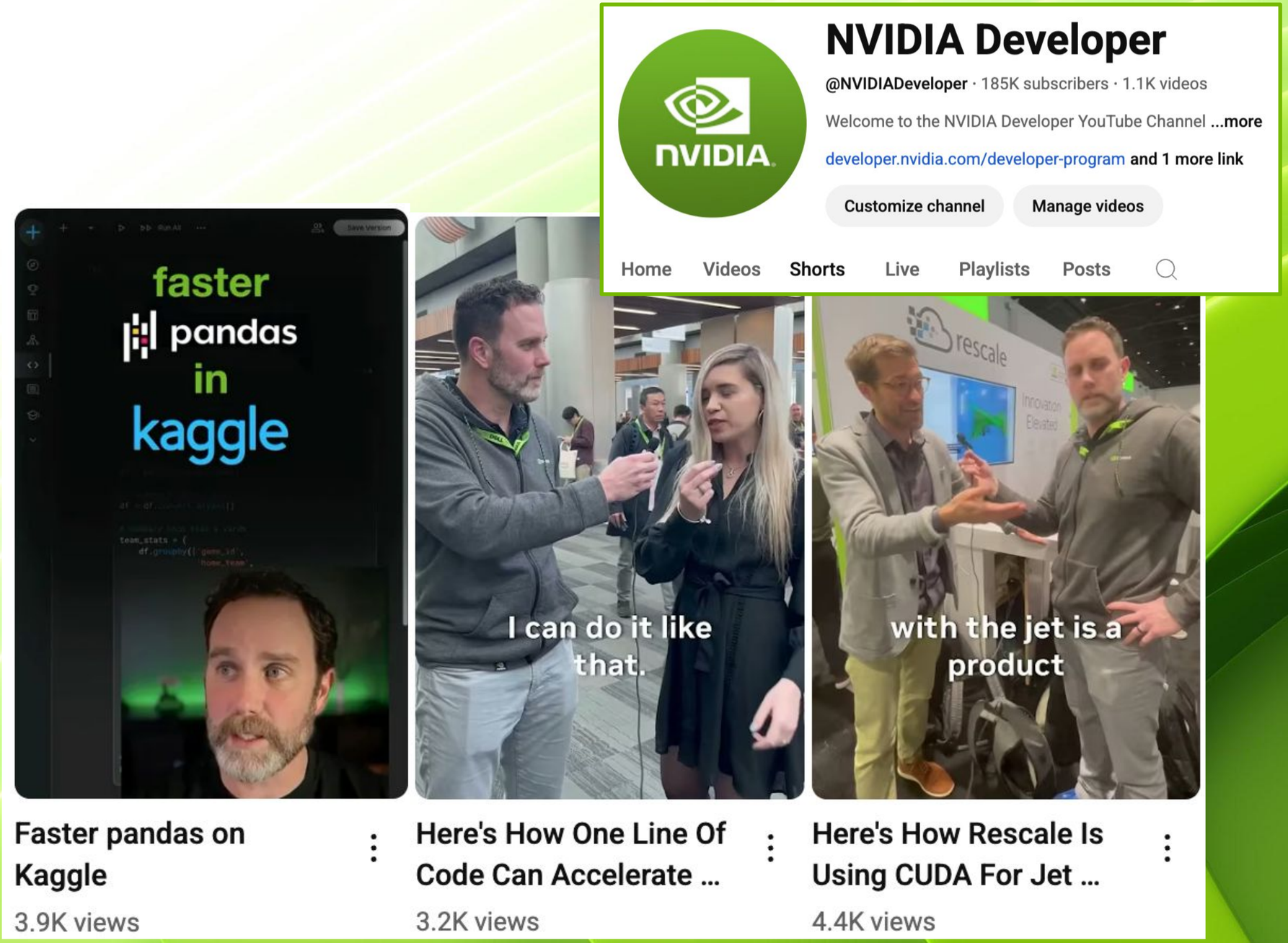
Accelerated Data Science
Same Code, but *Faster*

William Hill
Sr. Developer Advocate, NVIDIA

Speaker Intro
Who am I?

