



Parallel and concurrent programming

Machine Learning usando RAPIDS y PANDAS-Colab

DOCENTE: JOSE ALFREDO HERRERA QUISPE

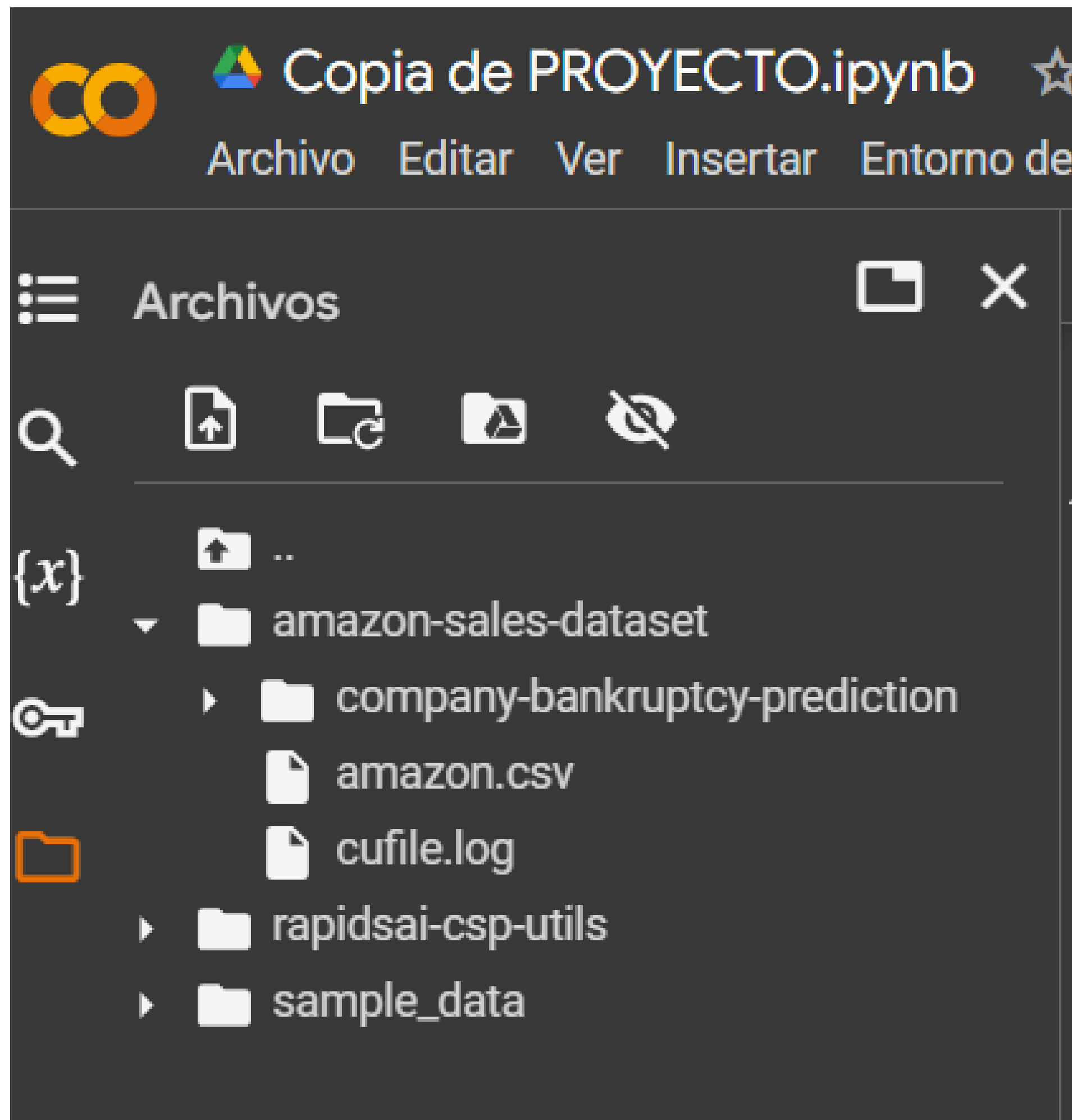
ALUMNO: FABRICIO MARIANO CRESPO MIRANDA



**DATASET
UTILIZADO**



**DEPENDENCIAS
RAPIDS**



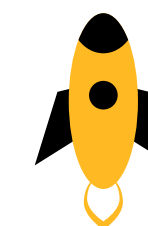

```
✓ 7s [3] !pip install opendatasets
import opendatasets as op

Collecting opendatasets
  Downloading opendatasets-0.1.22-py3-none-any.whl (15 kB)
Requirement already satisfied: tqdm in /usr/local/lib/python3.10/dist-packages (from opendatasets) (4.
Requirement already satisfied: kaggle in /usr/local/lib/python3.10/dist-packages (from opendatasets) (
Requirement already satisfied: click in /usr/local/lib/python3.10/dist-packages (from opendatasets) (8
Requirement already satisfied: six>=1.10 in /usr/local/lib/python3.10/dist-packages (from kaggle->open
Requirement already satisfied: certifi in /usr/local/lib/python3.10/dist-packages (from kaggle->openda
Requirement already satisfied: python-dateutil in /usr/local/lib/python3.10/dist-packages (from kaggle
Requirement already satisfied: requests in /usr/local/lib/python3.10/dist-packages (from kaggle->opend
Requirement already satisfied: python-slugify in /usr/local/lib/python3.10/dist-packages (from kaggle-
Requirement already satisfied: urllib3 in /usr/local/lib/python3.10/dist-packages (from kaggle->openda
Requirement already satisfied: bleach in /usr/local/lib/python3.10/dist-packages (from kaggle->opendat
Requirement already satisfied: webencodings in /usr/local/lib/python3.10/dist-packages (from bleach->k
Requirement already satisfied: text-unidecode>=1.3 in /usr/local/lib/python3.10/dist-packages (from py
