



# CS116 – LẬP TRÌNH PYTHON CHO MÁY HỌC

Bài 06

## Advanced Feature Engineering

TS. Nguyễn Vinh Tiệp



# NỘI DUNG

- ❑ Polars
- ❑ Rapid
- ❑ Tại sao phải lựa chọn đặc trưng
- ❑ Kỹ thuật lựa chọn đặc trưng
- ❑ Công cụ lựa chọn đặc trưng



# NỘI DUNG

- ❑ **Polars**
- ❑ Rapid
- ❑ Tại sao phải lựa chọn đặc trưng
- ❑ Kỹ thuật lựa chọn đặc trưng
- ❑ Công cụ lựa chọn đặc trưng



# Polars

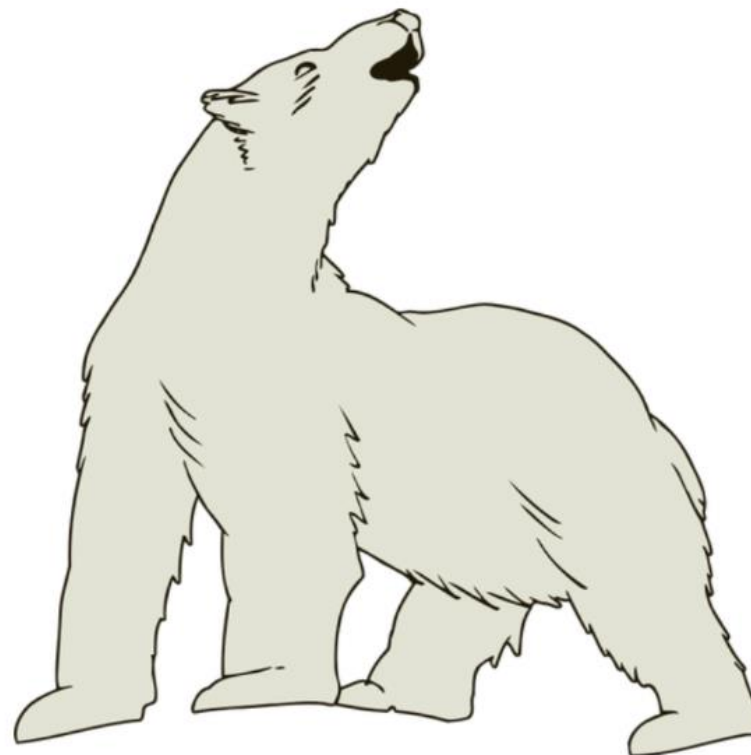
- ❑ **Tại sao chọn Polar**
- ❑ Cách sử dụng Polars
- ❑ So sánh Polars và Pandas



# Tại sao chọn Polars?



>





# Tại sao chọn Polars?

Tại sao Polars ? ●

● Hỗ trợ hoạt động song song

● Polars được thiết kế để tận dụng **đa luồng** và **SIMD** (Single Instruction, Multiple Data)  
→ **Hỗ trợ song song**: Tổng hợp, lọc, chuyển đổi

● Pandas cần **thư viện bên ngoài** như Dask → Yêu cầu thiết lập bổ sung

● Biểu diễn dữ liệu cột

● Polars sử dụng **Apache Arrow** để phân tích cột trong bộ nhớ → **Tăng tốc** thời gian tải dữ liệu, **giảm** bộ nhớ, tăng tốc tính toán

● Pandas Sử dụng biểu diễn dựa trên hàng có thể dẫn đến nhiều lần **bỏ lỡ bộ nhớ cache** hơn và **thực thi chậm hơn** cho một số thao tác nhất định.



# Tại sao chọn Polars?

Tại sao Polars ?

- Tối ưu hóa bộ nhớ

- Polars sử dụng **các kỹ thuật** tối ưu hóa bộ nhớ **khác nhau**. Hỗ trợ đánh giá **eager** và **lazy**

- Pandas **chỉ** hỗ trợ đánh giá **eager**

- Các thuật toán được tối ưu hóa

- Các cực thực hiện các thuật toán hiệu quả cho các hoạt động phổ biến: nối, tổng hợp theo nhóm, sắp xếp,...

- Các thuật toán được thiết kế để tận dụng định dạng dữ liệu cột và xử lý song song nhanh hơn Pandas trong thời gian thực thi



# Tại sao chọn Lazy API Polars?

- ❑ Với Lazy API, Polars không chạy từng dòng truy vấn mà thay vào đó xử lý truy vấn đầy đủ từ đầu đến cuối

Cho phép Polars áp dụng  
Tối ưu hóa Truy vấn Tự  
động với trình tối ưu hóa  
truy vấn

Cho phép bạn làm việc với  
tập dữ liệu lớn hơn bộ nhớ  
bằng cách phát trực tuyến