William Hill

linkedin.com/in/accelerated-ai/

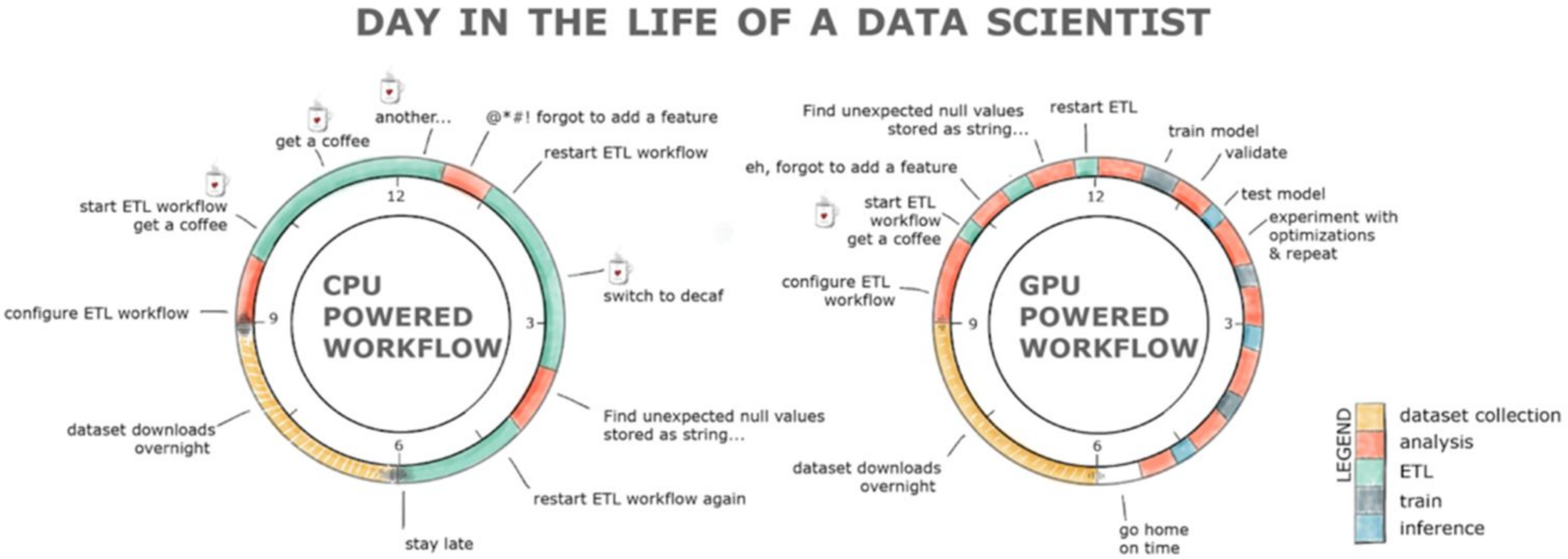


Intro

Traditional Data Science Pain Points

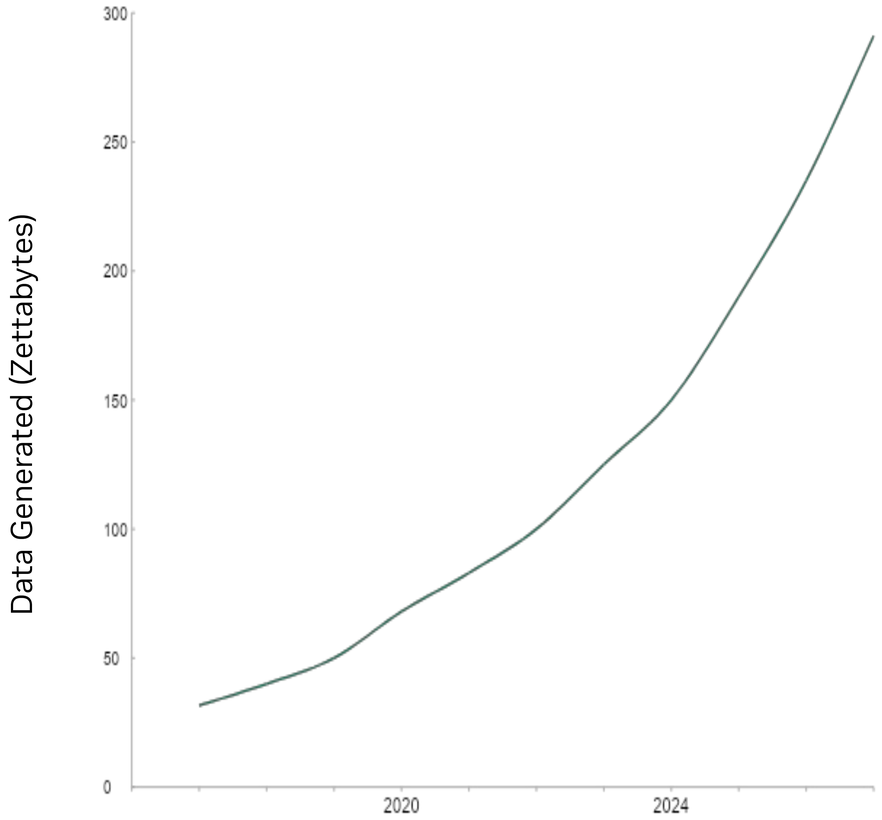
Back in Time 2015

Work Schedule

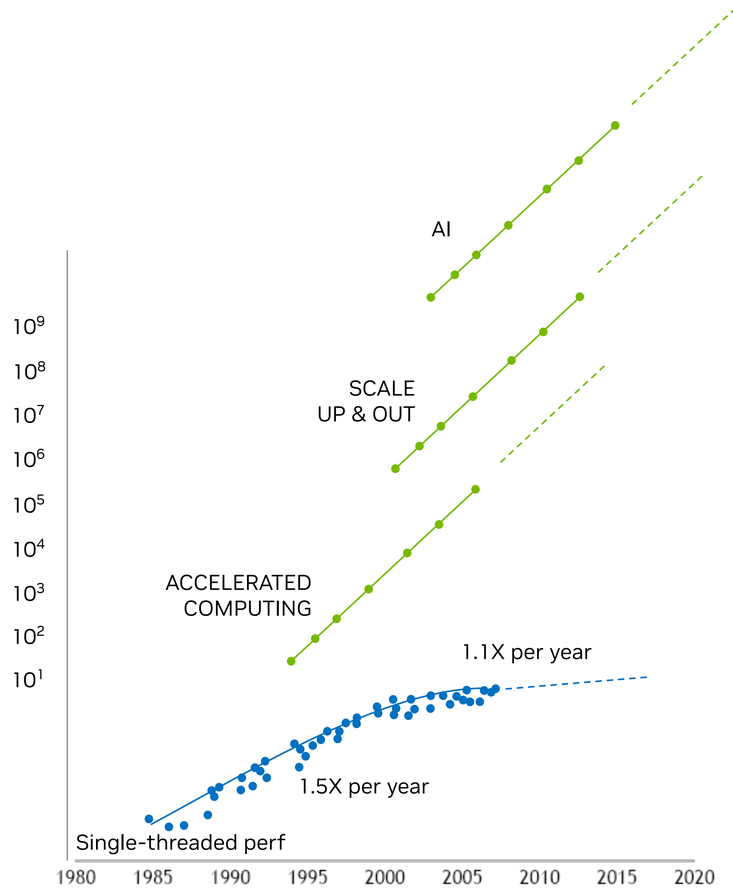


Data Analytics Needs Accelerated Computing

Petabytes of Data Generated Yearly



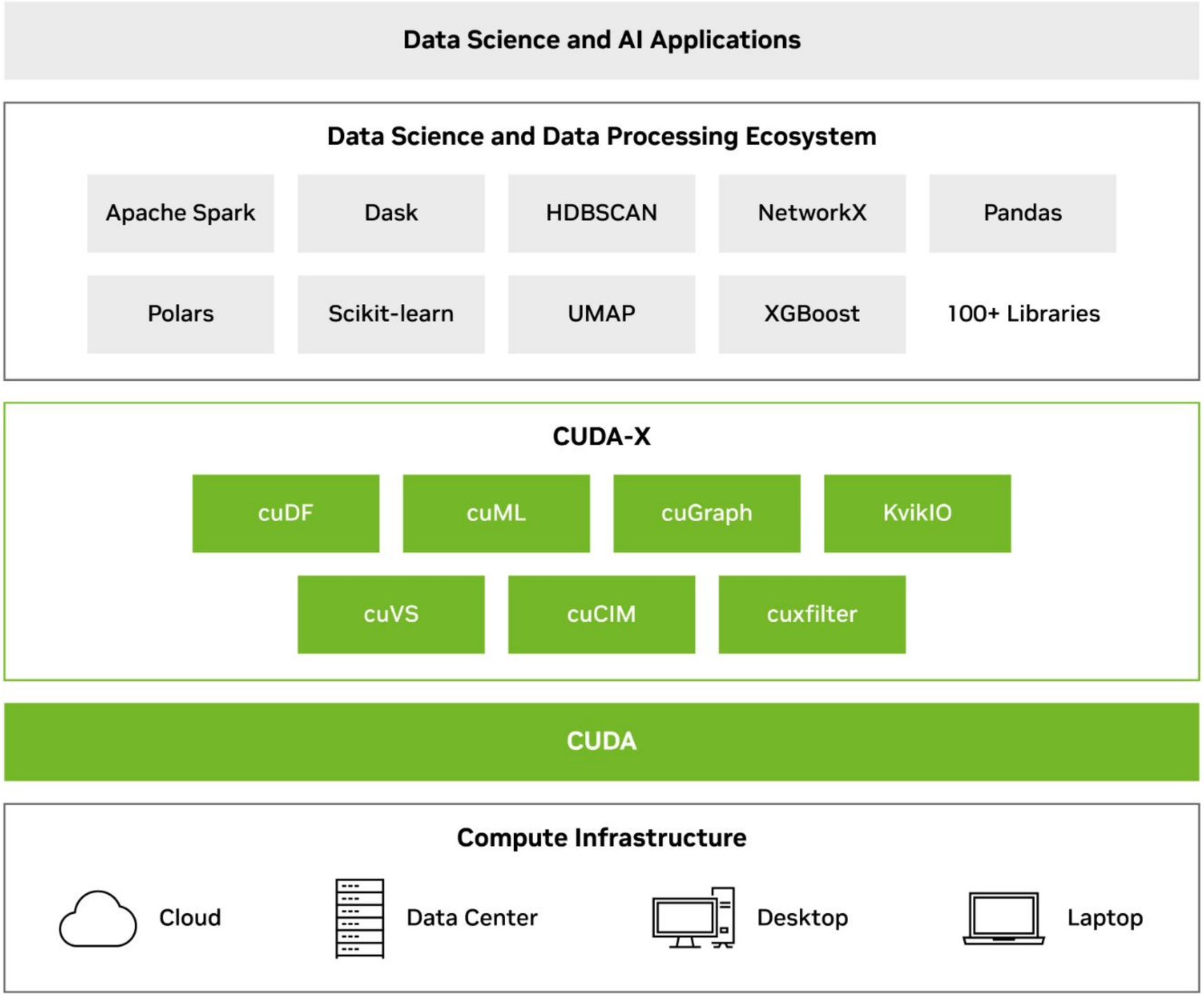
Moore's Law Has Ended



Accelerated Computing Swim Lanes

Making accelerated computing more seamless while enabling deep customization for maximum performance

The Open-Source Scientific Python Stack


Data Science and AI Applications
Data Science and Data Processing Ecosystem
Apache Spark
Dask
HDBSCAN
NetworkX
Pandas
Polars
Scikit-learn
UMAP
XGBoost
100+ Libraries
CUDA-X
cuDF
cuML
cuGraph
KvikIO
cuVS
cuCIM
cuxfilter
CUDA
Compute Infrastructure
Cloud
Data Center
Desktop
Laptop

Data Science and AI Applications

Data Science and Data Processing Ecosystem

100+ Libraries

CUDA-X

cuDF | cuML | cuGraph | KvikIO

cuVS | cuCIM | cuxfilter

CUDA

Compute Infrastructure


Cloud


Data Center


Desktop

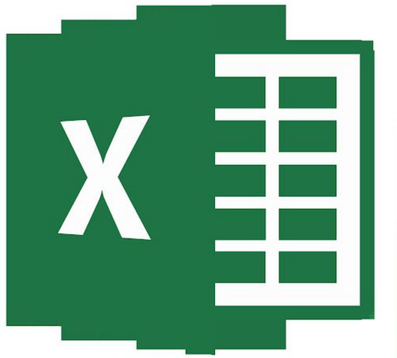

Laptop

The Data Frame

APACHE Spark™

Polars

pandas



Google Sheets

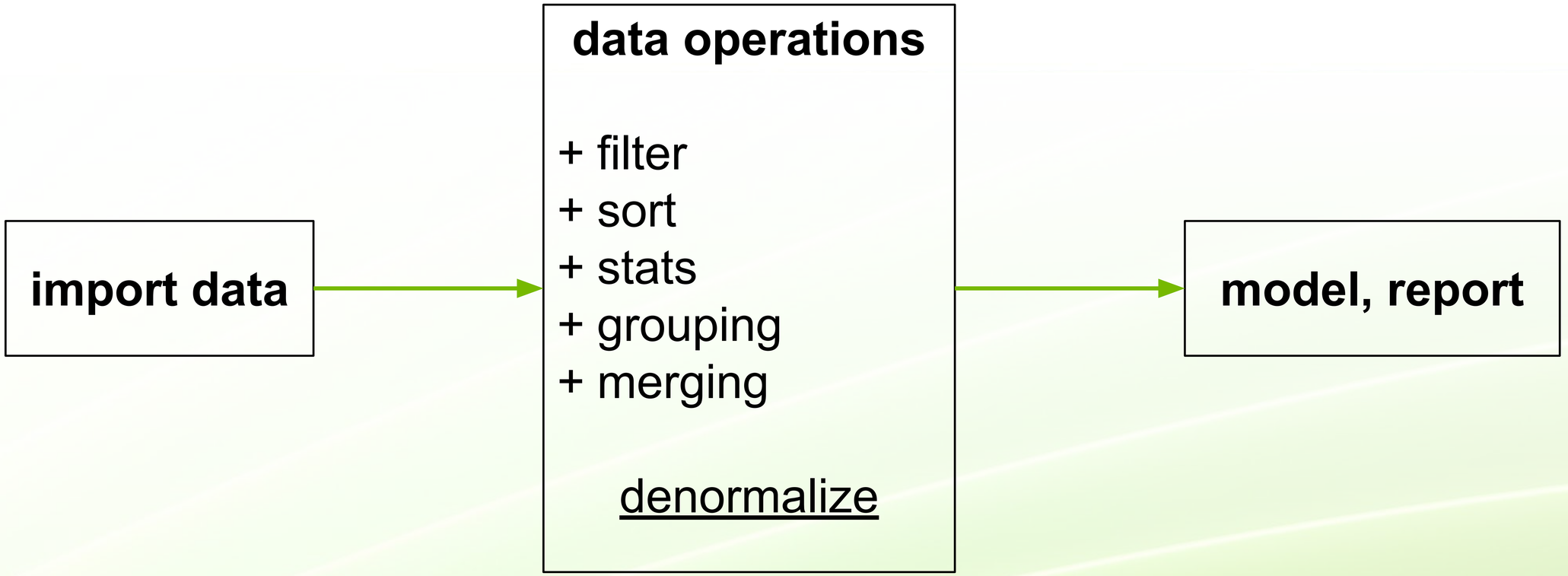
Data Prep for ML, Data Science, Reporting