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.10/dist-packages (fr
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-packages (from requests-
Installing collected packages: opendatasets
Successfully installed opendatasets-0.1.22
```

```
✓ 16s [26] #otro metodo para la carga de la data muy pesada
#username: fabrricciocrespo
#key : 4037245f953849db4a1618c83541317f
dataset_tarea="https://www.kaggle.com/datasets/fedesoriano/company-bankruptcy-prediction"
op.download(dataset_tarea)
```



HABILITAR UN ESPACIO PARA EL
DATASET DE KAGGLE



Metas

SUBIR EL DATA SET A GOOGLE COLAB A
PARTIR DEL LINK DE KAGGLE



CONTEXTO DEL DATASET



Los datos se obtuvieron del Taiwan Economic Journal para los años 1999 a 2009. La quiebra de una empresa se definió con base en las regulaciones comerciales de la Bolsa de Valores de Taiwán.



FUENTE ORIGINAL

Deron Liang y Chih-Fong Tsai, deronliang '@' gmail.com; cftsai '@' mgt.ncu.edu.tw, Universidad Nacional Central, Taiwán

Los datos se obtuvieron del repositorio de aprendizaje automático de UCI:

<https://archive.ics.uci.edu/ml/datasets/Taiwanese+Bankruptcy+Prediction>

```
dataset_tarea="https://www.kaggle.com/datasets/fedesoriano/company-bankruptcy-prediction"  
op.download(dataset_tarea)
```

➞ Please provide your Kaggle credentials to download this dataset. Learn more: <http://bit.ly/kaggle-creds>
Your Kaggle username: fabrricciocrespo
Your Kaggle Key:
Downloading company-bankruptcy-prediction.zip to ./company-bankruptcy-prediction
100%|██████████| 4.63M/4.63M [00:01<00:00, 3.74MB/s]

✓
0 s [27] `import os`
`os.chdir("company-bankruptcy-prediction")`
`os.listdir()`

`['data.csv']`

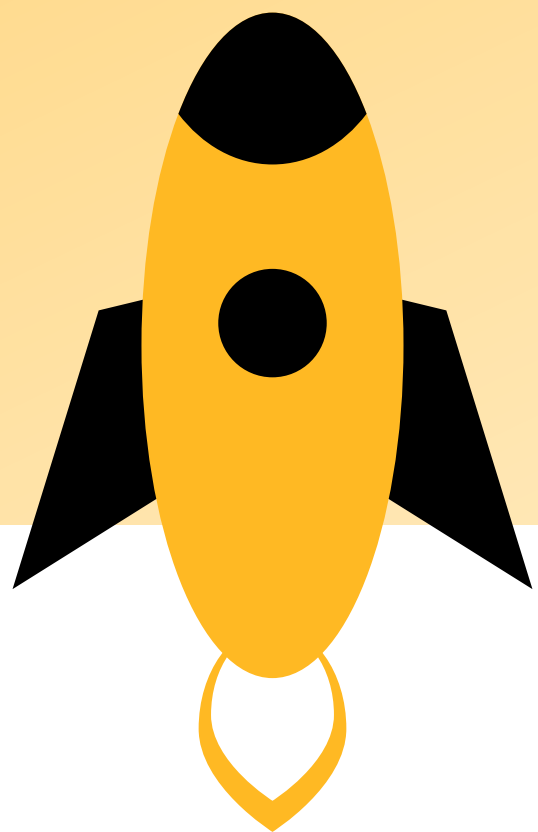
✓ [6] `import pandas as pd`
`import cudf as cd`