Có thể bắt lỗi lược đồ trước  
khi xử lý dữ liệu





# Mục tiêu của Polars

Sử dụng tất cả các lõi có sẵn trên máy của bạn

Tối ưu hóa các truy vấn để giảm phân bổ công việc / bộ nhớ không cần thiết

Giảm thiểu tranh chấp song song

Xử lý dữ liệu theo khối



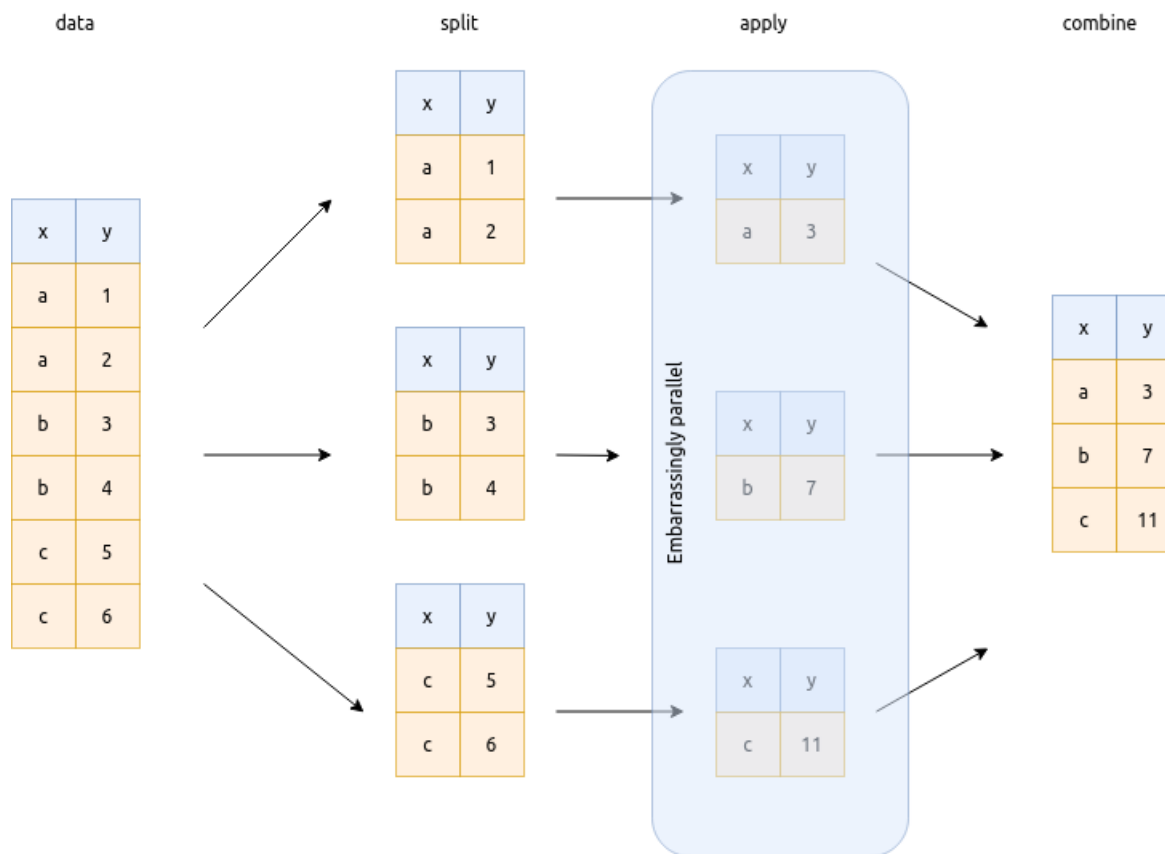
# Polars

- ❑ Tại sao chọn Polar
- ❑ **Cách sử dụng Polars**
- ❑ So sánh Polars và Pandas



# Biểu diễn dữ liệu cột

- Cách tiếp cận đa luồng trong hoạt động tổng hợp theo nhóm






# Polars Cheat Sheet

[Polars Cheat Sheet](#)

[Pandas Cheat Sheet](#)



## Polars Cheat Sheet

[Open in Colab](#)

### General

**Install**  
`pip install polars`

**Import**  
`import polars as pl`


### Creating/reading DataFrames

**Create DataFrame**

nrs	names	random	groups
1	"foo"	0.3	"A"
2	"ham"	0.7	"A"
3	"spam"	0.1	"B"
null	"egg"	0.9	"C"
5	null	0.6	"B"

```
df = pl.DataFrame({
    "nrs": [1, 2, 3, None, 5],
    "names": ["foo", "ham", "spam", "egg", None],
    "random": [0.3, 0.7, 0.1, 0.9, 0.6],
    "groups": ["A", "A", "B", "C", "B"],
})
```

### Subset Observations - rows



Filter: Extract rows that meet logical criteria.

```
df.filter(pl.col("random") > 0.5)
df.filter(
    (pl.col("groups") == "B")
    & (pl.col("random") > 0.5)
)
```

**Sample**

```
# Randomly select fraction of rows.
df.sample(frac=0.5)


# Randomly select n rows.
df.sample(n=2)
```

**Select first and last rows**

```
# Select first n rows
df.head(n=2)

# Select last n rows.
df.tail(n=2)
```

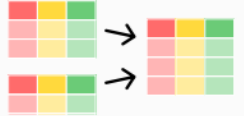
### Subset Variables - columns



Select multiple columns with specific names


```
df.select(["nrs", "names"])
```

### Reshaping Data – Change layout, sorting, renaming




Append rows of DataFrames

```
pl.concat([df, df2])
```



Append columns of DataFrames

```
pl.concat([df, df3], how="horizontal")
```



Gather columns into rows

```
df.melt(
    id_vars="nrs",
    value_vars=["names", "groups"]
)
```


### Summarize Data

Count number of rows with each unique value of variable  
`df["groups"].value_counts()`

# of rows in DataFrame  
`len(df)`  
# or  
`df.height`

Tuple of # of rows, # of columns in DataFrame  
`df.shape`

# of distinct values in a column  
`df["groups"].n_unique()`



Basic descriptive and statistics for each column

```
df.describe()
```

**Aggregation functions**

```
df.select(
    [
        # Sum values
        pl.sum("random").alias("sum"),
        # Minimum value
        pl.min("random").alias("min"),
        # Maximum value
        pl.max("random").alias("max"),
    ]
)
```



# Toán tử cơ bản

## Polars user guide

```
df_numerical = df.select(  
    [  
        (pl.col("nrs") + 5).alias("nrs + 5"),  
        (pl.col("nrs") - 5).alias("nrs - 5"),  
        (pl.col("nrs") * pl.col("random")).alias("nrs * random"),  
        (pl.col("nrs") / pl.col("random")).alias("nrs / random"),  
    ]  
)  
print(df_numerical)
```

Numerical

```
df_logical = df.select(  
    [  
        (pl.col("nrs") > 1).alias("nrs > 1"),  
        (pl.col("random") <= 0.5).alias("random < .5"),  
        (pl.col("nrs") != 1).alias("nrs != 1"),  
        (pl.col("nrs") == 1).alias("nrs == 1"),  
        ((pl.col("random") <= 0.5) & (pl.col("nrs") > 1)).alias("and_expr"), # and  
        ((pl.col("random") <= 0.5) | (pl.col("nrs") > 1)).alias("or_expr"), # or  
    ]  
)  
print(df_logical)
```

Logical



# Hàm

```
df_all = df.select([pl.col("*")])

# Is equivalent to
df_all = df.select([pl.all()])
print(df_all)
```

```
df_samename = df.select([pl.col("nrs") + 5])
print(df_samename)
```

```
df_alias = df.select(
    [
        (pl.col("nrs") + 5).alias("nrs + 5"),
        (pl.col("nrs") - 5).alias("nrs - 5"),
    ]
)
print(df_alias)
```

## Column Selection

```
df_alias = df.select(
    [
        pl.col("names").n_unique().alias("unique"),
        pl.approx_unique("names").alias("unique_approx"),
    ]
)
print(df_alias)
```

## Count unique values

```
df_conditional = df.select(
    [
        pl.col("nrs"),
        pl.when(pl.col("nrs") > 2)
            .then(pl.lit(True))
            .otherwise(pl.lit(False))
            .alias("conditional"),
    ]
)
print(df_conditional)
```

