Kevin Steven Olivera Caicedo (64178)

Grupo 2

RESULTADOS – TALLER CARGA DE DATOS GOOGLE COLAB

Se mostrarán los resultados de carga de información en Google Colab, con 5 archivos que contienen la cantidad necesaria de datos. Esto puede implicar el consumo de recursos de la herramienta (Memoria RAM), seguido por la concatenación de dichos archivos

1. **LIBRERÍA PANDAS**

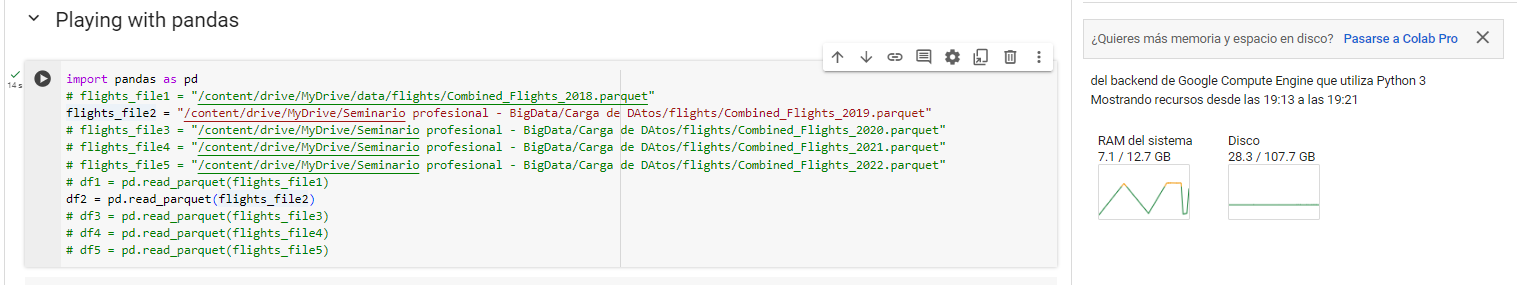
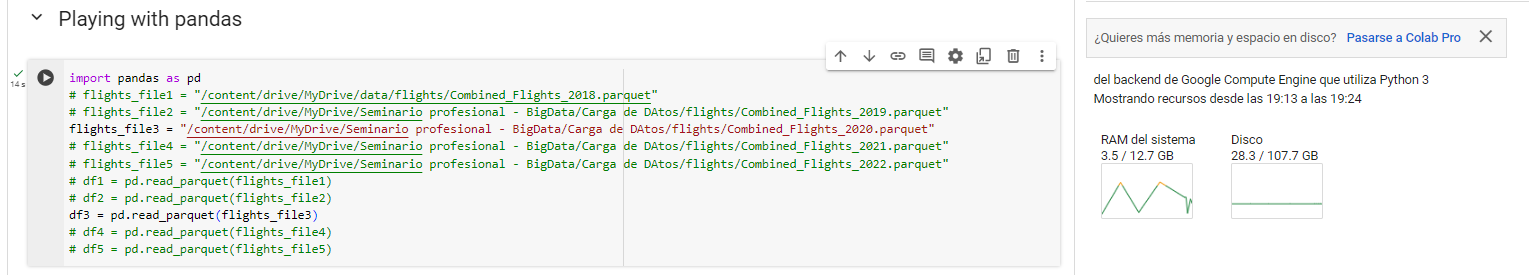
En la primera carga de los archivos, estos de ejecutaron 1 por 1, el cual se recopilaron el consumo de recursos y el tiempo con el que se demora en ejecutarlos.

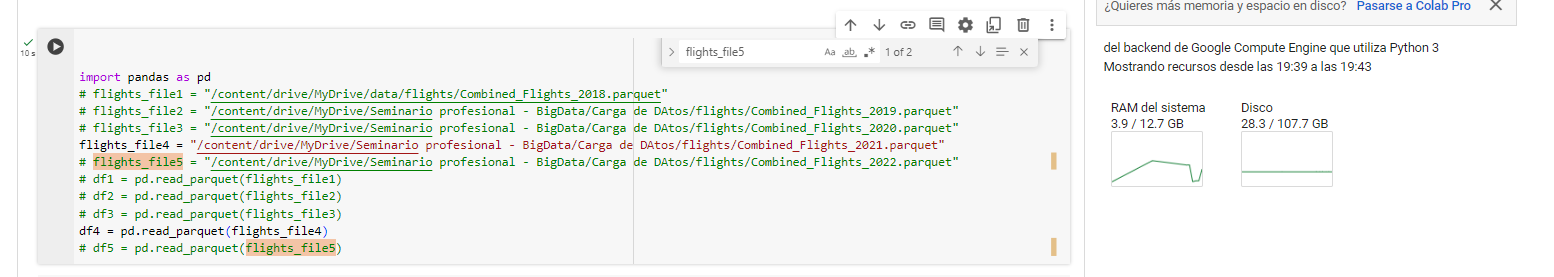
Los resultados de las primeras cargas se representan en la siguiente Tabla:

Tabla 1.