¡IMPORTAMOS PANDAS PARA EL ANALISIS
SECUENCIAL Y CUDF PARA EL ANALISIS
PARALELO



Presenta con facilidad y sorprende a cualquier público con las
presentaciones de Canva. Elige entre más de mil plantillas
profesionales adaptadas para cualquier objetivo o tema.



```
[28] #Mostramos los Datos del Dataset(Secuencial)
      archivo="data.csv"

      start_time = time.time()
      datas=pd.read_csv(archivo)
      data_load_times = time.time() - start_time

      %time dataS=pd.read_csv(archivo)
```

```
CPU times: user 173 ms, sys: 6.09 ms, total: 179 ms
Wall time: 172 ms
```

```
[ ] pd.read_csv(archivo)
```

```
[29] #Mostramos los Datos del Dataset(Paralelo)
      archivo="data.csv"
      start_time = time.time()
      datap=cd.read_csv(archivo)
      data_load_timeP = time.time() - start_time
      #tiempo usando paralelismo
      %time dataP=cd.read_csv(archivo)
```

```
CPU times: user 18.6 ms, sys: 1.23 ms, total: 19.8 ms
Wall time: 37.7 ms
```



MUESTREO DE DATOS

[30]

cd.read_csv(archivo)

	Bankrupt?	ROA(C) before interest and depreciation before interest	ROA(A) before interest and % after tax	ROA(B) before interest and depreciation after tax	Operating Gross Margin	Realized Sales Gross Margin	Operating Profit Rate	Pre-tax net Interest Rate	After- tax net Interest Rate	Non-industry income and expenditure/revenue	...	Net Income to Total Assets	
0	1	0.370594	0.424389	0.405750	0.601457	0.601457	0.998969	0.796887	0.808809	0.302646	...	0.716845	0
1	1	0.464291	0.538214	0.516730	0.610235	0.610235	0.998946	0.797380	0.809301	0.303556	...	0.795297	0
2	1	0.426071	0.499019	0.472295	0.601450	0.601364	0.998857	0.796403	0.808388	0.302035	...	0.774670	0
3	1	0.399844	0.451265	0.457733	0.583541	0.583541	0.998700	0.796967	0.808966	0.303350	...	0.739555	0
4	1	0.465022	0.538432	0.522298	0.598783	0.598783	0.998973	0.797366	0.809304	0.303475	...	0.795016	0
...
6814	0	0.493687	0.539468	0.543230	0.604455	0.604462	0.998992	0.797409	0.809331	0.303510	...	0.799927	0
6815	0	0.475162	0.538269	0.524172	0.598308	0.598308	0.998992	0.797414	0.809327	0.303520	...	0.799748	0
6816	0	0.472725	0.533744	0.520638	0.610444	0.610213	0.998984	0.797401	0.809317	0.303512	...	0.797778	0
6817	0	0.506264	0.559911	0.554045	0.607850	0.607850	0.999074	0.797500	0.809399	0.303498	...	0.811808	0
6818	0	0.493053	0.570105	0.549548	0.627409	0.627409	0.998080	0.801987	0.813800	0.313415	...	0.815956	0

6819 rows × 96 columns

Separar características y etiquetas

Primero, elimina la columna 'Operating Gross Margin' del conjunto de datos para obtener las características x. Luego, extrae esta columna como la etiqueta y. Después de cada operación, imprime el resultado y registra el tiempo de ejecución correspondiente. El objetivo es medir el rendimiento de la separación de datos en un entorno secuencial, del mismo modo en paralelo.

```
start_time = time.time()
x = datas.drop(' Operating Gross Margin', axis=1)
y = datas[' Operating Gross Margin']
data_separations = time.time() - start_time

print("TIEMPO SECUENCIAL:::::")
%time print(x)
```


