Tutorial on reading large datasets

100 million rows and 10 columns

Different packages have their own way of reading data. The methods explored in the notebook:

- Pandas
- Dask
- Rapids

Apart from methods of reading data from the raw csv files, it is also common to convert the dataset into another format which uses lesser disk space, is smaller in size and/or can be read faster for subsequent reads. The file types explored in the notebook:

- csv
- feather
- hdf5
- parquet
- pickle

In [1]:
```
import pandas as pd
import dask.dataframe as dd
```

```
/opt/conda/lib/python3.10/site-packages/scipy/__init__.py:146: UserWarning: A NumPy version >=1.16.5 and <1.23.0 is required for this version of SciPy (detected version 1.24.3
  warnings.warn(f"A NumPy version >={np_minversion} and <{np_maxversion}"
```

Method: Pandas

Pandas is probably the most popular method of reading datasets. It has a lot of options, flexibility and functions for reading and processing data.

One of the challenges with using pandas for reading large datasets is it's conservative nature while infering data types of the columns of a dataset often resulting in unnecessary large memory usage for the pandas dataframe. You can pre-define optimal data types of the columns (based on prior knowledge or sample inspection) and provide it explicitly while reading the dataset.

This is the method used in the official starter notebook of the RiiiD competition as well.

Documentation: https://pandas.pydata.org/docs/

In [2]:
```
%%time

dtypes = {
    "row_id": "int64",
    "timestamp": "int64",
    "user_id": "int32",
    "content_id": "int16",
    "content_type_id": "boolean",
    "task_container_id": "int16",
    "user_answer": "int8",
    "answered_correctly": "int8",
    "prior_question_elapsed_time": "float32",
    "prior_question_had_explanation": "boolean"
}

data = pd.read_csv("../input/riiid-test-answer-prediction/train.csv", dtype=dtypes)

print("Train size:", data.shape)
```

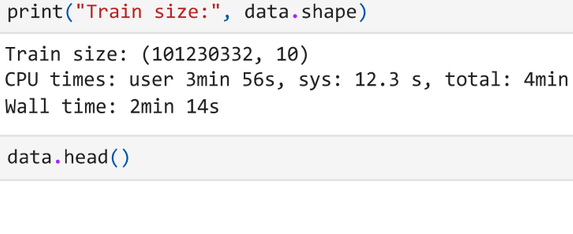
```
Train size: (101230332, 10)
CPU times: user 3min 6s, sys: 5.52 s, total: 3min 11s
Wall time: 4min 9s
```

In [3]:
```
data.head()
```

Out[3]:

	row_id	timestamp	user_id	content_id	content_type_id	task_container_id	user_answer	answered_correctly	prior_question_elapsed_time	prior_question_had_explanation
0	0	0	115	5692	False	1	3	1	NaN	<NA>
1	1	56943	115	5716	False	2	2	1	37000.0	False
2	2	118363	115	128	False	0	0	1	55000.0	False
3	3	131167	115	7860	False	3	0	1	19000.0	False
4	4	137965	115	7922	False	4	1	1	11000.0	False

Method: Dask



Dask provides a framework to scale pandas workflows natively using a parallel processing architecture. For those of you who have used Spark, you will find an uncanny similarity between the two.

Documentation: https://docs.dask.org/en/latest/

In [4]:
```
%%time

dtypes = {
    "row_id": "int64",
    "timestamp": "int64",
    "user_id": "int32",
    "content_id": "int16",
    "content_type_id": "boolean",
    "task_container_id": "int16",
    "user_answer": "int8",
    "answered_correctly": "int8",
    "prior_question_elapsed_time": "float32",
    "prior_question_had_explanation": "boolean"
}

data = dd.read_csv("../input/riiid-test-answer-prediction/train.csv", dtype=dtypes).compute()

print("Train size:", data.shape)
```

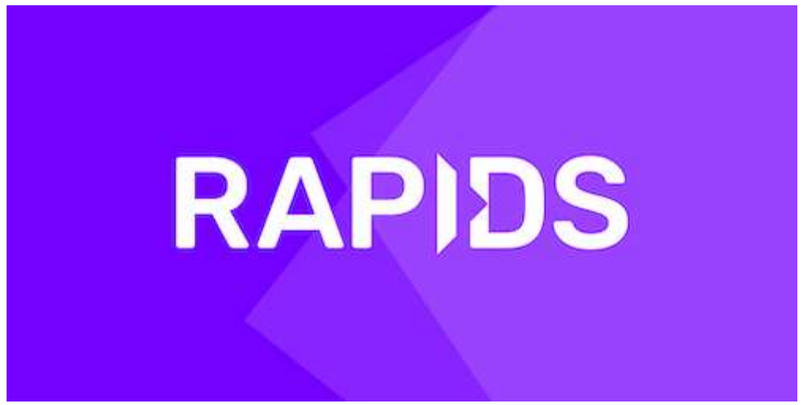
```
Train size: (101230332, 10)
CPU times: user 3min 56s, sys: 12.3 s, total: 4min 8s
Wall time: 2min 14s
```