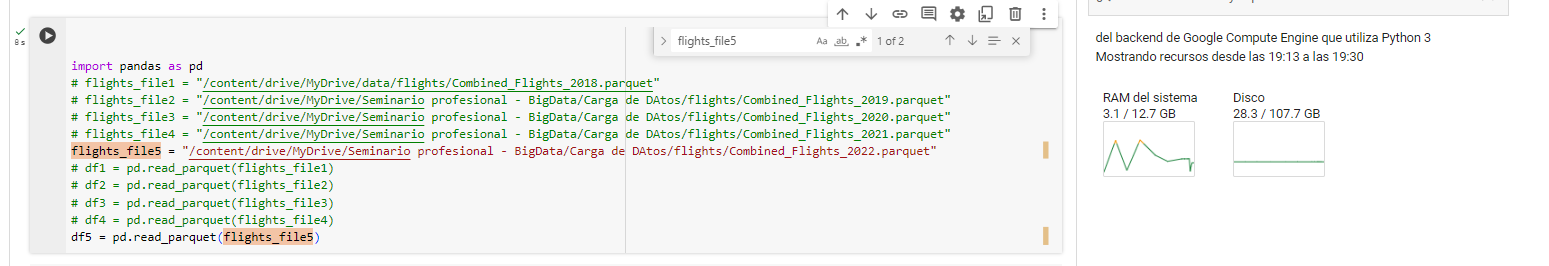
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **CAT** | **TIEMPO** | **RAM** | **DISCO** | **Status** |
| 2019 | 14 | 7,1 | 28,3 |  |
| 2020 | 14 | 4,5 | 28,3 |  |
| 2021 | 11 | 5,3 | 28,3 |  |
| 2022 | 8 | 3,1 | 28,3 |  |

Los resultados de cada conjunto de datos son:

* 2019:  
    
  
* 2020:  
    
  
* 2021



* 2022

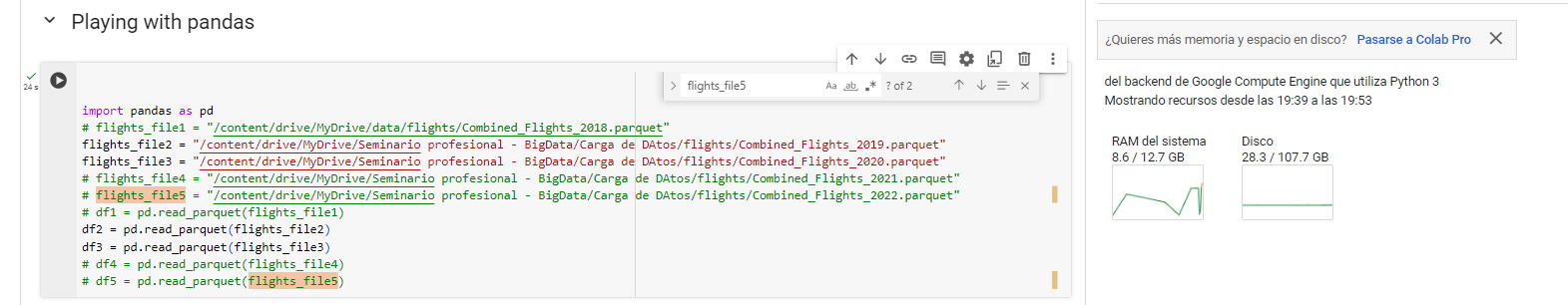


Se identifica que para que el sistema logre ejecutar correctamente el código con los datos suministrados, este tiene que reiniciarse por completo el sistema, para que este pueda ejecutar correctamente cada uno de ellos.