import data

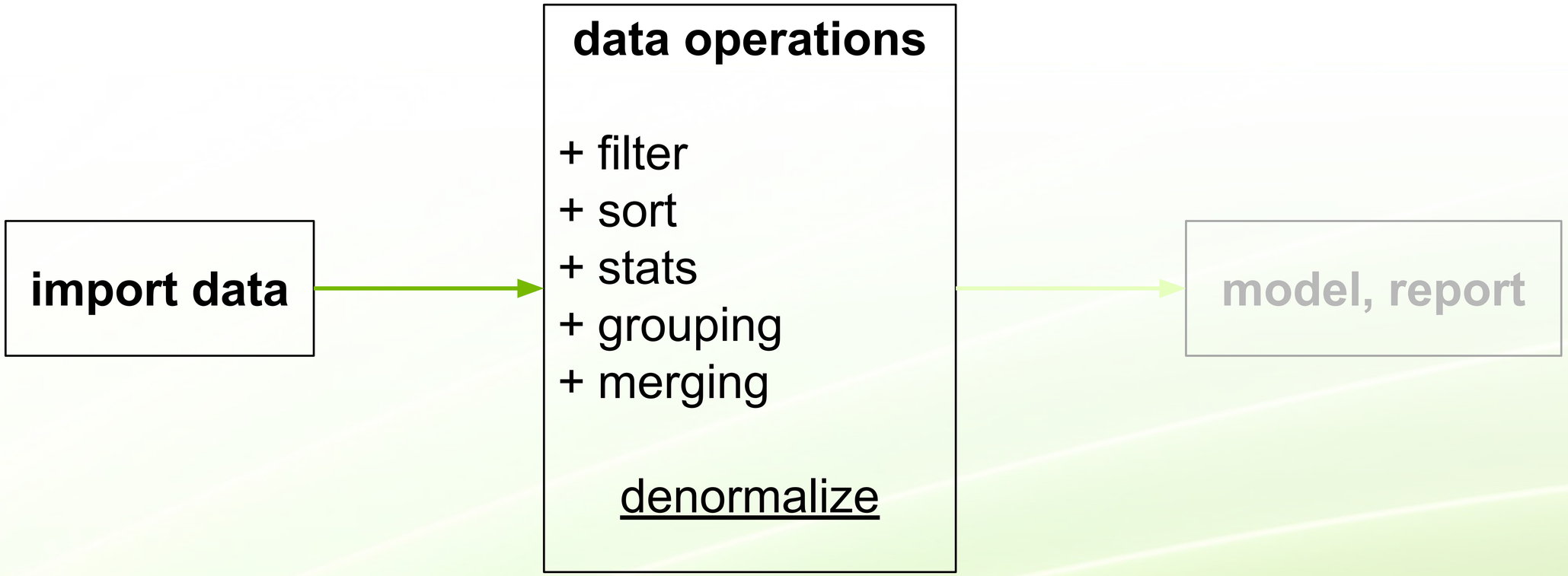
data operations

model, report

Data Prep for ML, Data Science, Reporting



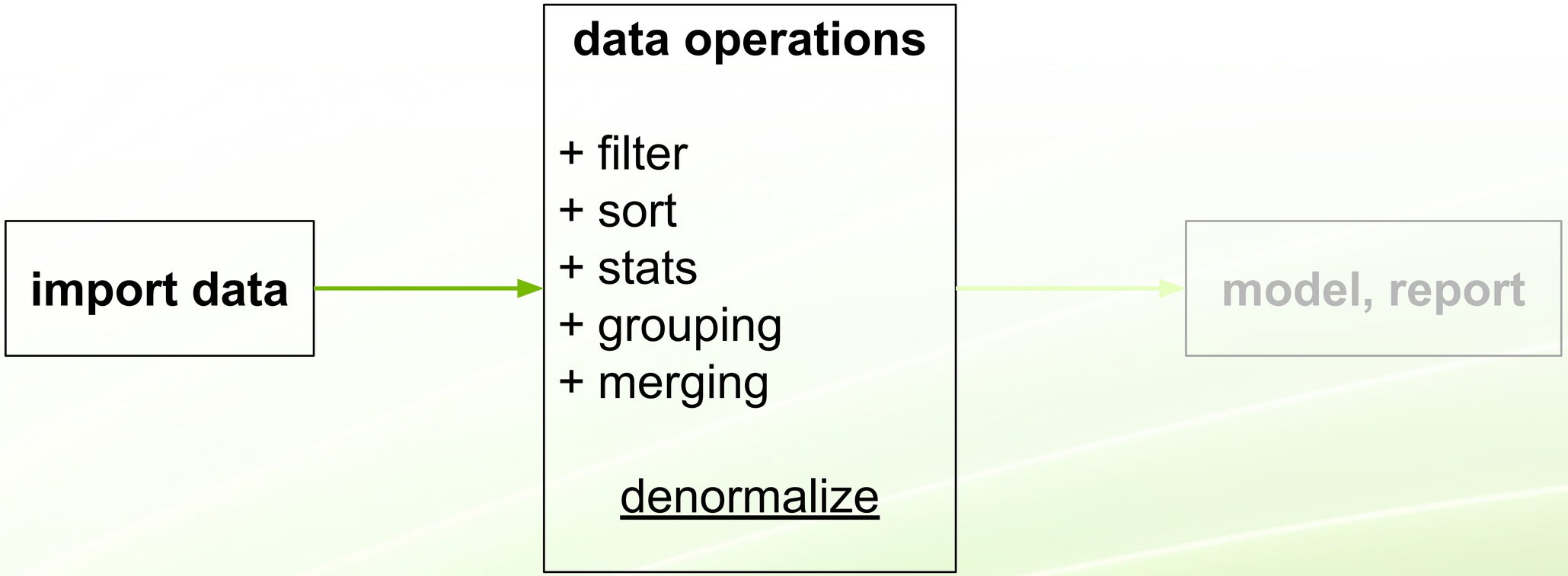
Data Prep for ML, Data Science, Reporting



NVIDIA cuDF

“CUDA for Data Frames”

Data Prep for ML, Data Science, Reporting



Accelerating *Data* with
Zero Code Change for

- ***pandas***
- ***Apache Spark***
- ***Polars***

Machine Specs

```
!nproc
!free -g
!nvidia-smi
```

```
32
               total        used        free      shared  buff/cache   available
Mem:              62           4          55           0           2          57
Swap:              0           0           0
Thu Oct  2 16:49:58 2025
+-----------------------------------------------------------------------------------------+
| NVIDIA-SMI 550.120                Driver Version: 550.120        CUDA Version: 12.4     |
|-----------------------------------------+------------------------+----------------------+
| GPU  Name                 Persistence-M | Bus-Id          Disp.A | Volatile Uncorr. ECC |
| Fan  Temp   Perf          Pwr:Usage/Cap |           Memory-Usage | GPU-Util  Compute M. |
|                                         |                        |               MIG M. |
|=========================================+========================+======================|
|   0  NVIDIA RTX 6000 Ada Gene...    Off |   00000000:01:00.0 Off |                    0 |
| 30%   35C    P8             28W /  300W |       2MiB /  46068MiB |      0%      Default |
|                                         |                        |                  N/A |
+-----------------------------------------+------------------------+----------------------+

+-----------------------------------------------------------------------------------------+
| Processes:                                                                              |
|  GPU   GI   CI        PID   Type   Process name                              GPU Memory |
|        ID   ID                                                               Usage      |
|=========================================================================================|
|  No running processes found                                                             |
+-----------------------------------------------------------------------------------------+
```

NVIDIA cuDF

Load NYC Taxi Data into pandas withOUT GPU.

[1]:
```
import pandas as pd
import glob
```

[2]:
```
%%time
# load NYC Data into pandas withOUT GPU
df = pd.concat([pd.read_parquet(f) for f in glob.glob("nyc_taxi_data/*.parquet")], ignore_index=True)
```

```
CPU times: user 25.1 s, sys: 10.6 s, total: 35.7 s
Wall time: 5.57 s
```

Load NYC Taxi Data into pandas WITH GPU

[1]:
```
%load_ext cudf.pandas
```

[2]:
```
import pandas as pd
import glob
```

[3]:
```
%%time
# load NYC Data into pandas withOUT GPU
df = pd.concat([pd.read_parquet(f) for f in glob.glob("nyc_taxi_data/*.parquet")], ignore_index=True)
```

```
CPU times: user 1.64 s, sys: 520 ms, total: 2.16 s
Wall time: 1.53 s
```

NVIDIA cuDF

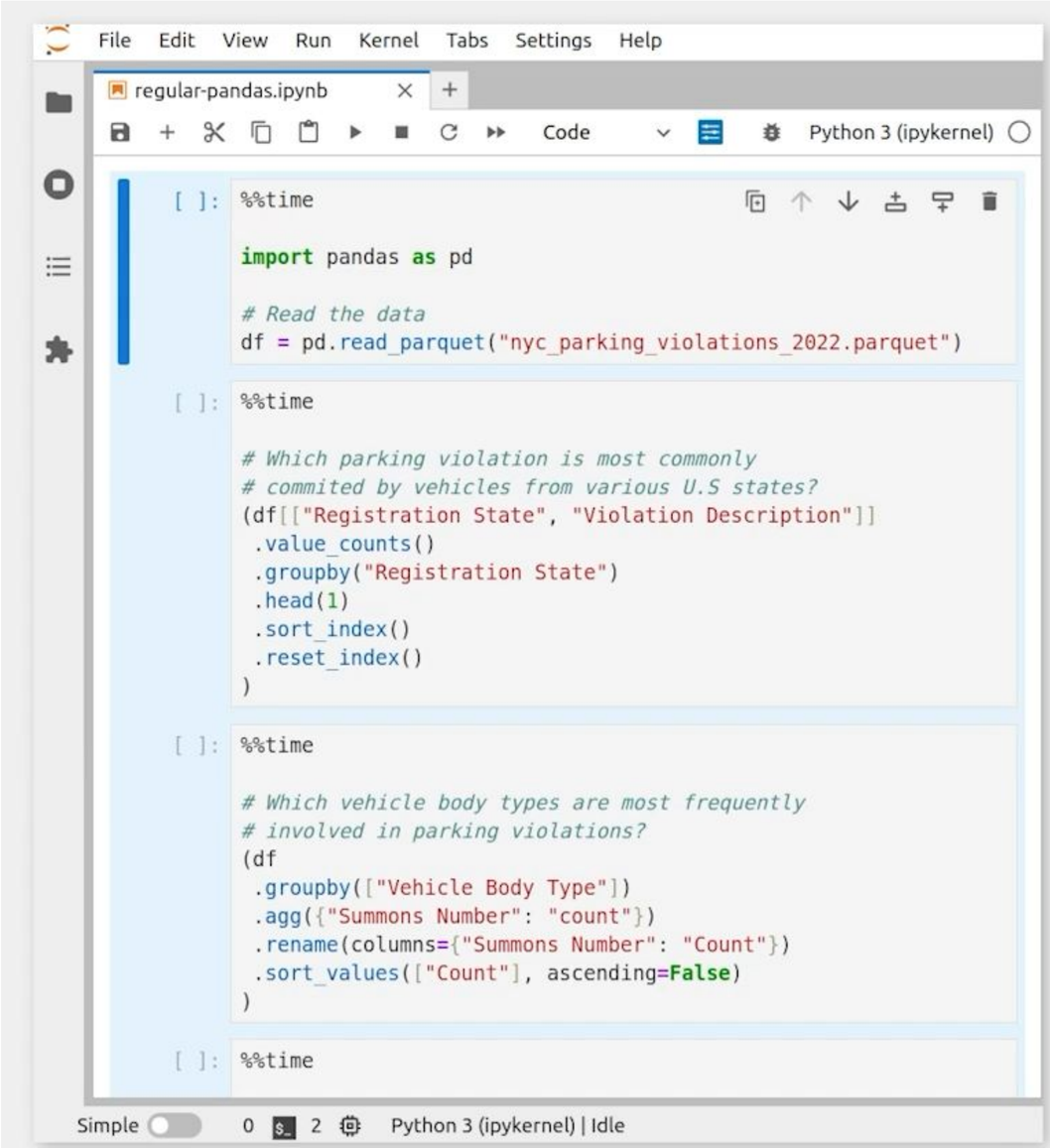
x3.6 Speed up!

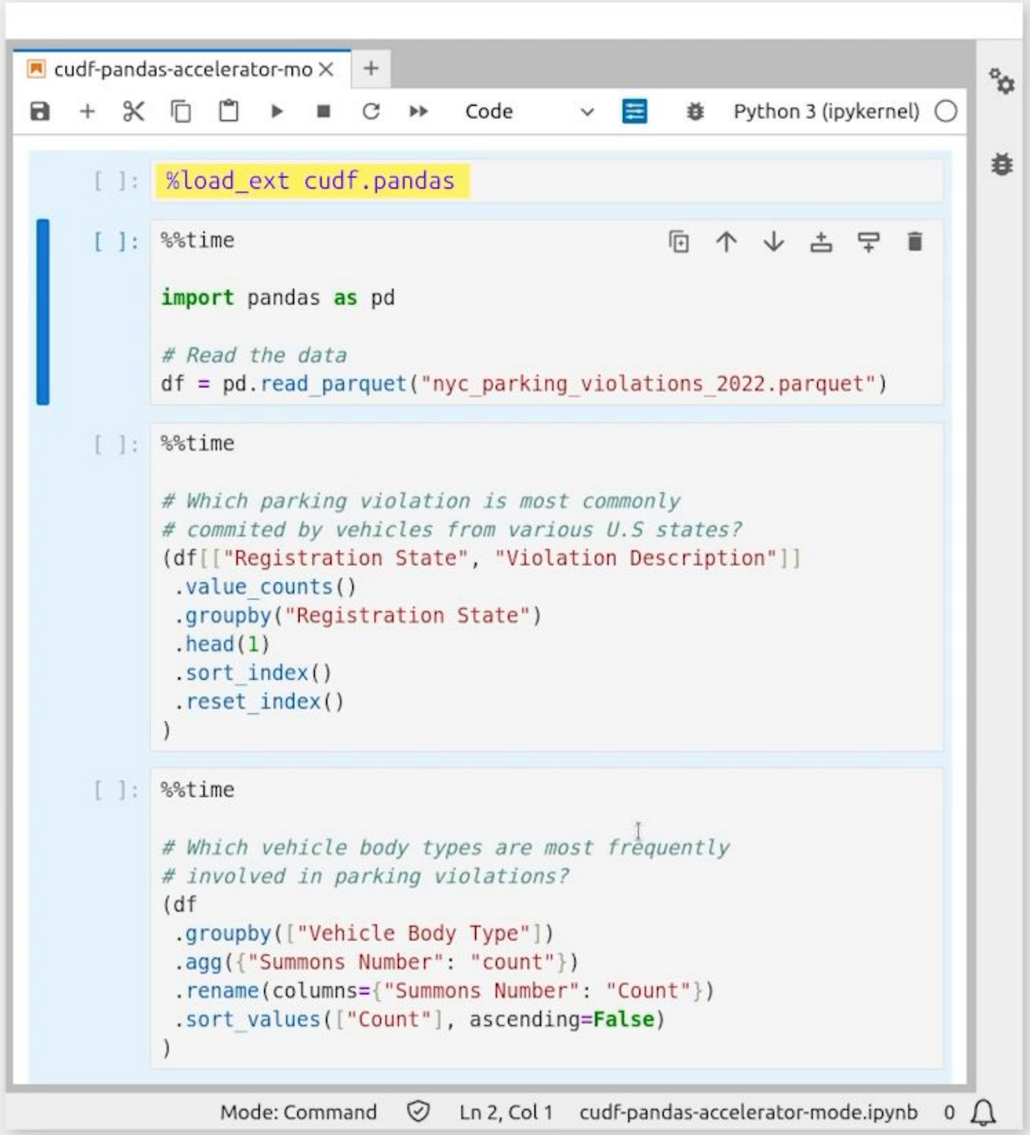
[4]:
```
df.shape
```

```
(115939623, 20)
```

