si evidenciamos algún cambio en los recursos de la máquina.

LIBRERÍA	CARGA DE ARCHIVOS	RAM UTILIZADA
Pandas Prueba 1	Archivo 2 y 3	
Pandas Prueba 2	Archivo 4 y 5	Aunque esta vez si pudo concatenar, tuvo un pico de 10.6 de RAM para luego bajar a 1.4 de RAM, puede llegar a colgarse
Pandas Prueba 3	Archivo 2 y 5	
Pandas Prueba 4	Archivos 3,4 y 5	
Pandas Prueba 5	Archivos 2,3 y 4	Saturado Completamente
Pandas Prueba 6	Archivos 3 y 5	Mejoro considerablemente de 9.6 de RAM paso solo ha utilizar 2.2 de RAM
Pandas Prueba 7	Archivo 2,3,4 y 5	

Prueba 6

0 s

```
import pandas as pd
import dask.dataframe as dd

# flights_file1 = "/content/drive/MyDrive/bigData/flights/Combined_Flights_2018.parquet"
# flights_file2 = "/content/drive/MyDrive/bigData/flights/Combined_Flights_2019.parquet"
flights_file3 = "/content/drive/MyDrive/bigData/flights/Combined_Flights_2020.parquet"
# flights_file4 = "/content/drive/MyDrive/bigData/flights/Combined_Flights_2021.parquet"
flights_file5 = "/content/drive/MyDrive/bigData/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)
```

[16]

df = dd.concat([df3, df5])

[17]

print(df.compute())

	FlightDate	Airline	Origin	Dest	Cancelled	Diverted	\
0	2020-09-01	Comair Inc.	PHL	DAY	False	False	
1	2020-09-02	Comair Inc.	PHL	DAY	False	False	
2	2020-09-03	Comair Inc.	PHL	DAY	False	False	
3	2020-09-04	Comair Inc.	PHL	DAY	False	False	

Recursos

No te has suscrito. M
En estos momentos, disponible. Los recu
garantizados. Puede
Con tu nivel de uso e
hasta 82 horas 30 m

¿Quieres más mem
del backend de Goo
Mostrando recursos

RAM del sistema
2.2 / 12.7 GB

Prueba 2

1 s

```
[1] import pandas as pd
import dask.dataframe as dd
# flights_file1 = "/content/drive/MyDrive/bigData/flights/Combined_Flights_2018.parquet"
# flights_file2 = "/content/drive/MyDrive/bigData/flights/Combined_Flights_2019.parquet"
# flights_file3 = "/content/drive/MyDrive/bigData/flights/Combined_Flights_2020.parquet"
flights_file4 = "/content/drive/MyDrive/bigData/flights/Combined_Flights_2021.parquet"
flights_file5 = "/content/drive/MyDrive/bigData/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)
```

[2]

df = dd.concat([df4, df5])

print(df.compute())