Dividir el conjunto de datos en entrenamiento y prueba(PARALELO)

```
start_time = time.time()
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2, random_state=42)
data_split_timeP = time.time() - start_time


div=X_train, X_test, Y_train, Y_test
%time print(div)
```

(Bankrupt?	ROA(C) before interest and depreciation before interest \
3759	0	0.498513
1782	0	0.506606
5013	0	0.508799
5412	0	0.499976
3066	0	0.477892
...
3772	0	0.542729
5191	0	0.550870
5226	0	0.507288
5390	0	0.581339

```

✓ 0s [63] # Inicializa y entrena el modelo de regresión lineal Secue
start_time = time.time()
model = LinearRegression()
model.fit(x_train, y_train)
data_split_regres = time.time() - start_time
model

```

 LinearRegression
 LinearRegression()

```

✓ 0s [64] # Realiza predicciones en el conjunto de prueba
start_time = time.time()
predictions = model.predict(x_test)
data_split_predicS = time.time() - start_time
predictions

array([0.59633265, 0.60341562, 0.61056362, ..., 0.60670367,
       0.63338642])

```

```

✓ 0s [65] # Calcula el error cuadrático medio
start_time = time.time()
mse = mean_squared_error(y_test, predictions)
data_split_errors = time.time() - start_time
print("Error Cuadrático Medio:", mse)

Error Cuadrático Medio: 4060337.065416138

```

```

✓ 0s [79] # Inicializa y entrena el modelo de regresión lineal PARALELO
X_train_ = X_train.to_numpy()
Y_train_ = Y_train.to_numpy()
start_time = time.time()
modelp = LinearRegression()
modelp.fit(X_train_, Y_train_)
data_split_regreP = time.time() - start_time
model
%time modelp.fit(X_train_, Y_train_)

```

CPU times: user 66.7 ms, sys: 0 ns, total: 66.7 ms
 Wall time: 77.7 ms

LinearRegression
 LinearRegression()

```

✓ 0s [84] # Realiza predicciones en el conjunto de prueba
X_test_ = X_test.to_numpy()
start_time = time.time()
predictions = modelp.predict(X_test_)
data_split_predicP = time.time() - start_time
predictions

array([0.5963326 , 0.60341559, 0.61056364, ..., 0.60670375, 0.59888329,
       0.63338654])


```


```

✓ 0s [86] # Calcula el error cuadrático medio
Y_test_ = Y_test.to_numpy()
start_time = time.time()
mse = mean_squared_error(Y_test_, predictions)
data_split_errorP = time.time() - start_time
print("Error Cuadrático Medio:", mse)

Error Cuadrático Medio: 3962338.522918427

```


 Primero, inicializa y entrena el modelo utilizando datos de entrenamiento. Luego, hace predicciones en un conjunto de prueba utilizando el modelo entrenado. Finalmente, calcula el error cuadrático medio entre las predicciones y las etiquetas reales.


 Primero, convierte los datos de entrenamiento (X_train y Y_train) de cuDF DataFrame a matrices NumPy. Inicializa y entrena el modelo utilizando datos de entrenamiento. Luego, hace predicciones en un conjunto de prueba utilizando el modelo entrenado. Finalmente, calcula el error cuadrático medio entre las predicciones y las etiquetas reales.

➤ LATENCIA

La latencia se refiere al tiempo que tarda una operación individual en completarse desde el inicio hasta el final.

