Notebook 2: Introduction to RAPIDS cuDF

1. pandas vs cuDF: Tabular Titans

Imagine pandas as your trusty desktop spreadsheet—great for small tasks, but it groans under massive tables. Enter cuDF, the same spreadsheet, but supercharged by your GPU:

python

```python
import pandas as pd
df = pd.DataFrame({
    'key':   [0, 0, 2, 2, 3],
    'value': [10.0, 11.0, 12.0, 13.0, 14.0]
})
print(df)
```

With cuDF, you simply copy your pandas DataFrame into GPU memory:

python

```python
import cudf
gdf = cudf.DataFrame(df)    # pandas → cuDF
print(gdf)
```

Now all your filters, merges, and aggregations zip along on the GPU!

Self-Assessment: Section 1
(Answer sheet is available in the end of the document)

1. Which object lives in GPU memory?
    A) pandas.DataFrame
    B) cudf.DataFrame
    C) numpy.ndarray
    D) list

2. How do you convert a pandas DataFrame df to a cuDF DataFrame?
    A) df.cuda()
    B) cudf.DataFrame(df)
    C) df.to_gpu()
    D) pd.to_cudf(df)

2. CPU vs GPU: GroupBy Showdown

Let's see who wins at grouping and aggregating 50 million rows—your CPU or your GPU?

python

```python
rom time import perf_counter
import numpy as np
import pandas as pd

# Prepare 50M-row table
n = 50_000_000
pdf = pd.DataFrame({
    'key':   np.random.randint(0, 1000, n),
    'value': np.random.randn(n)
})

# pandas timing
start = perf_counter()
pdf.groupby('key')['value'].agg(['sum','mean'])
print("pandas time:", perf_counter() - start)

# cuDF timing
import cudf
gdf = cudf.DataFrame.from_pandas(pdf)
start = perf_counter()
gdf.groupby('key')['value'].agg(['sum','mean'])
cudf.cuda.stream().synchronize()
print("cuDF time:", perf_counter() - start)
```

Watch pandas pant while cuDF sprints!

Self-Assessment: Section 2
(Answer sheet is available in the end of the document)

3. Why call synchronize() after the cuDF groupby?
    A) To free GPU memory
    B) To wait for GPU work before stopping the timer
    C) To merge GPU results into CPU memory
    D) To enable kernel fusion

4. For very large groupby tasks, which is usually faster?
    A) pandas on the CPU
    B) cuDF on the GPU
    C) They're always identical
    D) It depends on your Python version

3. Zero-Code GPU Acceleration

What if you could GPU-boost your pandas code without rewriting a single line? That's the magic of the cudf.pandas extension:

bash
%load_ext cudf.pandas

Now all your usual pandas imports automatically try to run on the GPU:

python

```python
import pandas as pd
# pd.groupby(), pd.merge(), pd.dropna(), etc.
# all get offloaded to cuDF under the hood!
```

No code changes. Just instant GPU power!

Self-Assessment: Section 3
(Answer sheet is available in the end of the document)

5. Loading cudf.pandas lets you:
    A) Write SQL in Python
    B) Auto-offload pandas ops to the GPU
    C) Replace pandas with Dask
    D) Use pandas without installing it

6. If a pandas operation can't run on the GPU, cuDF will:
    A) Throw an error
    B) Fall back to CPU execution
    C) Halt the notebook
    D) Recompile the kernel

4. Real-World Example: NYC 311 Calls

Let's apply our newfound speed to NYC's 311 Service Requests—millions of records cleaned, grouped, and summarized in a flash:

Python

```
%%cudf.pandas.profile
import pandas as pd

# Load & trim to useful columns
df = pd.read_csv('311_service_requests.csv')[[
    "borough","complaint_type"
]].dropna()

# Count calls per borough
counts = df.groupby('borough').size()
print(counts)

# Find top complaint in each borough
top = (df.groupby(['borough','complaint_type'])
        .size()
        .reset_index(name='count')
        .sort_values('count', ascending=False)
        .groupby('borough')
        .first())
print(top)
```

The built-in profiler shows you exactly which steps ran on GPU and how long they took—data insight at warp speed!