In [5]:
```
data.head()
```

Out[5]:

	row_id	timestamp	user_id	content_id	content_type_id	task_container_id	user_answer	answered_correctly	prior_question_elapsed_time	prior_question_had_explanation
0	0	0	115	5692	False	1	3	1	NaN	<NA>
1	1	56943	115	5716	False	2	2	1	37000.0	False
2	2	118363	115	128	False	0	0	1	55000.0	False
3	3	131167	115	7860	False	3	0	1	19000.0	False
4	4	137965	115	7922	False	4	1	1	11000.0	False

Method: Rapids



Rapids is a great option to scale data processing on GPUs. With a lot of machine learning modelling moving to GPUs, Rapids enables to build end-to-end data science solutions on one or more GPUs.

Documentation: https://docs.rapids.ai/

In [6]:
```
# rapids installation (make sure to turn on GPU)
import sys
!cp ../input/rapids/rapids.0.16.0 /opt/conda/envs/rapids.tar.gz
!cd /opt/conda/envs/ && tar -xzvf rapids.tar.gz > /dev/null
sys.path = ["/opt/conda/envs/rapids/lib/python3.7/site-packages"] + sys.path
sys.path = ["/opt/conda/envs/rapids/lib/python3.7"] + sys.path
sys.path = ["/opt/conda/envs/rapids/lib"] + sys.path

import cudf
```

```
cp: cannot stat '../input/rapids/rapids.0.16.0': No such file or directory
tar (child): rapids.tar.gz: Cannot open: No such file or directory
tar (child): Error is not recoverable: exiting now
tar: Child returned status 2
tar: Error is not recoverable: exiting now
```

In [7]:
```
%%time

data = cudf.read_csv("../input/riiid-test-answer-prediction/train.csv")

print("Train size:", data.shape)
```

```
Train size: (101230332, 10)
CPU times: user 5.76 s, sys: 1.03 s, total: 6.79 s
Wall time: 8.11 s
```