Latencia = Tiempo de finalización -
Tiempo de inicio

```
[87] # Calcular latencia
print("Tiempos Secuenciales::::::::::::")
print("Tiempo de carga de datos:", data_load_timeS)
print("Tiempo de separacion de datos:", data_separationS)
print("Tiempo de división de datos:", data_split_timeS)
print("Tiempo de entrenamiento:", data_split_regreS)
print("Tiempo de predicción de conjuntos de prueba:", data_split_predicS)
print("Tiempo de error cuadrático medio:", data_split_errorS)
print("Tiempos Paralelos::::::::::::")
print("Tiempo de carga de datos:", data_load_timeP)
print("Tiempo de separacion de datos:", data_separationP)
print("Tiempo de división de datos:", data_split_timeP)
print("Tiempo de entrenamiento:", data_split_regreP)
print("Tiempo de predicción de conjuntos de prueba:", data_split_predicP)
print("Tiempo de error cuadrático medio:", data_split_errorP)
```

```
Tiempos Secuenciales::::::::::::
Tiempo de carga de datos: 0.16738390922546387
Tiempo de separacion de datos: 0.01849055290222168
Tiempo de división de datos: 0.007200002670288086
Tiempo de entrenamiento: 0.04433441162109375
Tiempo de predicción de conjuntos de prueba: 0.0077168941497802734
Tiempo de error cuadrático medio: 0.0010421276092529297
Tiempos Paralelos::::::::::::
Tiempo de carga de datos: 0.04528951644897461
Tiempo de separacion de datos: 0.01141047477722168
Tiempo de división de datos: 0.019708633422851562
Tiempo de entrenamiento: 0.12764835357666016
Tiempo de predicción de conjuntos de prueba: 0.0006895065307617188
Tiempo de error cuadrático medio: 0.0007891654968261719
```

➤ throughput (RENDIMIENTO)

Se refiere a la cantidad de trabajo que se puede completar en un período de tiempo específico.

Throughput = Cantidad de trabajo
/ Tiempo transcurrido



```
# Calcular throughput
print("Tiempos Secuenciales::::::::::::")
total_operations = len(x) # Suponiendo que cada instancia se procesa como
total_time = data_load_timeS + data_separationS + data_split_timeS + data_
throughput = total_operations / total_time
print("Throughput:", throughput)
print("Tiempos Paralelos::::::::::::")
total_operations = len(X)
total_time = data_load_timeP + data_separationP + data_split_timeP + data_
throughput = total_operations / total_time
print("Throughput:", throughput)
```

```
Tiempos Secuenciales::::::::::::
Throughput: 27700.606173541386
Tiempos Paralelos::::::::::::
Throughput: 33176.72623506662
```




SPEEDUP

El speedup es una medida de la mejora en el rendimiento que se obtiene al utilizar paralelismo para realizar una tarea en comparación con realizar la misma tarea de forma secuencial.

$$\text{Speedup} = \frac{\text{Tiempo de ejecución secuencial}}{\text{Tiempo de ejecución paralelo}}$$

```
[89] # Calcular speedup
print("Tiempos Secuenciales:::::::::")
baseline_time = data_load_timeS
speedup = baseline_time / total_time
print("Speedup data_load:", speedup)
baseline_time = data_split_timeS
speedup = baseline_time / total_time
print("Speedup data_split:", speedup)
baseline_time = data_separationS
speedup = baseline_time / total_time
print("Speedup data_separation:", speedup)
baseline_time = data_split_predictS
speedup = baseline_time / total_time
print("Speedup data_split_predict:", speedup)
print("\nTiempos Paralelos:::::::::")
baseline_time = data_load_timeP
speedup = baseline_time / total_time
print("Speedup data_load:", speedup)
baseline_time = data_split_timeP
speedup = baseline_time / total_time
print("Speedup data_split:", speedup)
baseline_time = data_separationP
speedup = baseline_time / total_time
print("Speedup data_separation:", speedup)
baseline_time = data_split_predictP
speedup = baseline_time / total_time
print("Speedup data_split_predict:", speedup)
```

```
Tiempos Secuenciales:::::::::
Speedup data_load: 0.8143789606288983
Speedup data_split: 0.03503043224576866
Speedup data_separation: 0.08996275283355702
Speedup data_split_predict: 0.03754528297290619
```