## Conditionals



# Basic Aggregations

```
q = (  
    dataset.lazy()  
    .groupby("first_name")  
    .agg(  
        [  
            pl.count(),  
            pl.col("gender"),  
            pl.first("last_name"),  
        ]  
    )  
    .sort("count", descending=True)  
    .limit(5)  
)
```

```
df = q.collect()  
print(df)
```

```
q = (  
    dataset.lazy()  
    .groupby("state")  
    .agg(  
        [  
            (pl.col("party") == "Anti-Administration").sum().alias("anti"),  
            (pl.col("party") == "Pro-Administration").sum().alias("pro"),  
        ]  
    )  
    .sort("pro", descending=True)  
    .limit(5)  
)
```

```
df = q.collect()  
print(df)
```



# Filtering

```
def compute_age() -> pl.Expr:
    return date(2021, 1, 1).year - pl.col("birthday").dt.year()

def avg_birthday(gender: str) -> pl.Expr:
    return (
        compute_age()
        .filter(pl.col("gender") == gender)
        .mean()
        .alias(f"avg {gender} birthday")
    )
```

```
q = (
    dataset.lazy()
    .groupby(["state"])
    .agg(
        [
            avg_birthday("M"),
            avg_birthday("F"),
            (pl.col("gender") == "M").sum().alias("# male"),
            (pl.col("gender") == "F").sum().alias("# female"),
        ]
    )
    .limit(5)
)

df = q.collect()
print(df)
```





# Sorting

```
def get_person() -> pl.Expr:
    return pl.col("first_name") + pl.lit(" ") + pl.col("last_name")

q = (
    dataset.lazy()
    .sort("birthday", descending=True)
    .groupby(["state"])
    .agg(
        [
            get_person().first().alias("youngest"),
            get_person().last().alias("oldest"),
        ]
    )
    .limit(5)
)

df = q.collect()
print(df)
```



# Missing data

```
fill_literal_df = (  
    df.with_columns(  
        pl.col("col2").fill_null(  
            pl.lit(2),  
        ),  
    ),  
)  
print(fill_literal_df)
```

```
fill_forward_df = df.with_columns(  
    pl.col("col2").fill_null(strategy="forward"),  
)  
print(fill_forward_df)
```

```
fill_median_df = df.with_columns(  
    pl.col("col2").fill_null(pl.median("col2")),  
)  
print(fill_median_df)
```

```
fill_interpolation_df = df.with_columns(  
    pl.col("col2").interpolate(),  
)  
print(fill_interpolation_df)
```



# Hàm do người dùng xác định: map

- ❑ Không bao giờ sử dụng map trong bối cảnh nhóm, trừ khi bạn biết bạn cần nó và biết bạn đang làm gì

```
df = pl.DataFrame(  
    {  
        "keys": ["a", "a", "b"],  
        "values": [10, 7, 1],  
    }  
)  
  
out = df.groupby("keys", maintain_order=True).agg(  
    [  
        pl.col("values").map(lambda s: s.shift()).alias("shift_map"),  
        pl.col("values").shift().alias("shift_expression"),  
    ]  
)  
print(df)
```

shape: (2, 3)

keys	shift_map	shift_expression
---	---	---
str	list[i64]	list[i64]
a	[null, 10]	[null, 10]
b	[7]	[null]



# Hàm do người dùng xác định: apply

```
out = df.groupby("keys", maintain_order=True).agg([
    pl.col("values").apply(lambda s: s.shift()).alias("shift_map"),
    pl.col("values").shift().alias("shift_expression"),
])
print(out)
```

shape: (2, 3)

keys	shift_map	shift_expression
---	---	---
str	list[i64]	list[i64]
a	[null, 10]	[null, 10]
b	[null]	[null]



# Kết hợp nhiều cột

```
out = df.select(  
    [  
        pl.struct(["keys", "values"])  
        .apply(lambda x: len(x["keys"]) + x["values"])  
        .alias("solution_apply"),  
        (pl.col("keys").str.lengths() + pl.col("values")).alias("solution_expr"),  
    ]  
)  
print(out)
```

shape: (3, 2)

solution_apply	solution_expr
---	---
i64	i64
11	11
8	8
2	2



# Polars

- ❑ Tại sao chọn Polar
- ❑ Cách sử dụng Polars
- ❑ **So sánh Polars và Pandas**



# So sánh Pandas và Polars

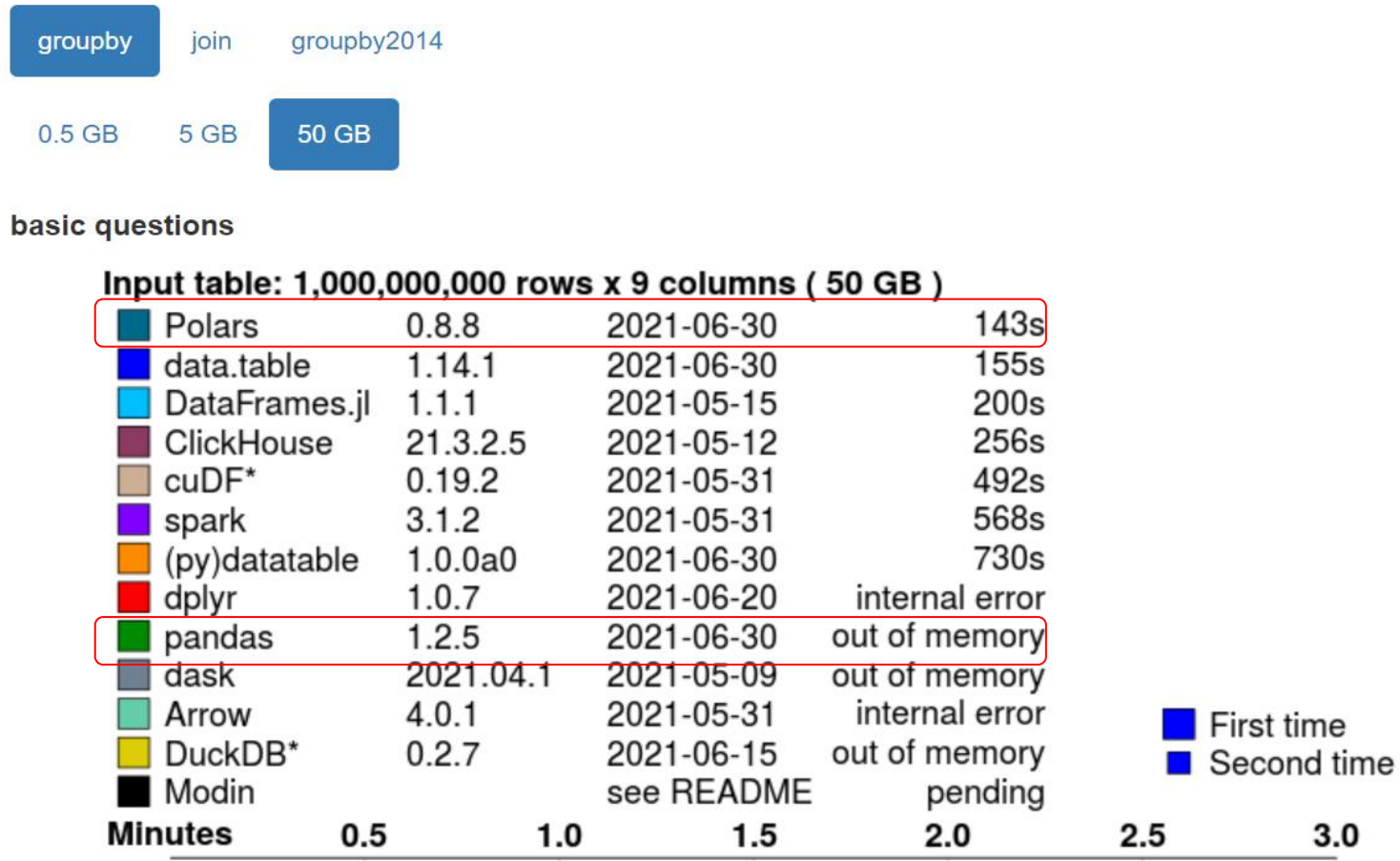
Đọc & Viết file (.csv)