116M Rows

Accelerating Pandas with NVIDIA cuDF





```
[1]: %load_ext cudf.pandas
     # > python -m cudf.pandas your_code.py

[2]: import pandas as pd
     import glob

     pd

[2]: <module 'pandas' (ModuleAccelerator(fast=cudf, slow=pandas))>

[3]: %%time
     # load NYC Data into pandas withOUT GPU
     df = pd.concat([pd.read_parquet(f) for f in glob.glob("nyc_taxi_data/*.parquet")], ignore_index=True)

     CPU times: user 1.67 s, sys: 477 ms, total: 2.15 s
     Wall time: 1.52 s

[4]: %%time
     df = (
         df[['VendorID', 'passenger_count', 'sto
             .sort_values(['tip_amount','fare_am
             .groupby(['VendorID', 'passenger_co
             .mean()
             .sort_values(['tip_amount','fare_am
     )

     CPU times: user 707 ms, sys: 68.8 ms, total
     Wall time: 755 ms
```

```
[1]: import pandas as pd
     import glob

     pd

[1]: <module 'pandas' from '/home/will/git/smu/venv/lib/python3.12/site-packages/pandas/__init__.py'>

[2]: %%time
     # load NYC Data into pandas withOUT GPU
     df = pd.concat([pd.read_parquet(f) for f in glob.glob("nyc_taxi_data/*.parquet")], ignore_index=True)

     CPU times: user 24.3 s, sys: 9.74 s, total: 34 s
     Wall time: 5.25 s

[3]: %%time
     df = (
         df[['VendorID', 'passenger_count', 'store_and_fwd_flag', 'PULocationID', 'tip_amount','fare_amount','total_amount']]
             .sort_values(['tip_amount','fare_amount','total_amount'], ascending=[True, False, True])
             .groupby(['VendorID', 'passenger_count', 'store_and_fwd_flag', 'PULocationID'])
             .mean()
             .sort_values(['tip_amount','fare_amount','total_amount'], ascending=[False, True, False])
     )

     CPU times: user 25 s, sys: 3.15 s, total: 28.2 s
     Wall time: 28.2 s
```

ℹ HOW TO USE IT

This mode is available in the standard cuDF package. To use `cudf.pandas`, enable it *before importing or using pandas* using one of these methods:
To accelerate IPython or Jupyter Notebooks, use the magic:

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

To accelerate a Python script, use the Python module flag on the command line:

```
python -m cudf.pandas script.py
```

Or, explicitly enable `cudf.pandas` via import if you can't use command line flags:

```
import cudf.pandas
cudf.pandas.install()

import pandas as pd
...
```

Trust but Verify

How to check if your pandas is GPU-Accelerated

```
[1]: import pandas as pd
     pd
```

```
[1]: <module 'pandas' from '/home/will/git/smu/venv/lib/python3.12/site-packages/pandas/__init__.py'>
```

```
[2]: %load_ext cudf.pandas
```

NVIDIA cuDF

```
[3]: import pandas as pd
     pd
```

```
[3]: <module 'pandas' (ModuleAccelerator(fast=cudf, slow=pandas))>
```

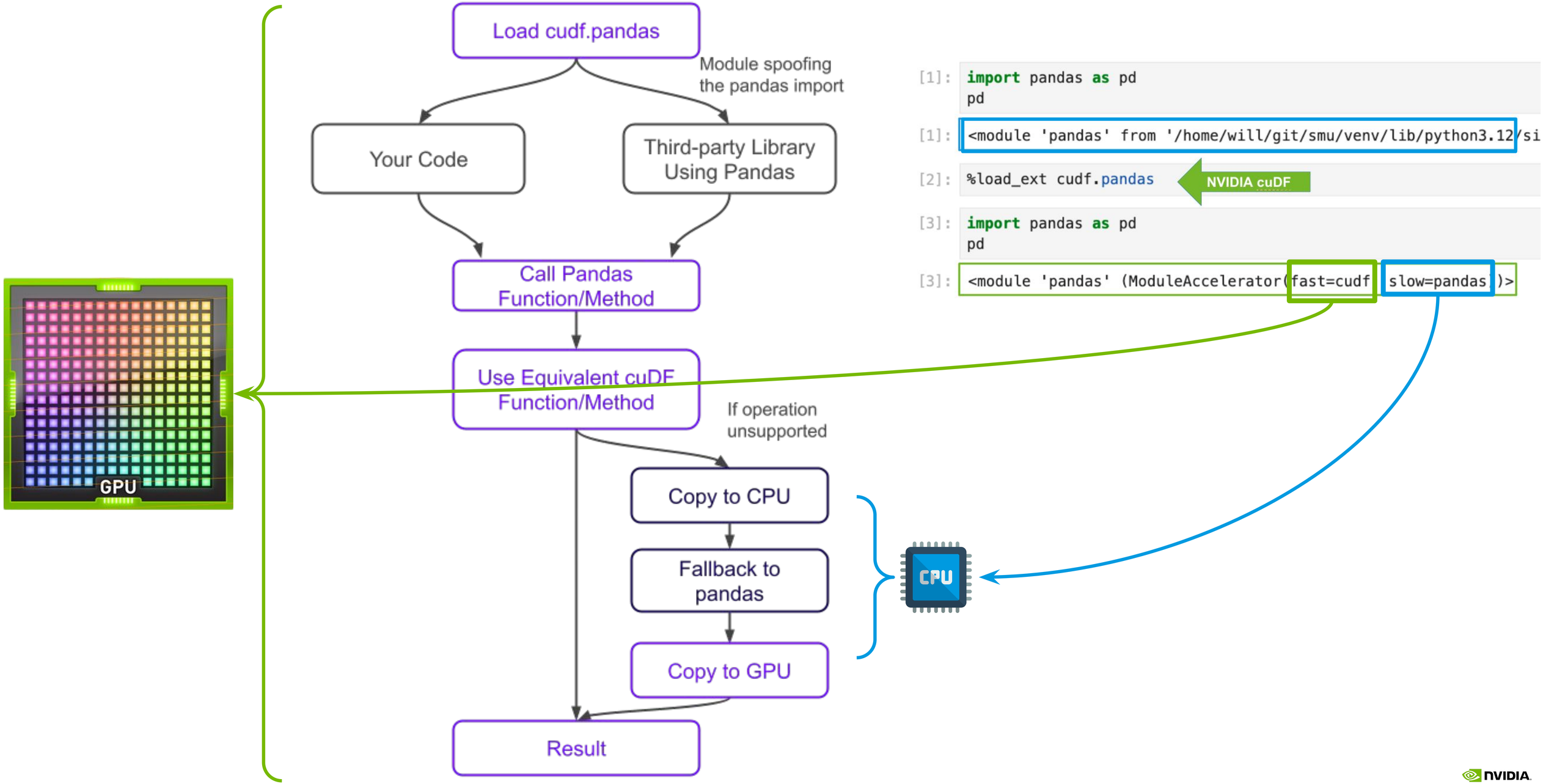
Trust but Verify

How to check if your pandas is GPU-Accelerated

```
[1]: import pandas as pd
     pd

[1]: <module 'pandas' from '/home/will/git/smu/venv/lib/python3.12/site-packages/pandas/__init__.py'>

[2]: %load_ext cudf.pandas

[3]: import pandas as pd
     pd

[3]: <module 'pandas' (ModuleAccelerator(fast=cudf, slow=pandas))>
```