In [8]:
```
data.head()
```

```
/opt/conda/lib/python3.10/contextlib.py:79: FutureWarning: is_datetime64tz_dtype is deprecated and will be removed in a future version. Check `isinstance(dtype, pd.DatetimeTZDtype)` instead.
  return func(*args, **kwds)
/opt/conda/lib/python3.10/site-packages/cudf/core/dtypes.py:975: FutureWarning: is_categorical_dtype is deprecated and will be removed in a future version. Use isinstance(dtype, CategoricalDtype) inst
ead
  return pd_types.is_categorical_dtype(obj)
/opt/conda/lib/python3.10/site-packages/cudf/core/dtypes.py:975: FutureWarning: is_categorical_dtype is deprecated and will be removed in a future version. Use isinstance(dtype, CategoricalDtype) inst
ead
  return pd_types.is_categorical_dtype(obj)
/opt/conda/lib/python3.10/site-packages/cudf/core/column/numerical.py:338: FutureWarning: is_datetime64tz_dtype is deprecated and will be removed in a future version. Check `isinstance(dtype, pd.Datet
imeTZDtype)` instead.
  return string._numeric_to_str_typecast_functions[
/opt/conda/lib/python3.10/site-packages/cudf/core/column/column.py:1567: FutureWarning: is_datetime64tz_dtype is deprecated and will be removed in a future version. Check `isinstance(dtype, pd.Datetim
eTZDtype)` instead.
  elif is_datetime64tz_dtype(dtype):
/opt/conda/lib/python3.10/site-packages/cudf/core/dtypes.py:975: FutureWarning: is_categorical_dtype is deprecated and will be removed in a future version. Use isinstance(dtype, CategoricalDtype) inst
ead
  return pd_types.is_categorical_dtype(obj)
/opt/conda/lib/python3.10/contextlib.py:79: FutureWarning: is_datetime64tz_dtype is deprecated and will be removed in a future version. Check `isinstance(dtype, pd.DatetimeTZDtype)` instead.
  return func(*args, **kwds)
/opt/conda/lib/python3.10/site-packages/cudf/core/column/column.py:1567: FutureWarning: is_datetime64tz_dtype is deprecated and will be removed in a future version. Check `isinstance(dtype, pd.Datetim
eTZDtype)` instead.
  elif is_datetime64tz_dtype(dtype):
/opt/conda/lib/python3.10/site-packages/cudf/core/dtypes.py:975: FutureWarning: is_categorical_dtype is deprecated and will be removed in a future version. Use isinstance(dtype, CategoricalDtype) inst
ead
  return pd_types.is_categorical_dtype(obj)
/opt/conda/lib/python3.10/site-packages/cudf/core/dtypes.py:975: FutureWarning: is_categorical_dtype is deprecated and will be removed in a future version. Use isinstance(dtype, CategoricalDtype) inst
ead
  return pd_types.is_categorical_dtype(obj)
/opt/conda/lib/python3.10/site-packages/cudf/core/column/numerical.py:338: FutureWarning: is_datetime64tz_dtype is deprecated and will be removed in a future version. Check `isinstance(dtype, pd.Datet
imeTZDtype)` instead.
  return string._numeric_to_str_typecast_functions[
/opt/conda/lib/python3.10/site-packages/cudf/core/column/column.py:1567: FutureWarning: is_datetime64tz_dtype is deprecated and will be removed in a future version. Check `isinstance(dtype, pd.Datetim
eTZDtype)` instead.
  elif is_datetime64tz_dtype(dtype):
/opt/conda/lib/python3.10/contextlib.py:79: FutureWarning: is_datetime64tz_dtype is deprecated and will be removed in a future version. Check `isinstance(dtype, pd.DatetimeTZDtype)` instead.
  return func(*args, **kwds)
/opt/conda/lib/python3.10/site-packages/cudf/core/dtypes.py:975: FutureWarning: is_categorical_dtype is deprecated and will be removed in a future version. Use isinstance(dtype, CategoricalDtype) inst
ead
  return pd_types.is_categorical_dtype(obj)
/opt/conda/lib/python3.10/site-packages/cudf/core/column/numerical.py:338: FutureWarning: is_datetime64tz_dtype is deprecated and will be removed in a future version. Check `isinstance(dtype, pd.Datet
imeTZDtype)` instead.
  return string._numeric_to_str_typecast_functions[
/opt/conda/lib/python3.10/site-packages/cudf/core/column/column.py:1567: FutureWarning: is_datetime64tz_dtype is deprecated and will be removed in a future version. Check `isinstance(dtype, pd.Datetim
eTZDtype)` instead.
  elif is_datetime64tz_dtype(dtype):
/opt/conda/lib/python3.10/contextlib.py:79: FutureWarning: is_datetime64tz_dtype is deprecated and will be removed in a future version. Check `isinstance(dtype, pd.DatetimeTZDtype)` instead.
```

```
  return func(*args, **kwds)
/opt/conda/lib/python3.10/site-packages/cudf/core/dtypes.py:975: FutureWarning: is_categorical_dtype is deprecated and will be removed in a future version. Use isinstance(dtype, CategoricalDtype) inst
ead
  return pd_types.is_categorical_dtype(obj)
/opt/conda/lib/python3.10/site-packages/cudf/core/column/numerical.py:338: FutureWarning: is_datetime64tz_dtype is deprecated and will be removed in a future version. Check `isinstance(dtype, pd.Datet
imeTZDtype)` instead.
  return string._numeric_to_str_typecast_functions[
/opt/conda/lib/python3.10/site-packages/cudf/core/column/column.py:1567: FutureWarning: is_datetime64tz_dtype is deprecated and will be removed in a future version. Check `isinstance(dtype, pd.Datetim
eTZDtype)` instead.
  elif is_datetime64tz_dtype(dtype):
/opt/conda/lib/python3.10/contextlib.py:79: FutureWarning: is_datetime64tz_dtype is deprecated and will be removed in a future version. Check `isinstance(dtype, pd.DatetimeTZDtype)` instead.
  return func(*args, **kwds)
/opt/conda/lib/python3.10/site-packages/cudf/core/dtypes.py:975: FutureWarning: is_categorical_dtype is deprecated and will be removed in a future version. Use isinstance(dtype, CategoricalDtype) inst
ead
  return pd_types.is_categorical_dtype(obj)
/opt/conda/lib/python3.10/site-packages/cudf/core/column/numerical.py:338: FutureWarning: is_datetime64tz_dtype is deprecated and will be removed in a future version. Check `isinstance(dtype, pd.Datet
imeTZDtype)` instead.
  return string._numeric_to_str_typecast_functions[
/opt/conda/lib/python3.10/site-packages/cudf/core/column/column.py:1567: FutureWarning: is_datetime64tz_dtype is deprecated and will be removed in a future version. Check `isinstance(dtype, pd.Datetim
eTZDtype)` instead.
  elif is_datetime64tz_dtype(dtype):
/opt/conda/lib/python3.10/contextlib.py:79: FutureWarning: is_datetime64tz_dtype is deprecated and will be removed in a future version. Check `isinstance(dtype, pd.DatetimeTZDtype)` instead.
  return func(*args, **kwds)
```