Library	Read 1 GB File	Write 1 GB File	Read 50 GB File and Filter
Pandas	13.5s	16.1s	9min 53s
Polars	350ms	951ms	1min 13s

<https://www.shipyardapp.com/blog/pandas-vs-polars/>



# So sánh Pandas và Polars



<https://h2oai.github.io/db-benchmark/>





# Pandas 2.0 (new)

- ❑ Di chuyển ra khỏi cách nó đại diện cho dữ liệu từ numpy sang Apache Arrow và sẽ tăng tốc nhiều tác vụ Pandas (ví dụ tải và lưu tệp csv,...)

Operation	Time with NumPy	Time with Arrow	Speed up
read parquet (50Mb)	141 ms	87 ms	1.6x
mean (int64)	2.03 ms	1.11 ms	1.8x
mean (float64)	3.56 ms	1.73 ms	2.1x
endswith (string)	471 ms	14.9 ms	31.6x

<https://datapythonista.me/blog/pandas-20-and-the-arrow-revolution-part-i>



# NỘI DUNG

- ❑ Polars
- ❑ **Rapid**
- ❑ Tại sao phải lựa chọn đặc trưng
- ❑ Kỹ thuật lựa chọn đặc trưng
- ❑ Công cụ lựa chọn đặc trưng