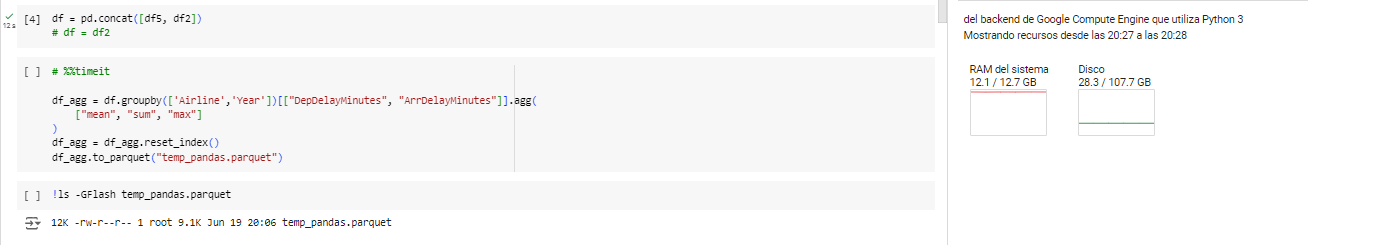
Se realizaron las pruebas con respecto a ejecutar mas de un conjunto de datos, con estas librerías, por lo que se identifico que el sistema se limita con respecto a la memoria RAM, ya que son una cantidad muy grande de archivos, provocando que se supere el limite de uso de la RAM del sistema, provocando un colapso, como se evidencia en la siguiente tabla, junto con las pruebas realizadas:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Pruobados** | **Tiempo Seg** | **12,7 GB RAM** | **107,7 GB Disco** | **Concatenado** |
| 2019 - 2020 | Colapse | Colapse | Colapse |  |
| 2020 - 2021 | 12 | 12,1 | 28,3 |  |
| 2021 - 2022 | 10 | 9,4 | 28,3 |  |
| 2022 - 2019 | 12 | 12,2 | 28,3 |  |

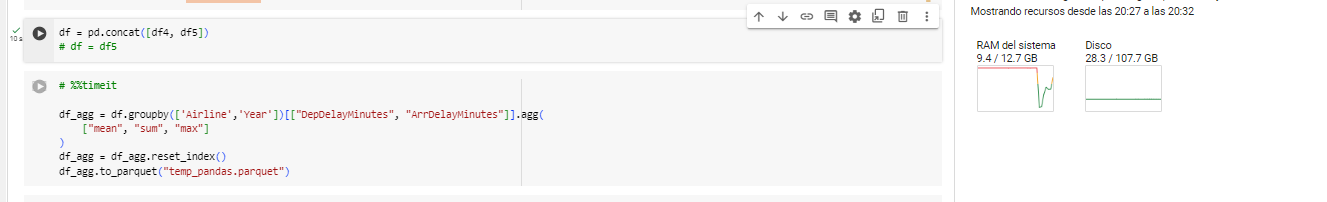
* 2019 – 2020



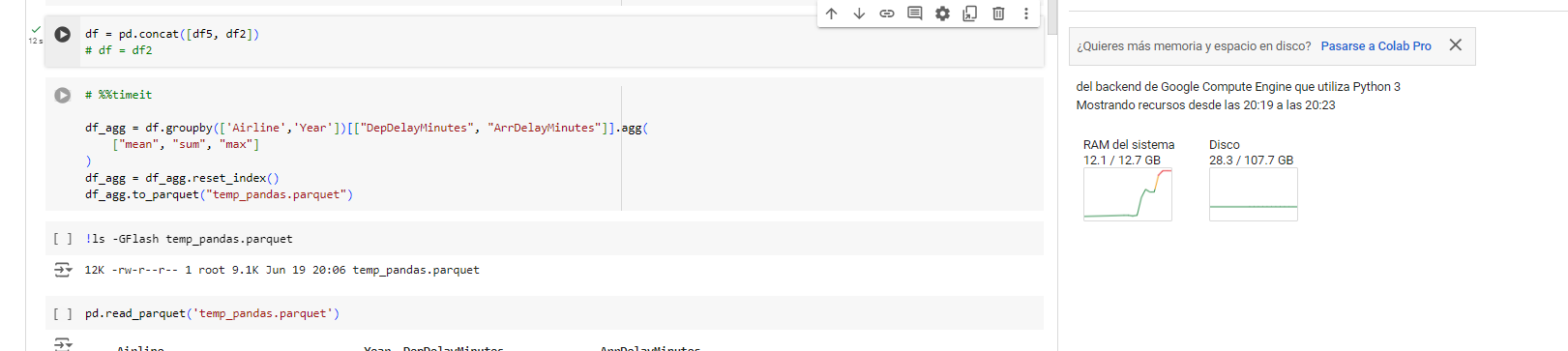
* 2020 – 2021



* 2021 – 2022



* 2022 – 2019



El código es capaz de mover una cantidad grande de archivos sin problemas, solo que hay que contar con mas capacidad de hardware, para que estos puedan ejecutarse correctamente, y mas cuándos e trata de la memoria RAM.