Self-Assessment: Section 4
(Answer sheet is available in the end of the document)

7. In the code above, .size() on a groupby returns:
    A) Number of DataFrame columns
    B) Row count per group
    C) GPU memory used
    D) Average complaint length

8. The %%cudf.pandas.profile magic lets you:
    A) See GPU vs CPU timings for each operation
    B) Auto-optimize your code
    C) Visualize charts in-line
    D) Export results to SQL

Notebook 3: Introduction to RAPIDS cuML

1. scikit-learn vs cuML: Same API, Supercharged

Ever wish your favorite scikit-learn models could run at GPU speed? cuML mimics scikit-learn's interface so you can copy-paste most code and watch it fly:

python

```python
from sklearn.datasets import fetch_california_housing
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
import sklearn.metrics as metrics

california = fetch_california_housing()
X = california.data
y = california.target

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42
)

model_cpu = LinearRegression(copy_X=False)
model_cpu.fit(X_train, y_train)
preds_cpu = model_cpu.predict(X_test)
score_cpu = metrics.r2_score(y_test, preds_cpu)
print(f"CPU R² score: {score_cpu:.2f}")
```

Now swap in cuML:

Python

```python
# GPU version with cuML
import cudf, cuml
from cuml.model_selection import train_test_split as gpu_split
from cuml.linear_model import LinearRegression as GPURegression
from cuml.metrics import r2_score as gpu_r2

# Load data via scikit-learn, then wrap in cudf
import sklearn.datasets
cal = sklearn.datasets.fetch_california_housing()
Xg = cudf.DataFrame(cal.data, columns=cal.feature_names)
yg = cudf.Series(cal.target)

Xg_train, Xg_test, yg_train, yg_test = gpu_split(
    Xg, yg, test_size=0.2, random_state=42
)

model_gpu = GPURegression(copy_X=False)
model_gpu.fit(Xg_train, yg_train)
preds_gpu = model_gpu.predict(Xg_test)
score_gpu = gpu_r2(yg_test, preds_gpu)
print(f"GPU R² score: {score_gpu:.2f}")
```

Tip: You'll see nearly identical code—only the imports change!

Self-Assessment: Section 1
(Answer sheet is available in the end of the document)

9. Which import gives you GPU-accelerated Linear Regression?
    A) from sklearn.linear_model import LinearRegression
    B) from cuml.linear_model import LinearRegression
    C) import cudf.LinearRegression
    D) import cuml.datasets

10. What metric do we use to evaluate the regression model here?
    A) Mean Squared Error
    B) R-squared
    C) Accuracy
    D) Log Loss

2. K-Means Clustering: CPU vs GPU + Visualization

Time to cluster 1.5 million random points. First, the CPU:

python

```python
import numpy as np
from sklearn.cluster import KMeans
from time import perf_counter
import matplotlib.pyplot as plt

# Generate data
np.random.seed(42)
data = np.random.rand(1_500_000, 2)

# CPU clustering
start = perf_counter()
kmeans_cpu = KMeans(n_clusters=3, random_state=42)
kmeans_cpu.fit(data)
cpu_time = perf_counter() - start
print(f"CPU time: {cpu_time:.4f} s")

# Plot CPU result
plt.figure(figsize=(6,6))
plt.scatter(data[:,0], data[:,1], c=kmeans_cpu.labels_, s=1)
plt.scatter(kmeans_cpu.cluster_centers_[:,0],
            kmeans_cpu.cluster_centers_[:,1],
            c='red', s=200, alpha=0.5)
plt.title("CPU K-Means")
plt.show()
```

Now the GPU version:

Python

```
import cudf, cuml

# Wrap data in a cudf DataFrame
df_gpu = cudf.DataFrame({'x': data[:,0], 'y': data[:,1]})

start = perf_counter()
kmeans_gpu = cuml.cluster.KMeans(n_clusters=3, random_state=42)
kmeans_gpu.fit(df_gpu)
gpu_time = perf_counter() - start
print(f"GPU time: {gpu_time:.4f} s")
print(f"Speedup: {cpu_time/gpu_time:.2f}×")

# Convert centers back for plotting
centers = kmeans_gpu.cluster_centers_.to_pandas().values

plt.figure(figsize=(6,6))
plt.scatter(data[:,0], data[:,1], c=kmeans_gpu.labels_.to_numpy(), s=1)
plt.scatter(centers[:,0], centers[:,1], c='red', s=200, alpha=0.5)
plt.title("GPU K-Means")
plt.show()
```

Self-Assessment: Section 2
(Answer sheet is available in the end of the document)

11. Which class runs K-Means on the GPU?
    A) sklearn.cluster.KMeans
    B) cuml.cluster.KMeans
    C) cudf.cluster.KMeans
    D) numpy.cluster.KMeans

12. In our scatter plot, what do the red points show?
    A) Data samples
    B) Cluster centers
    C) Outliers
    D) Noise

3. cuML in Practice: Clustering NYC 311 Calls

Let's cluster NYC's 311 calls by location! Start by loading and cleaning:

Python

```
import pandas as pd
from time import perf_counter
from sklearn.cluster import KMeans

# Load full dataset
df_full = pd.read_csv('311_service_requests.csv')

# Keep only records with lat & long
df_clean = df_full.dropna(subset=['latitude','longitude']).copy()
X = df_clean[['latitude','longitude']].to_numpy()

# Standardize (z-scale)
mean_lat, std_lat = X[:,0].mean(), X[:,0].std()
mean_lon, std_lon = X[:,1].mean(), X[:,1].std()
X_scaled = np.column_stack([
    (X[:,0] - mean_lat)/std_lat,
    (X[:,1] - mean_lon)/std_lon
])

# CPU clustering
start = perf_counter()
km_cpu = KMeans(n_clusters=3, random_state=42, n_init=100, max_iter=3000)
km_cpu.fit(X_scaled)
print(f"CPU 311 time: {perf_counter()-start:.4f} s")

# Plot
plt.figure(figsize=(6,6))
plt.scatter(X_scaled[:,0], X_scaled[:,1],
            c=km_cpu.labels_, s=1)
plt.scatter(km_cpu.cluster_centers_[:,0],
            km_cpu.cluster_centers_[:,1],
            c='red', s=200, alpha=0.5)
plt.title("CPU 311 Clusters")
plt.show()
```

Your turn: Try rewriting the above with cuML (using a cudf.DataFrame) and compare speeds!

Self-Assessment: Section 3
(Answer sheet is available in the end of the document)

13. What does df.dropna(subset=['latitude','longitude']) do?
    A) Removes rows missing lat/long
    B) Fills missing values with zeros

C) Converts floats to ints
D) Normalizes coordinates

14. Why standardize (z-scale) your latitude/longitude before clustering?
    A) To encrypt data
    B) To put both features on the same scale
    C) To reduce the number of clusters
    D) To speed up CSV reading

Summary & Key Takeaways

- Familiar API: cuML mirrors scikit-learn, so most code just needs import changes.
- Serious Speedups: Algorithms like Linear Regression and K-Means run much faster on GPU for large datasets.
- End-to-End GPU: Pair cuML with cuDF for a fully GPU-accelerated workflow.
- Visual & Practical: Always visualize clusters and compare CPU/GPU times to understand your speed gain.

ANSWER KEY

Question	Answer
1	**B**
2	**B**
3	C
4	**B**
5	**B**
6	**B**
7	**B**
8	**A**
9	**B**
10	**B**
11	**B**
12	**B**
13	**A**
14	**B**