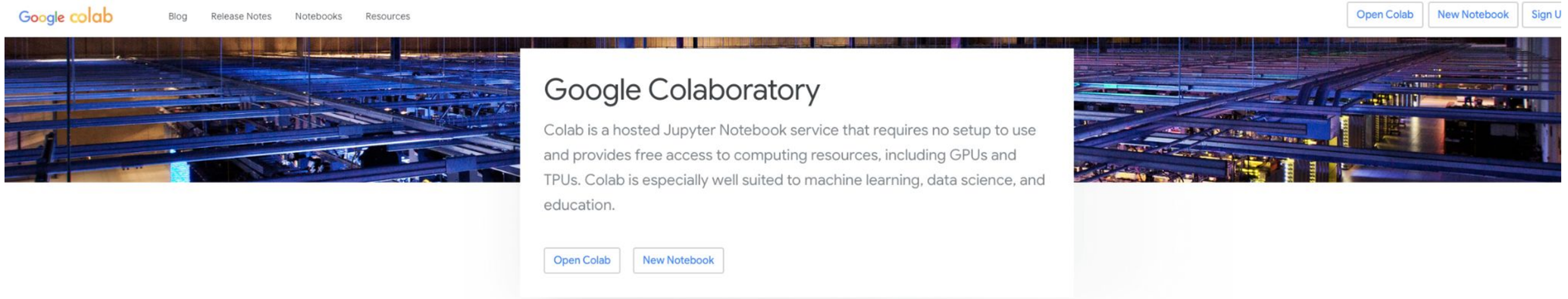
NVIDIA cuDF

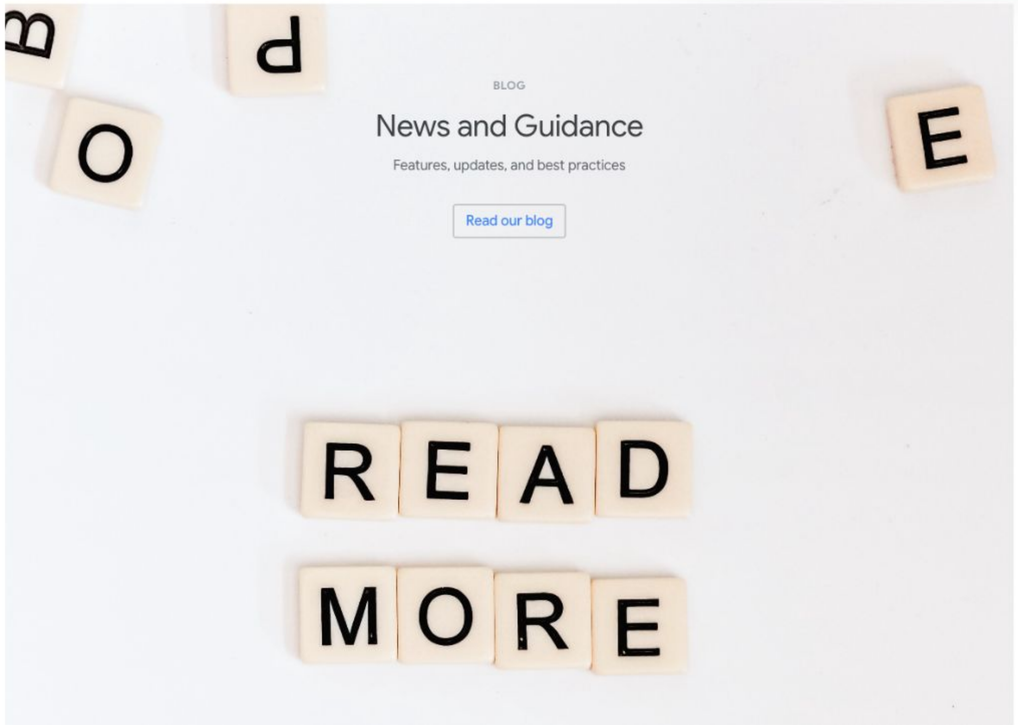
cuDF + pandas Workflow

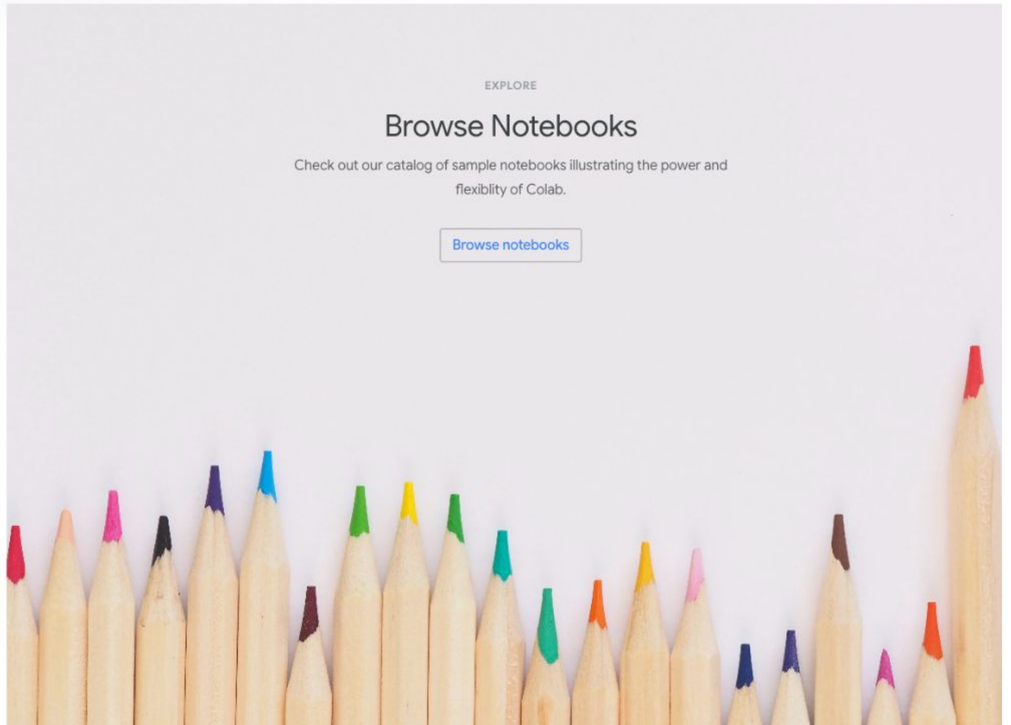


Colab Demo

Free GPUs!

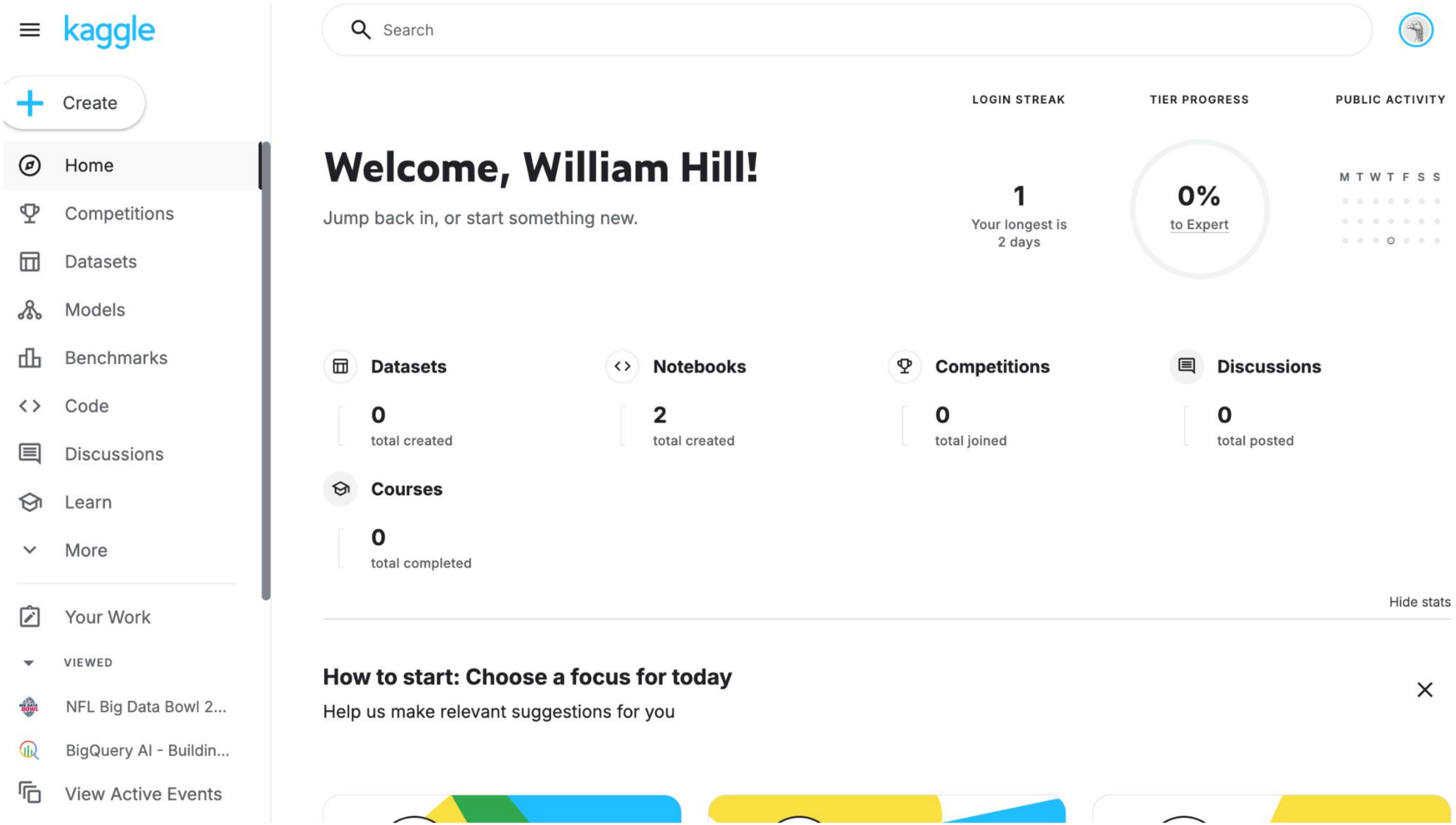






Kaggle Demo

Free GPUs!



Accelerating *ML*

Zero Code Change for XGBoost, Scikit-Learn

cuML.ipynb

Markdown | Python 3 (ipykernel)

[1]:
```
import sklearn
from sklearn.datasets import make_classification
from sklearn.model_selection import train_test_split
import xgboost as xgb
```

[2]:
```
X, y = make_classification(n_samples=500_000, n_features=100, random_state=42)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)
```

[3]:
```
%%time
model = xgb.XGBClassifier(n_estimators=500,
                          max_depth=15,
                          learning_rate=0.01,
                          device="cpu"
)

model = model.fit(X_train, y_t
```

```
CPU times: user 54min
Wall time: 1min 47s
```

"cpu"

1:47

[4]:
```
%%time
model = xgb.XGBClassifier(n_estimators=500,
                          max_depth=15,
                          learning_rate=0.01,
                          device="cuda"
)

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

```
CPU times: user 39.8 s, sys: 303 ms, total: 40.1 s
Wall time: 32.2 s
```

"cuda"

32.2 s

[6]:
```
107 / 32.2
```

[6]:
```
3.322981366459627
```

dmlc XGBoost

NVIDIA

Announcing GPU-Accelerated XGBOOST

Unleashing the Power of NVIDIA GPUs for the world's most popular ML Algorithm

- Leading algorithm for tabular data across regression, classification and ranking
- GPU-accelerated XGBoost:
  - 22X faster
  - reduces memory footprint by 50%
  - slashes costs vs CPUs
- Delivers improved accuracy by allowing time for more iterations

dmlc XGBoost

BEFORE

```
import xgboost as xgb

params = {'max_depth': 3,
          'learning_rate': 0.1}


dtrain = xgb.DMatrix(X, y)
bst = xgb.train(params, dtrain)
```

AFTER

```
import xgboost as xgb

params = {'tree_method': 'gpu_hist',
          'max_depth': 3,
          'learning_rate': 0.1}

dtrain = xgb.DMatrix(X, y)
bst = xgb.train(params, dtrain)
```

```
[1]: import sklearn
     from sklearn.datasets import make_classification
     from sklearn.model_selection import train_test_split
     from sklearn.ensemble import RandomForestClassifier

[2]: X, y = make_classification(n_samples=500_000, n_features=100)
     X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)

[3]: %%time
     rf = RandomForestClassifier(n_estimators=1_000, n_jobs=-1)
     rf.fit(X_train, y_train)