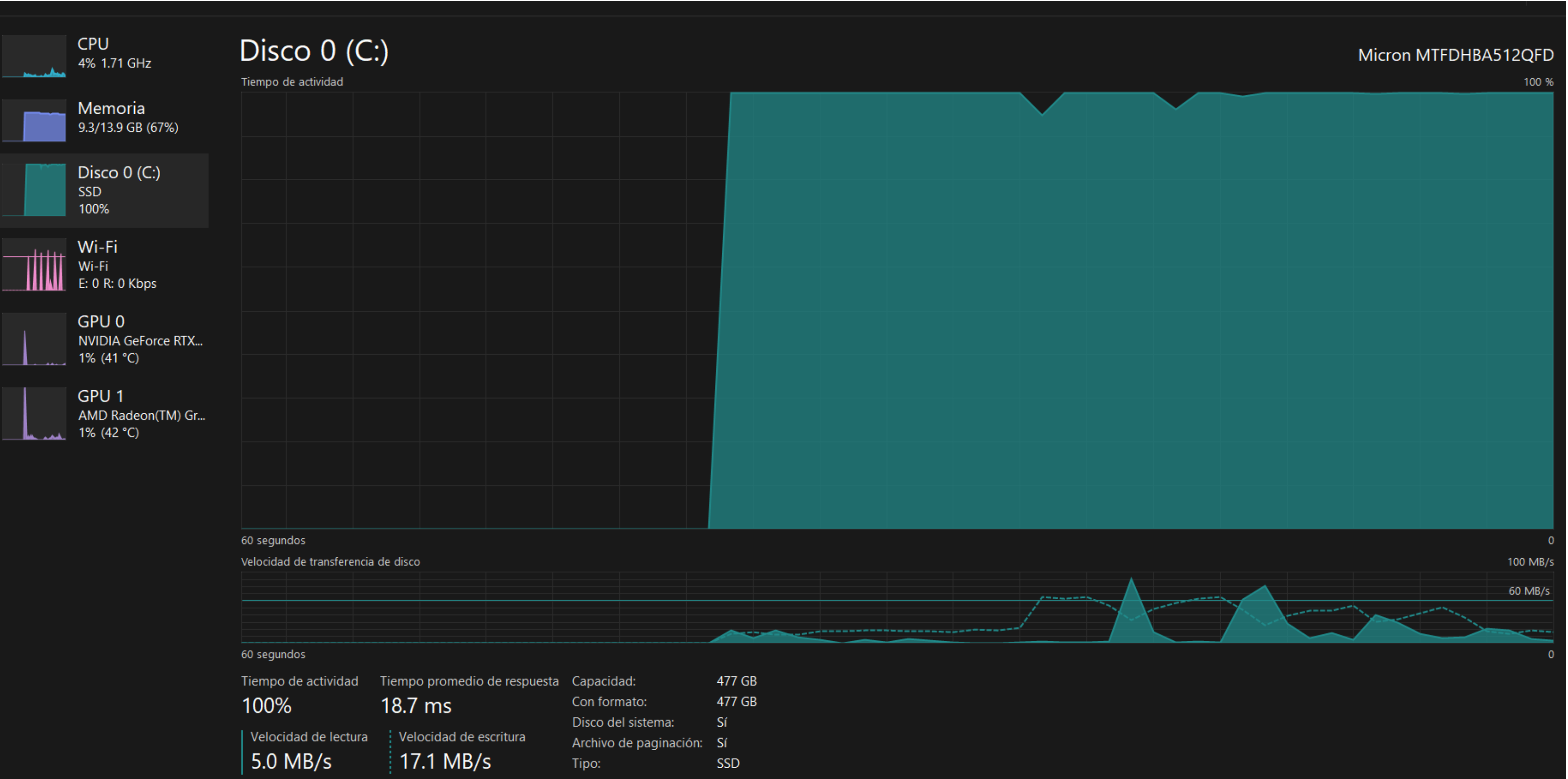
```
Tiempos Paralelos:::::::::
Speedup data_load: 0.22034871514095575
Speedup data_split: 0.09588912385059838
Speedup data_separation: 0.055515793796160215
Speedup data_split_predict: 0.0033546809515137242
```



LIMITACIONES



DATASET DE 1 MILLON DE REGISTROS





ALGORITMOS PESADOS

[29]: *# Inicializa y entrena el modelo de regresión lineal*

```
model = LinearRegression()  
model.fit(X_train, y_train)
```

ValueError Traceback (most recent call last)

~\AppData\Local\Temp\ipykernel_2836\2292996676.py in ?()

1 *# Inicializa y entrena el modelo de regresión lineal*

2 model = LinearRegression()

----> 3 model.fit(X_train, y_train)

~\AppData\Local\Programs\Python\Python312\Lib\site-packages\sklearn\base.py in ?(estimator, *args, **kwargs)

1470 skip_parameter_validation=(

1471 prefer_skip_nested_validation or global_skip_validation

1472)

1473):

-> 1474 return fit_method(estimator, *args, **kwargs)

~\AppData\Local\Programs\Python\Python312\Lib\site-packages\sklearn\linear_model_base.py in ?(self, X, y, sample_weight)

574 n_jobs_ = self.n_jobs

575

576 accept_sparse = False if self.positive else ["csr", "csc", "coo"]