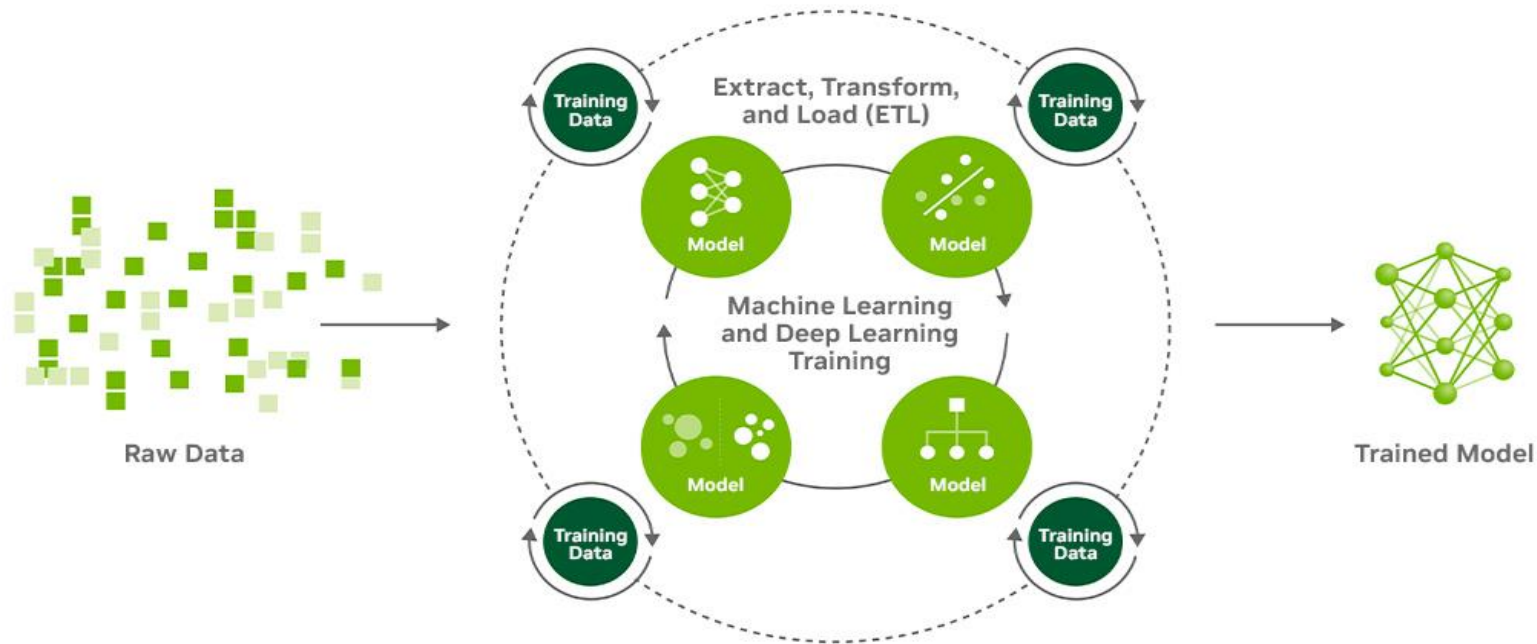
# Rapid

- ❑ **Giới thiệu về Rapid**
- ❑ Cách sử dụng Rapid
- ❑ So sánh Rapid và Pandas



# Rapid

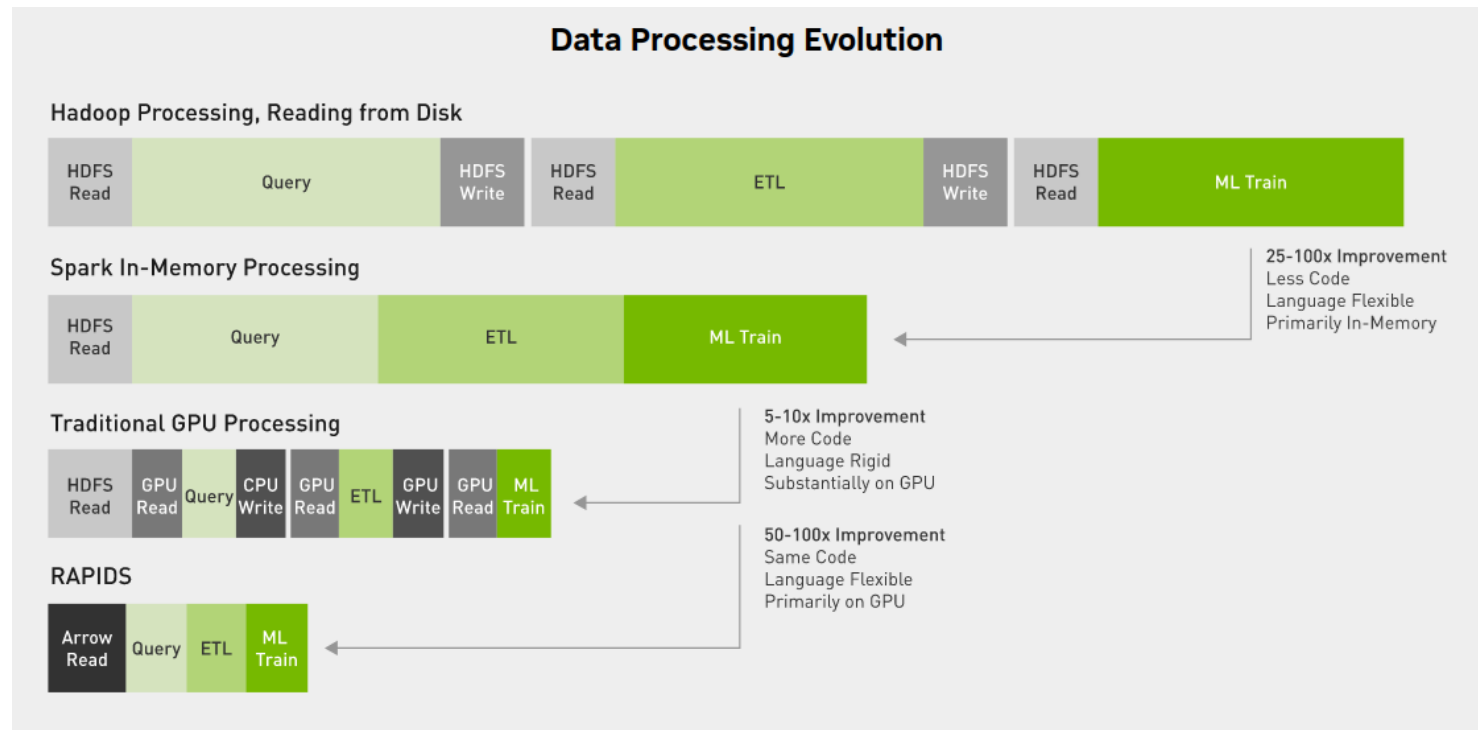
- ❑ Bộ thư viện phần mềm nguồn mở RAPIDS nhằm mục đích cho phép thực thi các đường ống phân tích và khoa học dữ liệu đầu cuối hoàn toàn trên GPU.





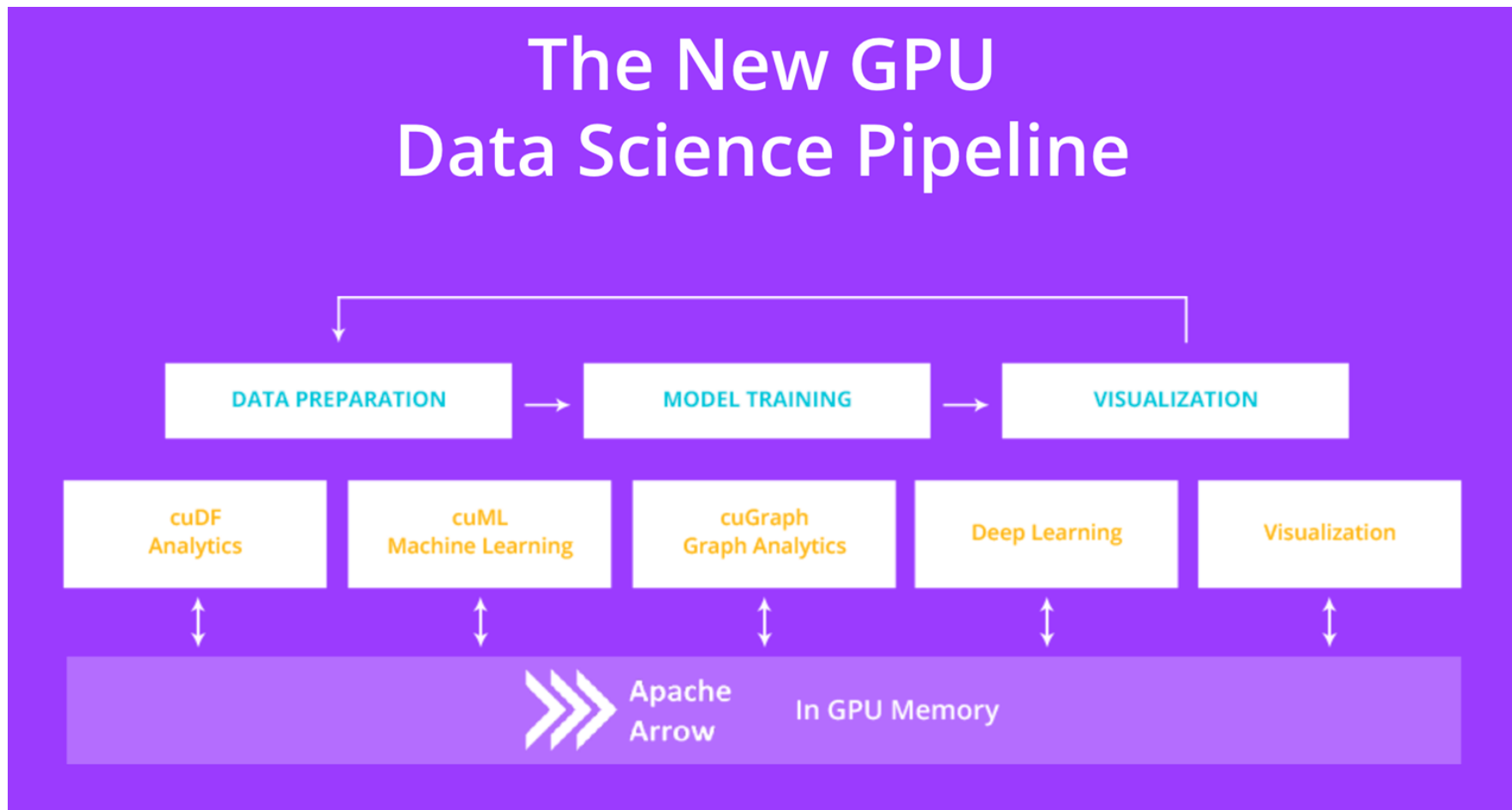
# Tại sao chọn Rapids

- ❑ Chạy toàn bộ quy trình làm việc khoa học dữ liệu với tính toán GPU tốc độ cao và song song hóa việc tải dữ liệu, thao tác dữ liệu và học máy để có đường ống khoa học dữ liệu đầu cuối nhanh hơn 50 lần