     CPU times: user 6h 9min [illegible]ys: 2 s, total: 6h 9min 35s
     Wall time: 11min 44s
```

11:44

x57 Speed up!

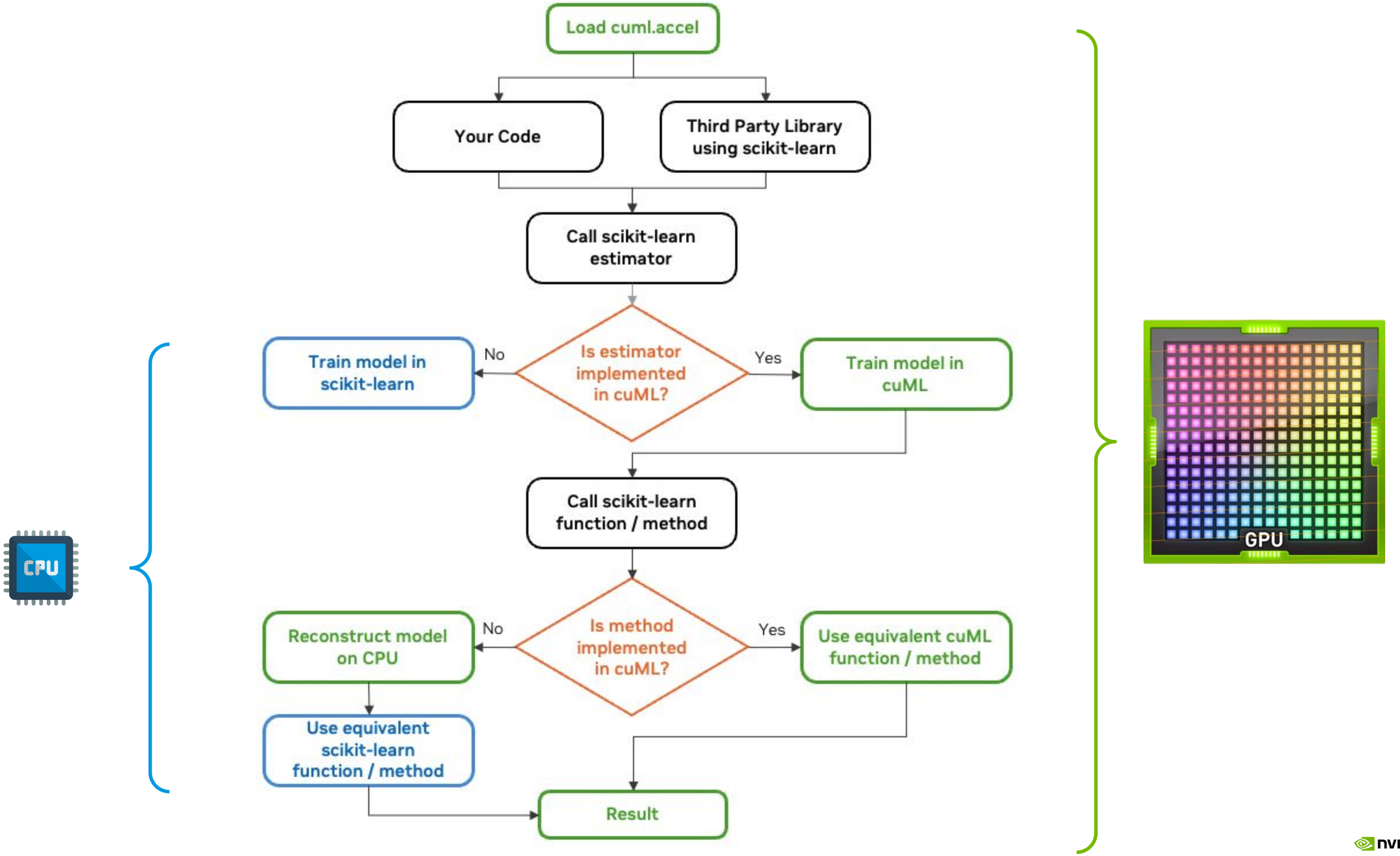
NVIDIA cuML

```
[1]: %load_ext cuml.accel

[2]: import sklearn
     from sklearn.datasets import make_classification
     from sklearn.model_selection import train_test_split
     from sklearn.ensemble import RandomForestClassifier

[3]: X, y = make_classification(n_samples=500_000, n_features=100)
     X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)

[4]: %%time
     rf = RandomForestClassifier(n_est[illegible]tors=1_000, n_jobs=-1)
     rf.fit(X_train, y_train)

     CPU times: user 22 [illegible]ys: 11 s, total: 33.8 s
     Wall time: 12.3 s
```

12.3 s

scikit learn

cuML + Scikit-Learn Workflow



Verification
How Will I Know?

cuML profiling

```
%%cuml.accel.profile

rf = RandomForestClassifier(n_estimators=1_000, n_jobs=-1)
rf = rf.fit(X_train, y_train)
```

cuml.accel profile

Function	GPU calls	GPU time	CPU calls	CPU time
RandomForestClassifier.fit	1	644.5ms	0	0s
Total	1	644.5ms	0	0s

cuDF profiling

```
%%cudf.pandas.profile

df = pd.concat([pd.read_parquet(f) for f in glob.glob("nyc_taxi_data/*.parquet")], ignore_index=True)

df = (
    df[['VendorID', 'passenger_count', 'store_and_fwd_flag', 'PULocationID', 'tip_amount','fare_amount','total_amount']]
        .sort_values(['tip_amount','fare_amount','total_amount'], ascending=[True, False, True])
        .groupby(['VendorID', 'passenger_count', 'store_and_fwd_flag', 'PULocationID'])
        .mean()
        .sort_values(['tip_amount','fare_amount','total_amount'], ascending=[False, True, False])
)
```

Total time elapsed: 2.496 seconds
37 GPU function calls in 2.347 seconds
0 CPU function calls in 0.000 seconds

Stats

Function	GPU ncalls	GPU cumtime	GPU percall	CPU ncalls	CPU cumtime	CPU percall
read_parquet	31	1.457	0.047	0	0.000	0.000
concat	1	0.146	0.146	0	0.000	0.000
DataFrame.__getitem__	1	0.002	0.002	0	0.000	0.000
DataFrame.sort_values	2	0.640	0.320	0	0.000	0.000
DataFrame.groupby	1	0.026	0.026	0	0.000	0.000
GroupBy.mean	1	0.075	0.075	0	0.000	0.000

cuML profiling

Command Line Interface (CLI)

If running using the CLI, you may add the `--profile` flag to profile your whole script.

```
python -m cuml.accel --profile script.py
```

Jupyter Notebook

If running in IPython or Jupyter, you may use the `cuml.accel.profile` cell magic to profile code running in a single cell.

```
%%cuml.accel.profile
# All code in this cell will be profiled
...
```

Programmatic Usage

Alternatively, the `cuml.accel.profile` contextmanager may be used to programmatically profile a section of code.

```
with cuml.accel.profile():
    # All code within this context will be profiled
    ...
```

More Profiling
cudf.pandas, cuml.accel

Command Line Interface (CLI)

If running using the CLI, you may add the `--profile` flag to profile your whole script.

```
python -m cuml.accel --profile script.py
```

Jupyter Notebook

If running in IPython or Jupyter, you may use the `cuml.accel.profile` cell magic to profile code running in a single cell.

```
%%cuml.accel.profile

# All code in this cell will be profiled
...
```

Programmatic Usage

Alternatively, the `cuml.accel.profile` contextmanager may be used to programmatically profile a section of code.

```
with cuml.accel.profile():
    # All code within this context will be profiled
    ...
```

```
%%cudf.pandas.line_profile

df = pd.DataFrame({'a': [0, 1, 2], 'b': [3, 4, 3]})
df.min(axis=1)
out = df.groupby('a').filter(lambda group: len(group) > 1)
```

Total time elapsed: 0.244 seconds

Stats

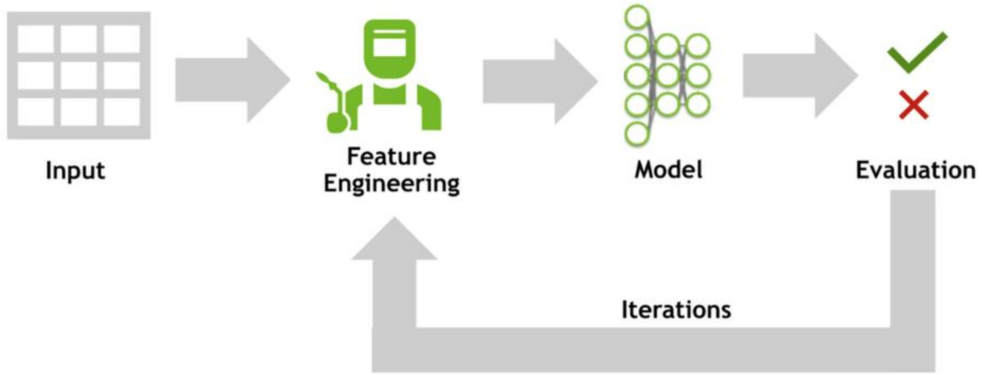
Line no.	Line	GPU TIME(s)	CPU TIME(s)
2	df = pd.DataFrame({'a': [0, 1, 2], 'b': [3, 4, 3]})	0.004833249	
3	df.min(axis=1)	0.006497159	
4	out = df.groupby('a').filter(lambda group: len(group) > 1)	0.000599624	0.000347643

Use-Case

Lessons from Kaggle Grandmasters

Data Science English

Grandmaster Pro Tip: Winning First Place in Kaggle Competition with Feature Engineering Using cuDF pandas



Apr 17, 2025 +14 Like Discuss (0)

By Chris Deotte

AI-Generated Summary

Related posts

Groupby(COL1)[COL2].agg(STAT)

The most powerful feature engineering technique is groupby aggregations. Namely, we execute the code `groupby(COL1)[COL2].agg(STAT)`. This is where we group by `COL1` column and aggregate (i.e. compute) a statistic `STAT` over another column `COL2`. We use the speed of NVIDIA cuDF-Pandas to explore thousands of `COL1`, `COL2`, `STAT` combinations. We try statistics (`STAT`) like “mean”, “std”, “count”, “min”, “max”, “nunique”, “skew” etc etc. We choose `COL1` ting columns. When `COL2` is the target column, then we use nested cross-validation putation. When `COL2` is the target, this operation is called Target Encoding.

Groupby(COL1)[‘Price’].agg(QUANTILES)

We can groupby and compute the quantiles for `QUANTILES = [5,10,40,45,55,60,90,95]` and return the eight values to create eight new columns.

```
for k in QUANTILES:
    result = X_train2.groupby('Weight Capacity (kg)').\
        agg({'Price': lambda x: x.quantile(k/100)})
```

All NANs as Single Base-2 Column

We can create a new column from all the NANs over multiple columns. This is a powerful column which we can subsequently use for groupby aggregations or combinations with other columns.

```
train["NaNs"] = np.float32(0)
for i,c in enumerate(CATS):
    train["NaNs"] += train[c].isna()*2**i
```

Put Numerical Column into Bins

The most powerful (predictive) column in this competition is Weight Capacity. We can create more powerful colum based on this column by binning this column with rounding.

```
for k in range(7,10):
    n = f"round{k}"
    train[n] = train["Weight Capacity (kg)"].round(k)
```

Groupby(COL1)[‘Price’].agg(HISTOGRAM BINS)

When we `groupby(COL1)[COL2]` we have a distribution (set) of numbers for each group. Instead of computing a single statistic (and making one new column), we can compute any collection of numbers that describe this distribution of numbers and make many new columns together.

Below we display a histogram for the group `Weight Capacity = 21.067673`. We can count the number of elements in each (equally spaced) bucket and create a new engineered feature for each bucket count to return to the groupby operation! Below we display seven buckets, but we can treat the number of buckets as a hyperparameter.