Out[8]:

	row_id	timestamp	user_id	content_id	content_type_id	task_container_id	user_answer	answered_correctly	prior_question_elapsed_time	prior_question_had_explanation
0	0	0	115	5692	0	1	3	1	<NA>	<NA>
1	1	56943	115	5716	0	2	2	1	37000.0	False
2	2	118363	115	128	0	0	0	1	55000.0	False
3	3	131167	115	7860	0	3	0	1	19000.0	False
4	4	137965	115	7922	0	4	1	1	11000.0	False

File Formats

It is common to convert a dataset into a format which is easier or faster to read or smaller in size to store. There are various formats in which datasets can be stored though not all will be readable across different packages. Let's look at how these datasets can be converted into different formats.

Most of them are available in pandas: https://pandas.pydata.org/pandas-docs/stable/user_guide/io.html

In [9]:
```
# reading data from csv using datatable and converting to pandas
# data = dt.fread("../input/riiid-test-answer-prediction/train.csv").to_pandas()

# writing dataset as csv
# data.to_csv("riiid_train.csv", index=False)

# writing dataset as hdf5
# data.to_hdf("riiid_train.h5", "riiid_train")

# writing dataset as feather
# data.to_feather("riiid_train.feather")

# writing dataset as parquet
# data.to_parquet("riiid_train.parquet")
```

```
# writing dataset as pickle
# data.to_pickle("riiid_train.pkl.gzip")
```

Format: csv

Most datasets are available in csv format and is pretty much the standard format in which datasets are shared. Almost all methods can be used to read data from csv.

Read more: https://en.wikipedia.org/wiki/Comma-separated_values

In [10]:
```
%%time

dtypes = {
    "row_id": "int64",
    "timestamp": "int64",
    "user_id": "int32",
    "content_id": "int16",
    "content_type_id": "boolean",
    "task_container_id": "int16",
    "user_answer": "int8",
    "answered_correctly": "int8",
    "prior_question_elapsed_time": "float32",
    "prior_question_had_explanation": "boolean"
}

data = pd.read_csv("../input/riiid-test-answer-prediction/train.csv", dtype=dtypes)

print("Train size:", data.shape)
```

```
Train size: (101230332, 10)
CPU times: user 3min 4s, sys: 4.25 s, total: 3min 8s
Wall time: 3min 6s
```

Format: feather

It is common to store data in feather (binary) format specifically for pandas. It significantly improves reading speed of datasets.

Read more: https://arrow.apache.org/docs/python/feather.html

In [11]:
```
%%time

data = pd.read_feather("../input/riiid-train-data-multiple-formats/riiid_train.feather")

print("Train size:", data.shape)
```

```
Train size: (101230332, 10)
CPU times: user 2.77 s, sys: 4.61 s, total: 7.39 s
Wall time: 22.8 s
```

Format: hdf5

HDF5 is a high-performance data management suite to store, manage and process large and complex data.

Read more: https://www.hdfgroup.org/solutions/hdf5

In [12]:
```
%%time

data = pd.read_hdf("../input/riiid-train-data-multiple-formats/riiid_train.h5", "riiid_train")

print("Train size:", data.shape)
```

```
Train size: (101230332, 10)
CPU times: user 9.97 s, sys: 9.2 s, total: 19.2 s
Wall time: 44.2 s
```

Format: parquet

In the Hadoop ecosystem, parquet was popularly used as the primary file format for tabular datasets and is now extensively used with Spark. It has become more available and efficient over the years and is also supported by pandas.

Read more: https://parquet.apache.org/documentation/latest/

In [18]:
```
%%time

data = pd.read_parquet("../input/riiid-train-data-multiple-formats/riiid_train.parquet")

print("Train size:", data.shape)
```

```
Train size: (101230332, 10)
CPU times: user 14.4 s, sys: 5.84 s, total: 20.2 s
Wall time: 7.15 s
```

Format: pickle

Python objects can be stored in the form of pickle files and pandas has inbuilt functions to read and write dataframes as pickle objects.

Read more: https://docs.python.org/3/library/pickle.html

In [19]:
```
%%time

data = pd.read_pickle("../input/riiid-train-data-multiple-formats/riiid_train.pkl.gzip")

print("Train size:", data.shape)
```

```
Train size: (101230332, 10)
CPU times: user 4.05 s, sys: 2.14 s, total: 6.19 s
Wall time: 6.14 s
```

Each method has it's own set of pros and cons. Some examples are:

- **Pandas** requires a lot more RAM to handle large datasets.
- **Dask** can be slow at times especially with transformations that cannot be parallelized.
- **Rapids** is not useful if you don't have a GPU.