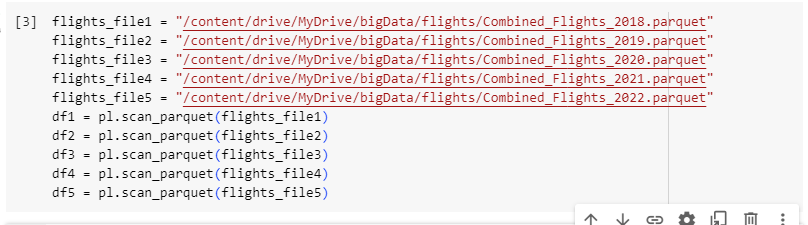
Posteriormente se realizo otra prueba para validar los datos de ejecución y consumo como se muestra en la siguiente tabla:

|  |  |  |  |
| --- | --- | --- | --- |
| **Pruobados** | **Tiempo Seg** | **12,7 GB RAM** | **107,7 GB Disco** |
| 2019 - 2020 | Colapse | Colapse | Colapse |
| 2020 - 2021 | 12 | 12,1 | 28,3 |
| 2021 - 2022 | 10 | 9,4 | 28,3 |
| 2022 - 2019 | 12 | 12,2 | 28,3 |

Se identifica, que de igual manera consume prácticamente la misma cantidad de recurso del sistema.

1. LIBRERÍA POLARS

Aunque Polars consume significativamente menos recursos, la agrupación y concatenación de datos puede resultar un poco más complicada en comparación con Pandas."



Con Polars, podemos procesar los cinco archivos de datos en un solo script sin ningún inconveniente, utilizando solo 3.7 GB de RAM en total."

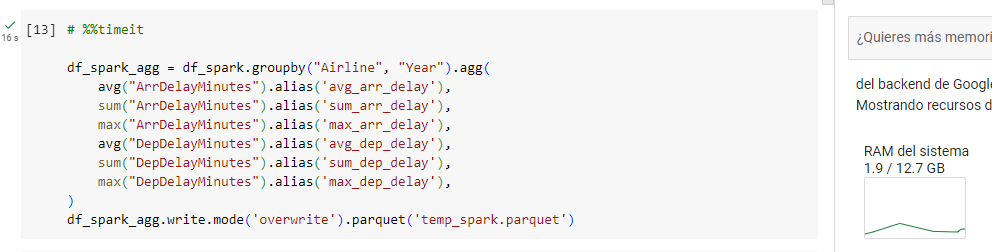


1. LIBRERÍA PYSPARK

Si bien con Polars podemos procesar los cinco archivos en un solo script, con PySpark también es posible hacerlo, aunque su script se complica un poco más en la llamada de los datos.



Utilizando Polars, consumimos 3.7 GB de RAM en recursos de la máquina, mientras que con PySpark solo utilizamos 1.9 GB, lo que representa un uso considerablemente menor de recursos, aunque el script puede volverse más extenso.



1. 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 |
| Dask Prueba 1 | Archivo 2 y 3 |  |
| Pandas Prueba 2 | Archivo 4 y 5 | Esta vez pudo concatenar, pero hubo un pico de consumo de RAM de 10.6 GB antes de bajar a 1.4 GB, con riesgo de colgarse |
| Dask Prueba 3 | Archivo 2 y 5 | Colapsado |
| Dask Prueba 4 | Archivos 3,4 y 5 | Colapsado |
| Dask Prueba 5 | Archivos 2,3 y 4 | Colapsado |
| Dask Prueba 6 | Archivos 3 y 5 | Mejoras de 9.6 de RAM a 2.2 de RAM |
| Dask Prueba 7 | Archivo 2,3,4 y 5 | Colapsado |

CONCLUSIÓN

Tras llevar a cabo este experimento comparativo entre las bibliotecas de Python para la carga de datos, concluimos que utilizar Polars sería más eficiente. Con Polars, podemos procesar simultáneamente los cinco archivos utilizando una cantidad moderada de RAM. Aunque su script es un poco extenso, no alcanza la complejidad del script en PySpark.

Esta elección se fundamenta en la capacidad de Polars para manejar eficientemente grandes volúmenes de datos con un consumo de recursos relativamente bajo, lo cual resulta crucial para aplicaciones que requieren procesamiento rápido y eficiente, y mas cuando se trata de BigData.