```
result = X_train2.groupby("WC")["Price"].apply(make_histogram)
X_valid2 = X_valid2.merge(result, on="WC", how="left")
```

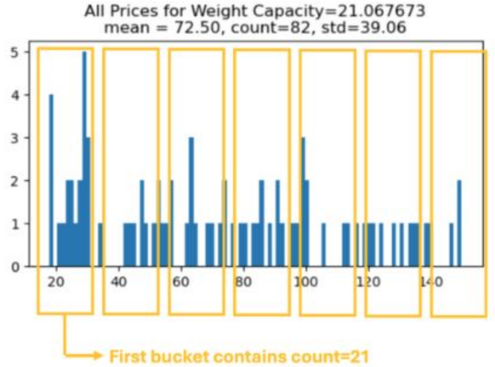


Figure 1. Histogram of price values when weight capacity equals 21.067673

```
[1]: %load_ext cudf.pandas
     # > python -m cudf.pandas your_code.py

[2]: import pandas as pd
     import glob

     pd

[2]: <module 'pandas' (ModuleAccelerator(fast=cudf, slow=pandas))>

[3]: %%time
     # load NYC Data into pandas withOUT GPU
     df = pd.concat([pd.read_parquet(f) for f in glob.glob("nyc_taxi_data/*.parquet")], ignore_index=True)

     CPU times: user 1.67 s, sys: 477 ms, total: 2.15 s
     Wall time: 1.52 s

[4]: %%time
     df = (
         df[['VendorID', 'passenger_count', 'sto
             .sort_values(['tip_amount','fare_am
             .groupby(['VendorID', 'passenger_co
             .mean()
             .sort_values(['tip_amount','fare_am
     )

     CPU times: user 707 ms, sys: 68.8 ms, total
     Wall time: 755 ms
```

```
[1]: import pandas as pd
     import glob

     pd

[1]: <module 'pandas' from '/home/will/git/smu/venv/lib/python3.12/site-packages/pandas/__init__.py'>

[2]: %%time
     # load NYC Data into pandas withOUT GPU
     df = pd.concat([pd.read_parquet(f) for f in glob.glob("nyc_taxi_data/*.parquet")], ignore_index=True)

     CPU times: user 24.3 s, sys: 9.74 s, total: 34 s
     Wall time: 5.25 s

[3]: %%time
     df = (
         df[['VendorID', 'passenger_count', 'store_and_fwd_flag', 'PULocationID', 'tip_amount','fare_amount','total_amount']]
             .sort_values(['tip_amount','fare_amount','total_amount'], ascending=[True, False, True])
             .groupby(['VendorID', 'passenger_count', 'store_and_fwd_flag', 'PULocationID'])
             .mean()
             .sort_values(['tip_amount','fare_amount','total_amount'], ascending=[False, True, False])
     )

     CPU times: user 25 s, sys: 3.15 s, total: 28.2 s
     Wall time: 28.2 s
```

Data Science

English

Grandmaster Pro Tip: Winning First Place in a Kaggle Competition with Stacking Using cuML

May 22, 2025

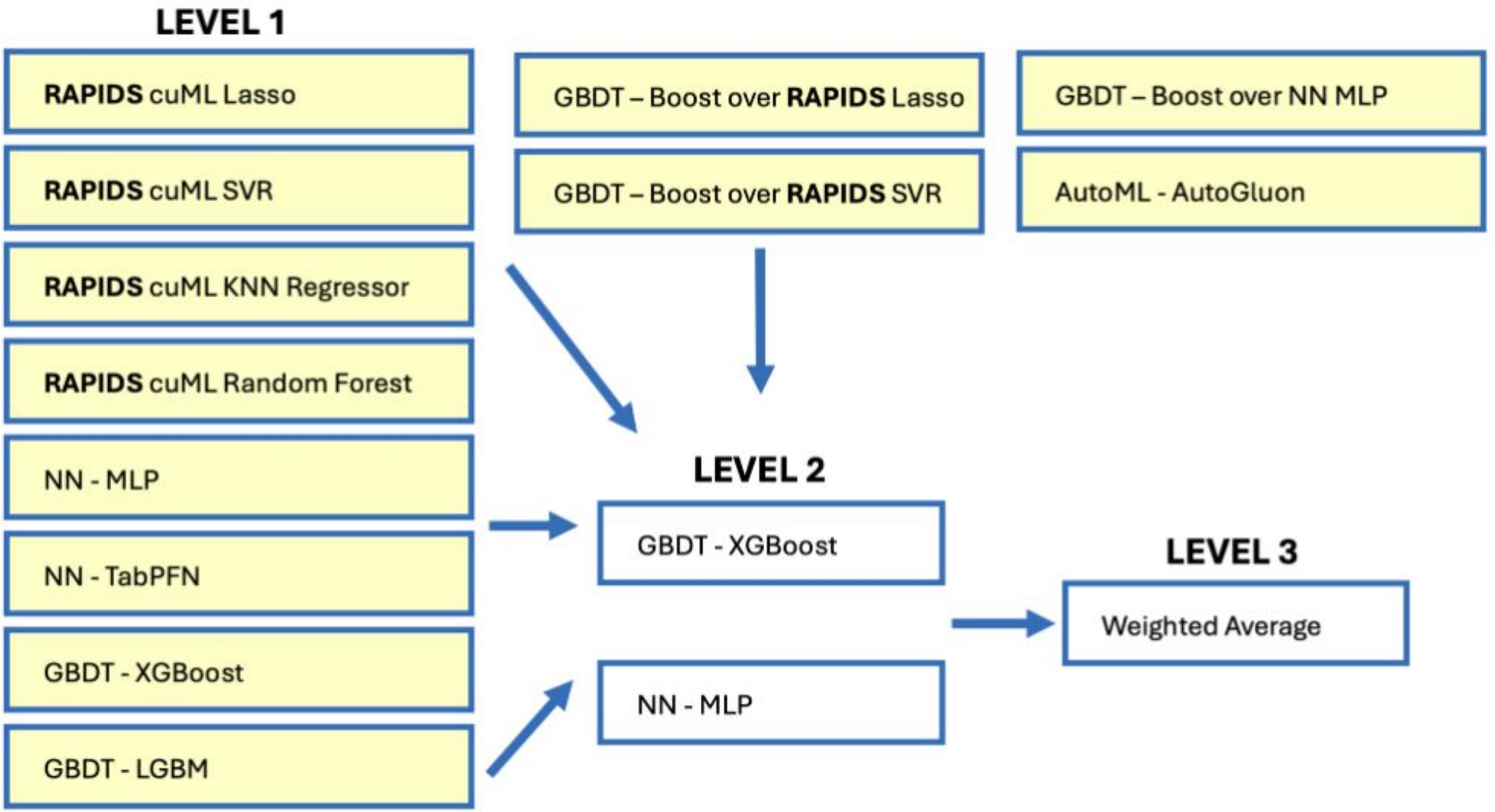


Figure 1. The winning entry in the Kaggle April 2025 Playground competition used stacking with three levels of models, with the results of each level used in subsequent levels

cuML.ipynb

Markdown

Python 3 (ipykernel)

[1]:
```
import sklearn
from sklearn.datasets import make_classification
from sklearn.model_selection import train_test_split
import xgboost as xgb
```

[2]:
```
X, y = make_classification(n_samples=500_000, n_features=100, random_state=42)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)
```

[3]:
```
%%time
model = xgb.XGBClassifier(n_estimators=500,
                          max_depth=15,
                          learning_rate=0.01,
                          device="cpu"
)

model = model.fit(X_train, y_t
```

```
CPU times: user 54min
Wall time: 1min 47s
```

“cpu”

1:47

[4]:
```
%%time
model = xgb.XGBClassifier(n_estimators=500,
                          max_depth=15,
                          learning_rate=0.01,
                          device="cuda"
)

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

```
CPU times: user 39.8 s, sys: 303 ms, total: 40.1 s
Wall time: 32.2 s
```

“cuda”

x3.3 Speedup!

[6]:
```
107 / 32.2
```

32.2 s

[6]:
```
3.322981366459627
```

dmlc XGBoost

NVIDIA

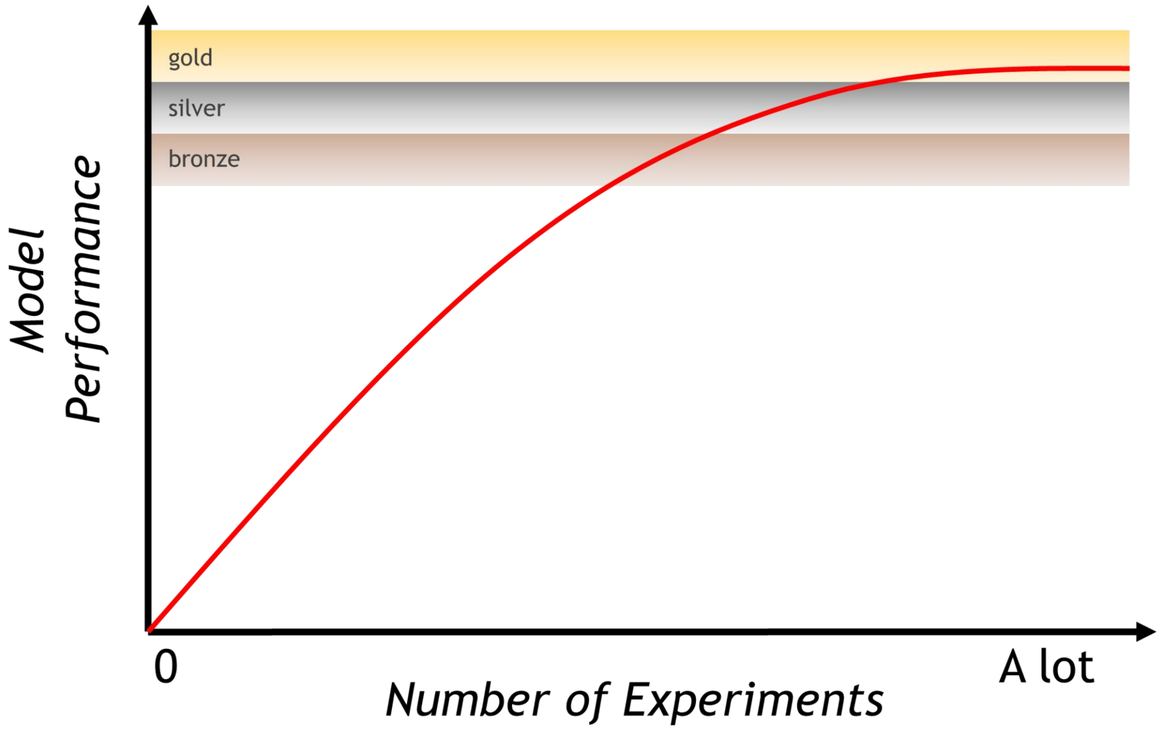
Fast Experimentation

Why Speed Matters

- The more you experiment …
  - The better your models
  - The more you understand the data
  - The higher your chances of finding a winning idea
- … But experimenting takes time :
  - Optimize your pipeline to be able to experiment more

➔ **Leverage GPU acceleration**

- RAPIDS CuDF for Analytics
  - Pandas, but faster !
  - Polars is also faster on GPU !
- RAPIDS CuML for Machine Learning