# Open GPU Data Science Pipeline

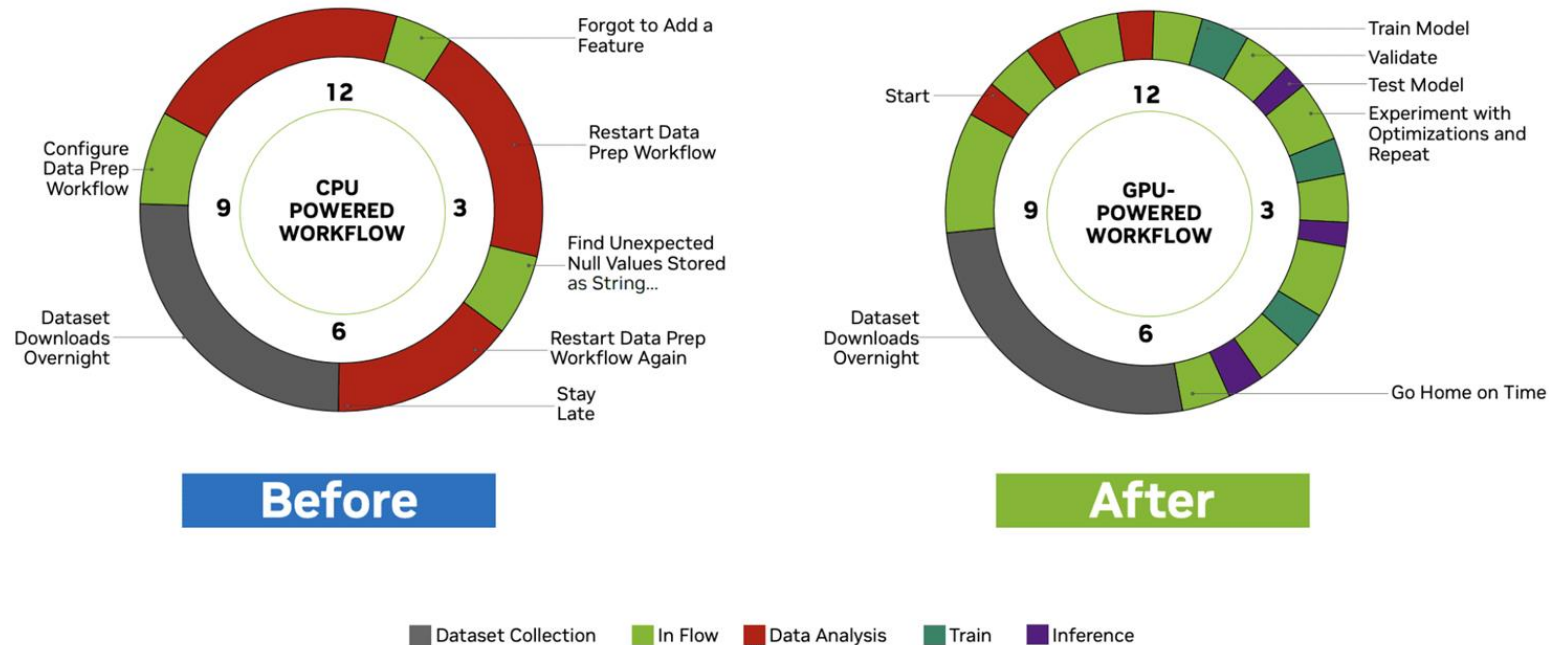


<https://github.com/rapidsai/cudf>



# Rapid Libraries

- ❑ [cuDF](#): Thư viện Python GPU DataFrame (được xây dựng trên định dạng bộ nhớ cột Apache Arrow) để tải, nối, tổng hợp, lọc và thao tác với dữ liệu dạng bảng.
- ❑ [Dask](#)
- ❑ [Dask-cuDF](#)





# cuDF and Dask-cuDF

- ❑ Nếu quy trình làm việc của bạn đủ nhanh trên một GPU duy nhất hoặc dữ liệu của bạn thoải mái phù hợp với bộ nhớ trên một GPU duy nhất, bạn sẽ muốn sử dụng cuDF.
- ❑ Nếu bạn muốn phân phối quy trình làm việc của mình trên nhiều GPU, hãy có nhiều dữ liệu hơn mức bạn có thể vừa với bộ nhớ trên một GPU duy nhất mà bạn muốn sử dụng Dask-cuDF



Scaling with RAPIDS + Dask  
Easy Multi-GPU for Python Programmers

- Distributed extensions with familiar APIs for DataFrames and Arrays
- Scale from laptop to supercomputer scales  
- tested up to 1024 GPUs
- Deploy on any cloud service or Kubernetes in minutes
- Integrate cuDF, cuPy, cuML, cuGraph, and XGBoost with a common framework
- Easily switch between CPU and GPU backends

```
# Start by telling Dask to use the GPU backend
with dask.config.set({"dataframe.backend": "cudf"}):
    ddf_s = dd.read_parquet('stores.parquet')

    ddf_p = dd.read_parquet("purchases.parquet")
    ddf_p["total"] = ddf_p.price * ddf_p.quantity

# Combine the two dataframes
ddf_join = ddf_p.merge(res,
                      on=["id"], how="inner")
ddf_join = ddf_join.set_index("key")
```





# Rapid Cheat Sheet

## Link Cheat Sheet

### TRANSFORM

Alter the information and structure of DataFrames

```
def func(foo, bar):  
    for i, f in enumerate(foo):  
        bar[i] = f + 1 - Kernel definition to use in apply_rows() function.
```

`cudf.concat([df1, df2])` - Append a DataFrame to another DataFrame.

`df.drop(1)` - Remove row with index equal to 1.

`df.drop([1,2])` - Remove rows with index equal to 1 and 2.

`df.drop('foo', axis=1)` - Remove column foo.

`df.dropna()` - Remove rows with one or more missing values.

`df.dropna(subset='foo')` - Remove rows with a missing value in column foo.