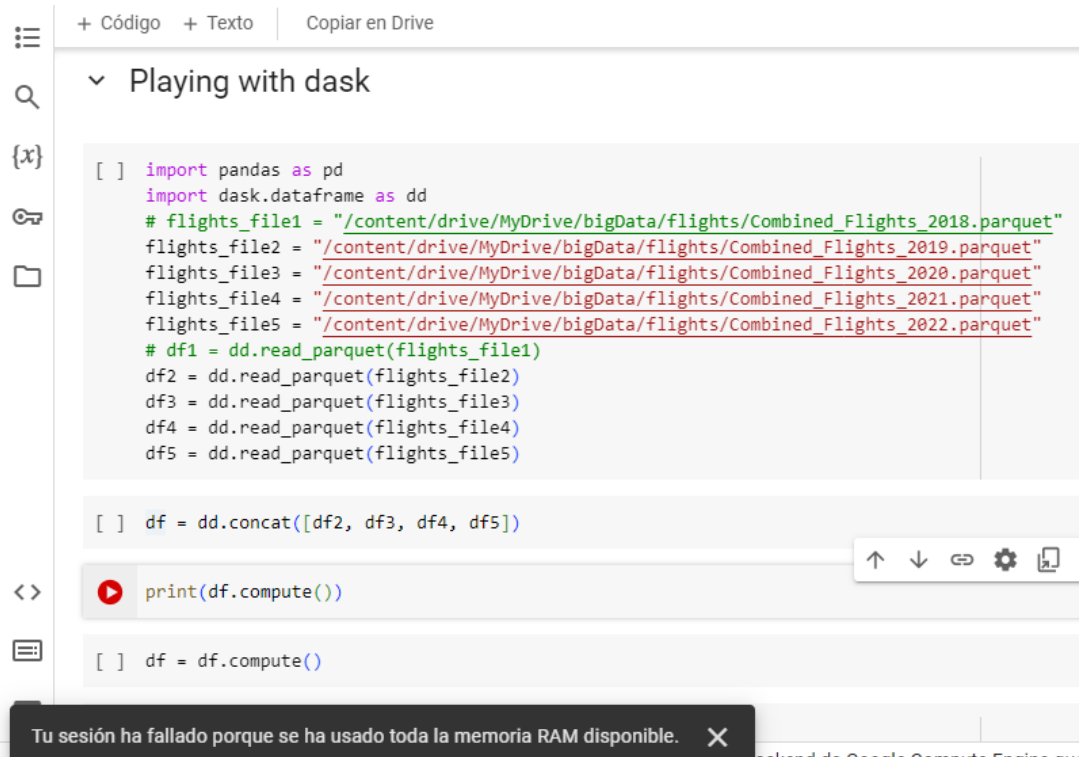
	FlightDate	Airline	Origin	Dest	Cancelled	Diverted	\
0	2021-03-03	SkyWest Airlines Inc.	SGU	PHX	False	False	

Podemos evidenciar que se ve un pico altísimo en recursos, pero termina bajando nuevamente al 1.4

RAM del sistema
9.1 / 12.7 GB

Terminamos de hacer las pruebas, pero nos enfocamos mas en las 3 pruebas en las que sacamos conclusiones en la librería de Pandas, que en algunas si hubo cierto cambio mas que todo en la carga y en la concatenación de los datos en este caso utilizar dask favoreció mas que utilizar solo pandas.

Con dask cargar los archivos y concatenarlos es mas eficiente, pero al momento de procesarlos aun sigue gastando demasiado recursos de la maquina



The screenshot shows a Google Colab notebook interface. At the top, there are tabs for '+ Código', '+ Texto', and 'Copiar en Drive'. The notebook title is 'Playing with dask'. The code is written in a light gray editor with syntax highlighting. It imports 'pandas as pd' and 'dask.dataframe as dd'. It then defines five file paths for flight data from 2018 to 2022. The code reads each file into a Dask DataFrame (df1 to df5) using 'dd.read_parquet'. It concatenates df2, df3, df4, and df5 into a new Dask DataFrame 'df' using 'dd.concat'. Finally, it prints 'df.compute()' and then 'df = df.compute()'. A black error banner at the bottom states: 'Tu sesión ha fallado porque se ha usado toda la memoria RAM disponible.' (Your session has failed because all available RAM has been used).

```
[ ] import pandas as pd
import dask.dataframe as dd
# flights_file1 = "/content/drive/MyDrive/bigData/flights/Combined_Flights_2018.parquet"
flights_file2 = "/content/drive/MyDrive/bigData/flights/Combined_Flights_2019.parquet"
flights_file3 = "/content/drive/MyDrive/bigData/flights/Combined_Flights_2020.parquet"
flights_file4 = "/content/drive/MyDrive/bigData/flights/Combined_Flights_2021.parquet"
flights_file5 = "/content/drive/MyDrive/bigData/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)

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

[ ] print(df.compute())

[ ] df = df.compute()
```

Tu sesión ha fallado porque se ha usado toda la memoria RAM disponible. X

5. CONCLUSIÓN

Después de hacer este experimento comparando las librerías de Python para cargar datos sería más eficiente utilizar POLARS ya que podemos procesar los 5 archivos al mismo tiempo utilizando un poco de RAM y a pesar que su script es un poco largo, pero no es tan largo como el de pySpark.