RAPIDS

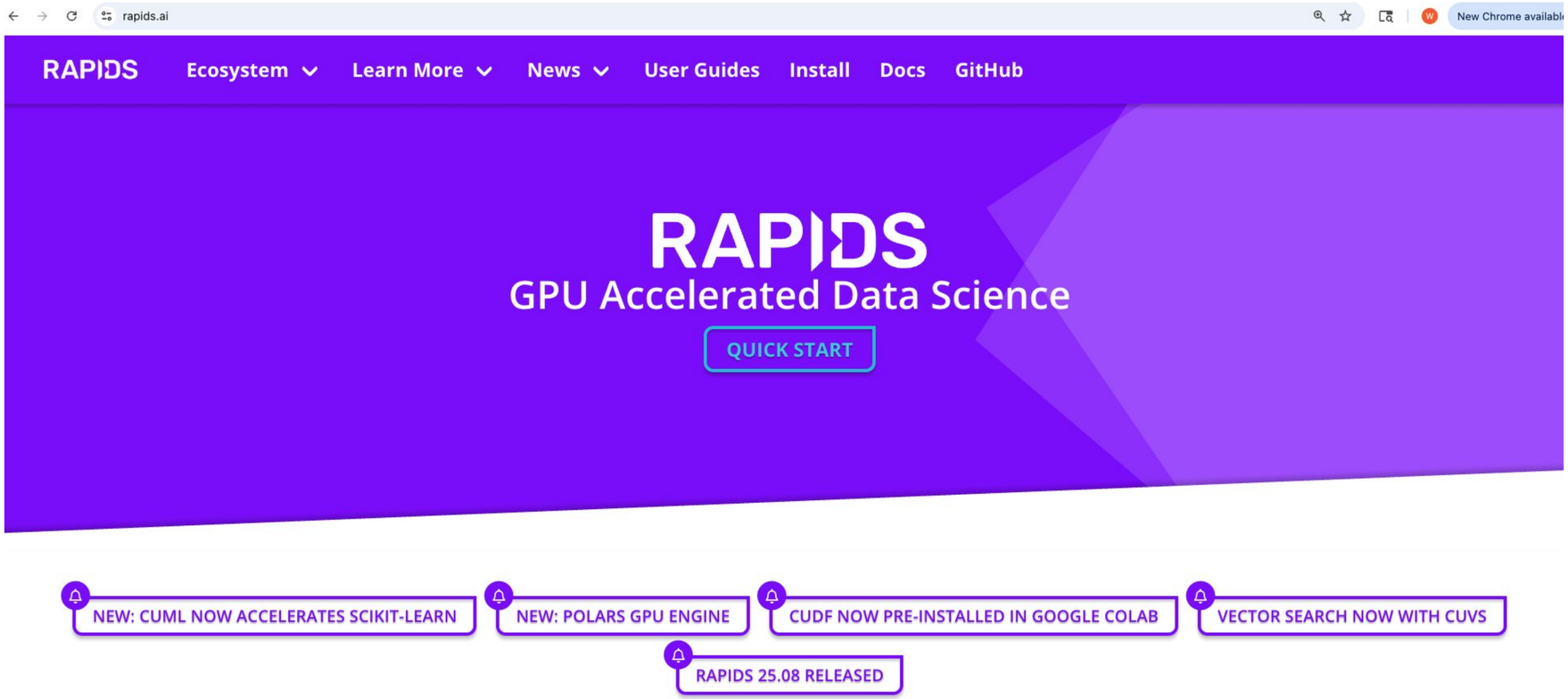
Accelerated with NVIDIA

Summary

Wrapping it All Up

Resources

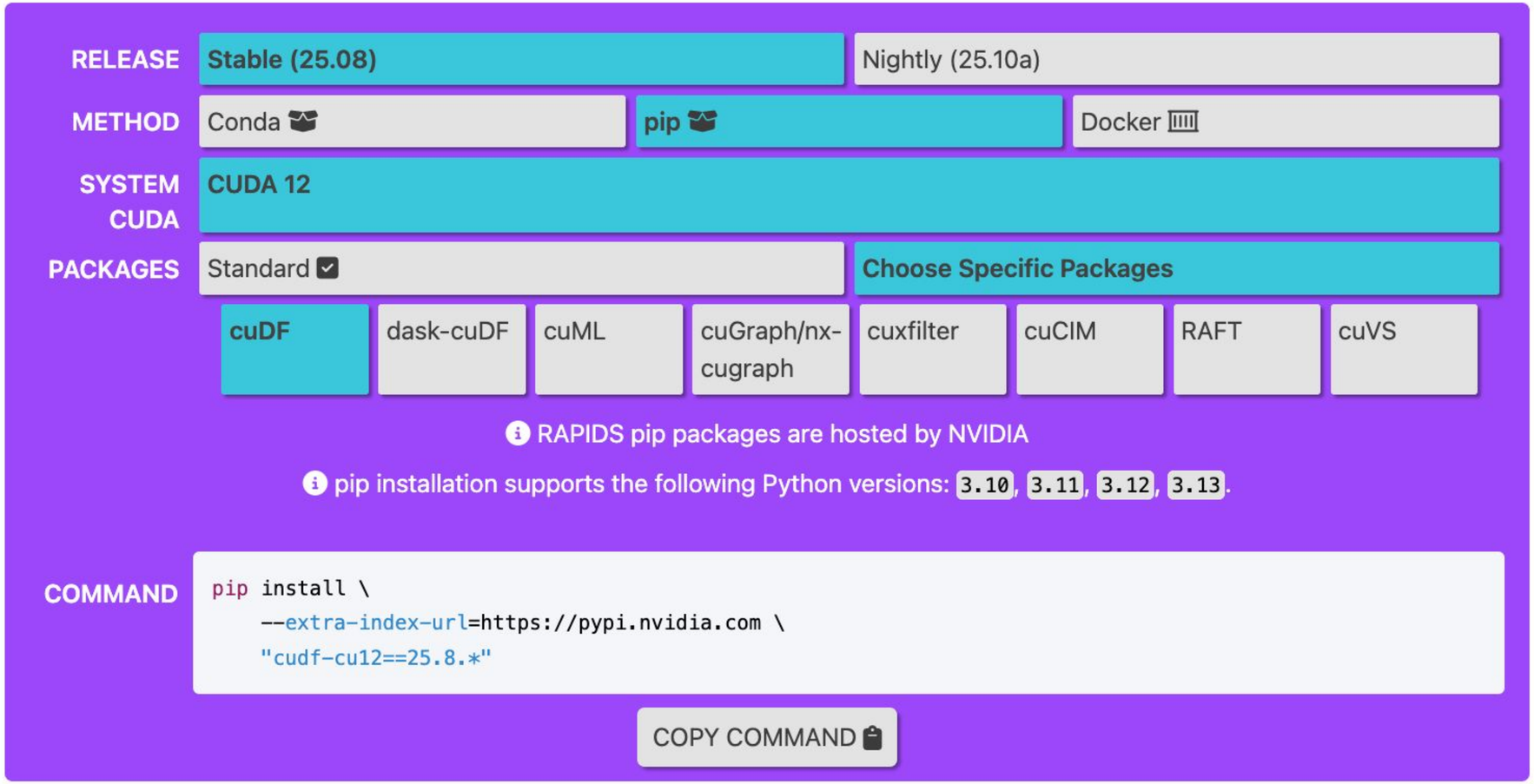
RAPIDS.ai


rapids.ai
New Chrome available
RAPIDS
Ecosystem
Learn More
News
User Guides
Install
Docs
GitHub
RAPIDS
GPU Accelerated Data Science
QUICK START
NEW: CUML NOW ACCELERATES SCIKIT-LEARN
NEW: POLARS GPU ENGINE
CUDF NOW PRE-INSTALLED IN GOOGLE COLAB
VECTOR SEARCH NOW WITH CUVS
RAPIDS 25.08 RELEASED
WHAT IS RAPIDS
RAPIDS provides unmatched speed with familiar APIs that match the most popular PyData libraries. Built on state-of-the-art foundations like NVIDIA CUDA and Apache Arrow, it unlocks the speed of GPUs with code you already know.
Jump to About Section ↗
WHY USE RAPIDS
RAPIDS allows fluid, creative interaction with data for everyone from BI users to AI researchers on the cutting edge. GPU acceleration means less time and less cost moving data and training models. Jump to RAPIDS Use Cases ↗
OPEN SOURCE ECOSYSTEM
RAPIDS is Open Source and available on GitHub. Our mission is to empower and advance the open-source GPU data science data engineering ecosystem. Jump to RAPIDS GitHub ↗

docs.rapids.ai/install

Install RAPIDS

Use the selector tool below to select your preferred method, packages, and environment to install RAPIDS. Certain combinations may not be possible and are dimmed automatically.



github.com/rapidsai/cudf

README | Code of conduct | Contributing | Apache-2.0 license | Security

RAPIDS cuDF - GPU DataFrames

📣 cuDF can now be used as a no-code-change accelerator for pandas! To learn more, see here!

cuDF (pronounced "KOO-dee-eff") is a GPU DataFrame library for loading, joining, aggregating, filtering, and otherwise manipulating data. cuDF leverages libcudf, a blazing-fast C++/CUDA dataframe library and the Apache Arrow columnar format to provide a GPU-accelerated pandas API.

You can import `cudf` directly and use it like `pandas`:

```
import cudf

tips_df = cudf.read_csv("https://github.com/plotly/datasets/raw/master/tips.csv")
tips_df["tip_percentage"] = tips_df["tip"] / tips_df["total_bill"] * 100

# display average tip by dining party size
print(tips_df.groupby("size").tip_percentage.mean())
```

Or, you can use cuDF as a no-code-change accelerator for pandas, using `cudf.pandas`. `cudf.pandas` supports 100% of the pandas API, utilizing cuDF for supported operations and falling back to pandas when needed:

```
%load_ext cudf.pandas  # pandas operations now use the GPU!
```

github.com/rapidsai/cuml

README | Code of conduct | Contributing | Apache-2.0 license | Security

RAPIDS cuML - GPU Machine Learning Algorithms

cuML is a suite of libraries that implement machine learning algorithms and mathematical primitives functions that share compatible APIs with other RAPIDS projects.

cuML enables data scientists, researchers, and software engineers to run traditional tabular ML tasks on GPUs without going into the details of CUDA programming. In most cases, cuML's Python API matches the API from scikit-learn.

For large datasets, these GPU-based implementations can complete 10-50x faster than their CPU equivalents. For details on performance, see the cuML Benchmarks Notebook.

As an example, the following Python snippet loads input and computes DBSCAN clusters, all on GPU, using cuDF:

```
import cudf
from cuml.cluster import DBSCAN

# Create and populate a GPU DataFrame
gdf_float = cudf.DataFrame()
gdf_float['0'] = [1.0, 2.0, 5.0]
gdf_float['1'] = [4.0, 2.0, 1.0]
gdf_float['2'] = [4.0, 2.0, 1.0]

# Setup and fit clusters
dbscan_float = DBSCAN(eps=1.0, min_samples=1)
dbscan_float.fit(gdf_float)

print(dbscan_float.labels_)
```

Developer Tools and Resources

Accelerate innovation and growth

Learn more: developer.nvidia.com

Individuals

Software
100s of APIs, models, SDKs, microservices, and early access to NVIDIA tech

Training
Hands-on self-paced courses, instructor-led workshops, and certifications

GPU Sandbox
Approval basis, multi-GPU and multi-node

Learning
Tutorials, self-paced courses, blogs, documentation, code samples

Community
Dedicated developer forums, meetups, hackathons

Ecosystem
GTC, NVIDIA Partner Network

Organizations

Startups
Cloud credits, engineering resources, technology discounts, exposure to VCs

Venture Capital
Deal flow and portfolio support for Venture Capital firms

Higher Education
Teaching kits, training, curriculum co-development, grants

ISVs and SIs
Engineering guidance, discounts, marketing opportunities

Research
Grant programs, collaboration opportunities

Enterprises
Tailored developer training, skills certification, technical support

Thank you!