`df.fillna(-1)` - Replace any missing value with a default.

`df.fillna({'foo': -1})` - Replace a missing value in column foo with a default.

`df1.join(df2)` - Join with a DataFrame on index.

`df1.merge(df2, on='foo', how='inner')` - Perform an inner join with a DataFrame on column foo.

`df1.merge(df2, left_on='foo', right_on='bar', how='left')` - Perform a left outer join with a DataFrame on different keys.

### STRING

Operate on string columns on GPU.

`ser.str.contains('foo')` - Check if Series of strings contains foo.

`ser.str.contains('foo[a-z]+')` - Check if Series of strings contains words starting with foo.

`ser.str.extract('(foo)')` - Retrieve regex groups matching pattern in Series of strings.

`ser.str.extract('[a-z]+flow (\d)')` - Retrieve IDs of dataflows, workflows, etc., in Series of strings.

`ser.str.findall('[a-z]+flow)')` - Retrieve all instances of words like dataflow, workflow, etc.

`ser.str.len()` - Find the total length of a string.

`ser.str.lower()` - Cast all the letters in a string to lowercase characters.

`ser.str.match('[a-z]+flow')` - Check if every element matches the pattern.

`ser.str.ngrams_tokenize(n=2, separator='_')` - Generate all bi-grams from a string separated by underscore.

`ser.str.pad(width=10)` - Make every string of equal length.

`ser.str.pad(width=10, side='both', fillchar='$')` - Make every string of equal length with word centered and padded with dollar signs.



# So sánh cuDF và Pandas

- ❑ Tham khảo: [notebook](#)
- ❑ Kích thước dataset: (887379 mẫu, 22 trường)

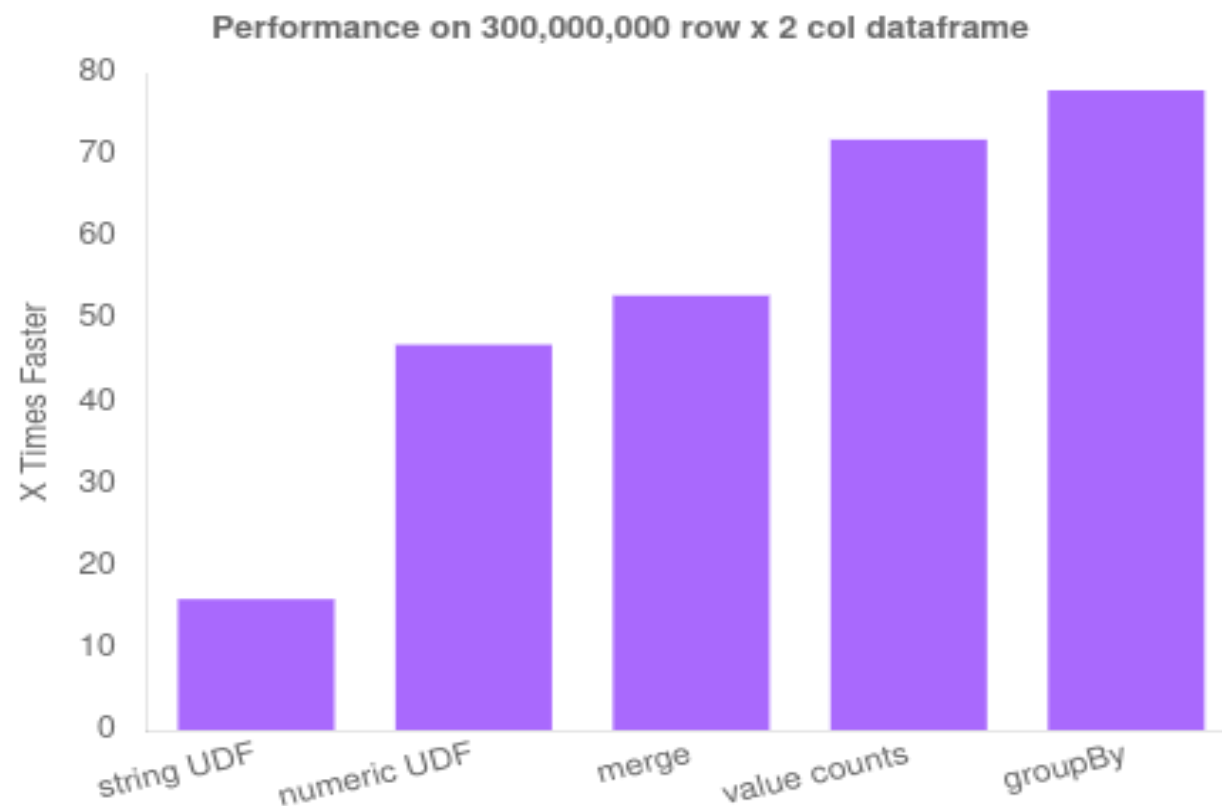
Method	Pandas	cuDF
Loading data	2.42 s	1.52 s
Sorting value	149 ms	37.8 ms
Selection by label	1.4 ms	1.82 ms
Boolean indexing	182 ms	34.7 ms
Query API	189 ms	178 ms
Isin method	49.9 ms	44.2 ms
Fillna	380 ms	29.8 ms

Method	Pandas	cuDF
Statistics operation	5.7 ms	4.51 ms
Apply map	211 ms	1.84 ms
Histogramming	68.4 ms	23.5 ms
Concat	1.41 ms	2.28 ms
Join	15.7 ms	9.05 ms
Groupby	792 ms	270 ms
Time series	213 ms	330 ms



# Faster Pandas with CuDF

- ❑ Benchmark on AMD EPYC 7642 (using 1x 2.3GHz CPU core) w/ 512GB and NVIDIA A100 80GB (1x GPU) w/ pandas v1.5 and cuDF v23.02
- ❑ Tham khảo: [notebook](#)