577

--> 578 X, y = self._validate_data(

579 X, y, accept_sparse=accept_sparse, y_numeric=True, multi_output=True

580)

581

~\AppData\Local\Programs\Python\Python312\Lib\site-packages\sklearn\base.py in ?(self, X, y, reset, validate_separately, cast_to_ndarray, **check_params)

646 if "estimator" not in check_y_params:

647 check_y_params = {**default_check_params, **check_y_params}

648 y = check_array(y, input_name="y", **check_y_params)

649 else:

--> 650 X, y = check_X_y(X, y, **check_params)

651 out = X, y

652


653 if not no_val_X and check_params.get("ensure_2d", True):

~\AppData\Local\Programs\Python\Python312\Lib\site-packages\sklearn\utils\validation.py in ?(X, y, accept_sparse, accept_large_sparse, dtype, order, copy, force_all_finite, ensure_2d, allow_nd, estimator)

1259 raise ValueError(


1260 f"{estimator_name} requires y to be passed, but the target y is None"

1261)



CPU
11% 1.80 GHz

Disco 0 (C:) SSD 3%



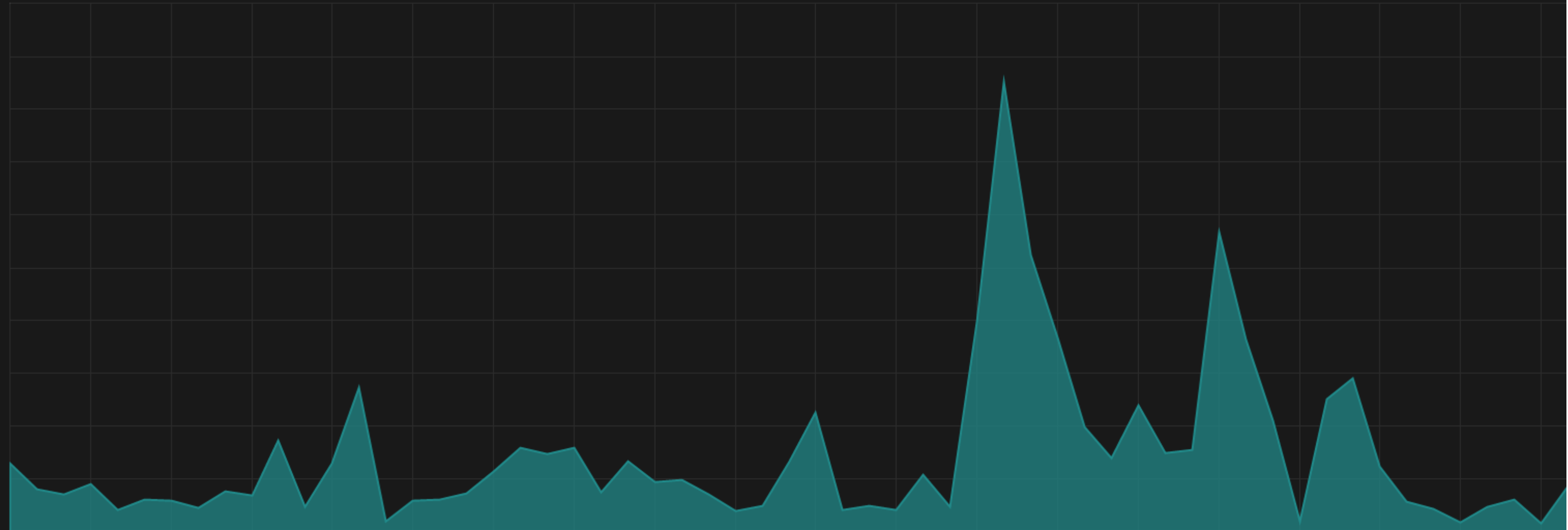
Wi-Fi
Wi-Fi
E: 0 R: 8.0 Kbps

GPU 0
NVIDIA GeForce RTX...
2% (41 °C)

GPU 1
AMD Radeon(TM) Gr...
9% (43 °C)

Micron MTFDHB512C

Tiempo de actividad



60 segundos

Velocidad de transferencia de disco

60 segundos

Tiempo de actividad	Tiempo promedio de respuesta	Capacidad:	477 GB
20%	1.2	Con formato:	477 GB



Parallel and concurrent programming

iMuchas gracias!