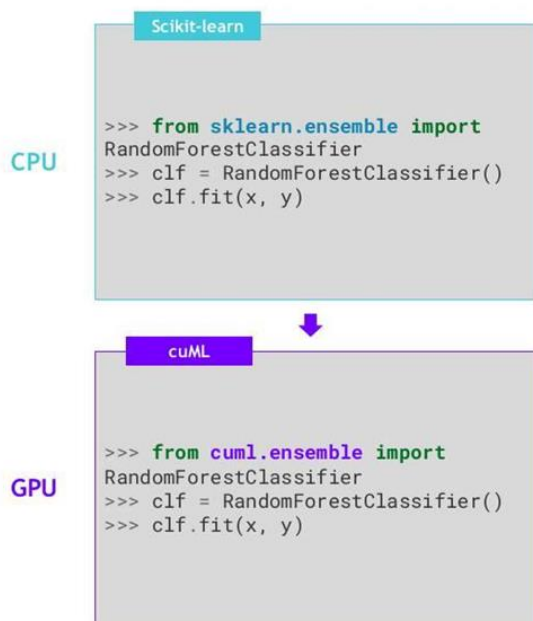




# cuML: Accelerated ML with a scikit-learn API

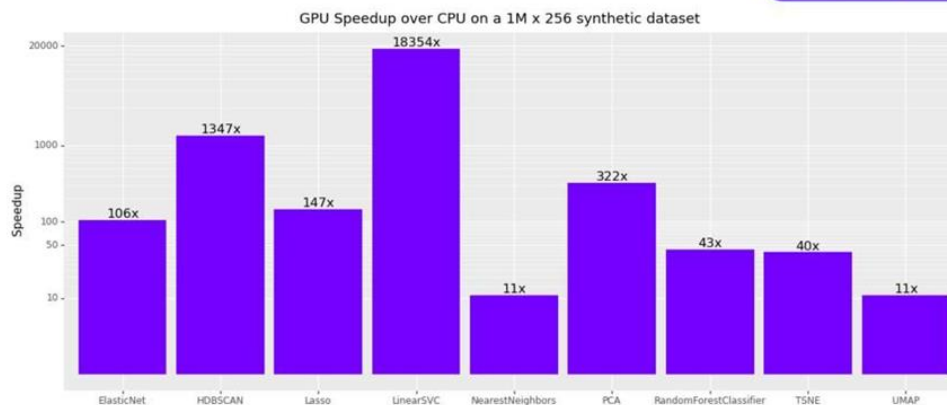
cuML

Accelerated Machine Learning with a scikit-learn API



40+ GPU-Accelerated Algorithms & Growing

Performance  
maximized on large  
in-memory datasets

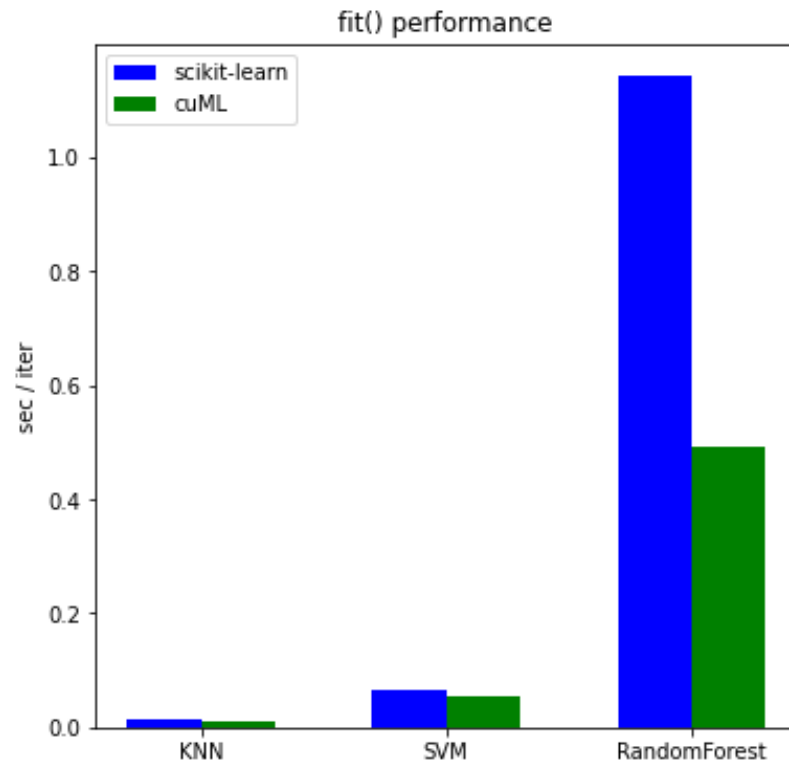


Time Series	Classification	Regression	Clustering	Preprocessing
Cross Validation	Tree Models	Dimensionality Reduction	Explainability	

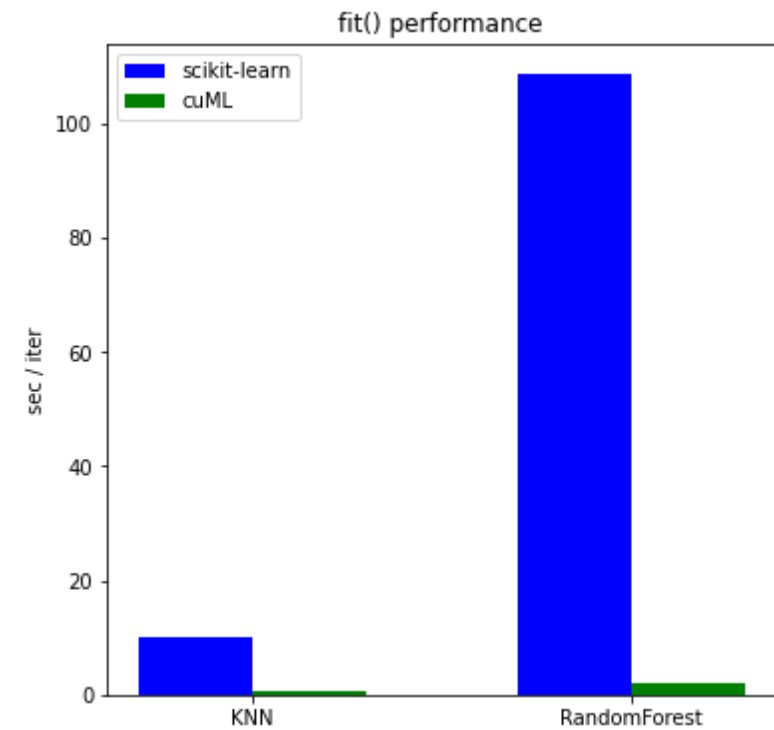
A100 GPU vs. 2x Intel Xeon E5-2698 CPUs (80 logical cores)  
cuML 23.02, scikit-learn 1.2, umap-learn 0.5.3



# cuML & scikit-learn



data Titanic



data Home Credit Default Risk



# Accelerated XGBoost

## Accelerated XGBoost and Inference for Trees

“XGBoost is All You Need” - Bojan Tunguz, 4x Kaggle Grandmaster



- One line of code change to unlock up to 20x speedups with GPUs
- Scalable to the world’s largest datasets with Dask and PySpark
- Built-in SHAP support for model explainability
- Deployable with Triton for lightning-fast inference in production
- Triton supports LightGBM and Random Forests as well as XGBoost for inference

**RAPIDS** ❤️ **dmlc XGBoost**



# QUIZ & QUESTIONS