In [1]:

```
import cudf
import pandas as pd
import numpy as np
import xgboost as xgb
from typing import Tuple

def make_categorical(n_samples: int, n_features: int, n_categories: int, onehot: bool) -> Tuple[pd.DataFrame, pd.Series]:
    """Make some random data for demo."""
    rng = np.random.RandomState(1994)

    pd_dict = {}
    for i in range(n_features + 1):
        c = rng.randint(low=0, high=n_categories, size=n_samples)
        pd_dict[str(i)] = pd.Series(c, dtype=np.int64)

    df = pd.DataFrame(pd_dict)
    label = df.iloc[:, 0]
    df = df.iloc[:, 1:]
    for i in range(0, n_features):
        label += df.iloc[:, i]
    label += 1

    df = df.astype("category")
    categories = np.arange(0, n_categories)
    for col in df.columns:
        df[col] = df[col].cat.set_categories(categories)

    if onehot:
        return pd.get_dummies(df), label
    return df, label


def main(X, y, gpu_training=False, cudf_dataframe=False, mix_datatypes=False) -> None:
    if cudf_dataframe:
        # Copy data from host to GPU memory
        X = cudf.DataFrame.from_pandas(X)
        y = cudf.Series.from_pandas(y)

    if mix_datatypes:
        # Convert first series from categoy to int64
        X[X.columns.values[0]] = X[X.columns.values[0]].astype('int64')
```

```
    print(list(X.dtypes))

    # Specify `enable_categorical` to True, also we use onehot encoding based split
    # here for demonstration. For details see the document of `max_cat_to_onehot`.
    tree_method="gpu_hist" if gpu_training else "hist"
    reg = xgb.XGBRegressor(tree_method=tree_method, enable_categorical=True, max_cat_to_onehot=5, n_estimators=2)
    reg.fit(X, y, eval_set=[(X, y)])

# Print libraries versions
print("Pandas version: " + pd.__version__)
print("NumPy version: " + np.__version__)
print("cuDF version: " + cudf.__version__)
print("XGBoost version: " + xgb.__version__)

# Use builtin categorical data support
# For scikit-learn interface, the input data must be pandas DataFrame or cudf
# DataFrame with categorical features
X, y = make_categorical(n_samples=100, n_features=2, n_categories=4, onehot=False)

# The following code works fine.
print("\n**************")
print("Test #1: GPU training, Pandas dataframe, categorical and integer data.")
main(X.copy(), y.copy(), gpu_training=True, cudf_dataframe=False, mix_datatypes=True)

# The following code fails.
print("\n**************")
print("Test #2: GPU training, cuDF dataframe, categorical and integer data.")
main(X.copy(), y.copy(), gpu_training=True, cudf_dataframe=True, mix_datatypes=True)
```

```
Pandas version: 1.4.3
NumPy version: 1.22.0
cuDF version: 22.08.00
XGBoost version: 1.6.1

**************
Test #1: GPU training, Pandas dataframe, categorical and integer data.
[dtype('int64'), CategoricalDtype(categories=[0, 1, 2, 3], ordered=False)]
[0]     validation_0-rmse:3.88012
[1]     validation_0-rmse:2.90380

**************
Test #2: GPU training, cuDF dataframe, categorical and integer data.
[dtype('int64'), <cudf.core.dtypes.CategoricalDtype object at 0x7fe52a231880>]
```

```
---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
/tmp/ipykernel_62513/1188575047.py in <module>
     68 print("\n***************")
     69 print("Test #2: GPU training, cuDF dataframe, categorical and integer data.")
---> 70 main(X.copy(), y.copy(), gpu_training=True, cudf_dataframe=True, mix_datatypes=True)

/tmp/ipykernel_62513/1188575047.py in main(X, y, gpu_training, cudf_dataframe, mix_datatypes)
     47     tree_method="gpu_hist" if gpu_training else "hist"
     48     reg = xgb.XGBRegressor(tree_method=tree_method, enable_categorical=True, max_cat_to_onehot=5, n_estimators=2)
---> 49     reg.fit(X, y, eval_set=[(X, y)])
     50
     51 # Print libraries versions

/opt/conda/envs/rapids/lib/python3.9/site-packages/xgboost/core.py in inner_f(*args, **kwargs)
    530         for k, arg in zip(sig.parameters, args):
    531             kwargs[k] = arg
--> 532         return f(**kwargs)
    533
    534     return inner_f

/opt/conda/envs/rapids/lib/python3.9/site-packages/xgboost/sklearn.py in fit(self, X, y, sample_weight, base_margin, eval_set, e
val_metric, early_stopping_rounds, verbose, xgb_model, sample_weight_eval_set, base_margin_eval_set, feature_weights, callbacks)
    929         """
    930         evals_result: TrainingCallback.EvalsLog = {}
--> 931         train_dmatrix, evals = _wrap_evaluation_matrices(
    932             missing=self.missing,
    933             X=X,

/opt/conda/envs/rapids/lib/python3.9/site-packages/xgboost/sklearn.py in _wrap_evaluation_matrices(missing, X, y, group, qid, sa
mple_weight, base_margin, feature_weights, eval_set, sample_weight_eval_set, base_margin_eval_set, eval_group, eval_qid, create_
dmatrix, enable_categorical)
    399
    400     """
--> 401     train_dmatrix = create_dmatrix(
    402         data=X,
    403         label=y,

/opt/conda/envs/rapids/lib/python3.9/site-packages/xgboost/sklearn.py in <lambda>(**kwargs)
    943             eval_group=None,
    944             eval_qid=None,
--> 945             create_dmatrix=lambda **kwargs: DMatrix(nthread=self.n_jobs, **kwargs),
    946             enable_categorical=self.enable_categorical,
```

```
    947         )

/opt/conda/envs/rapids/lib/python3.9/site-packages/xgboost/core.py in inner_f(*args, **kwargs)
    530         for k, arg in zip(sig.parameters, args):
    531             kwargs[k] = arg
--> 532         return f(**kwargs)
    533 
    534     return inner_f

/opt/conda/envs/rapids/lib/python3.9/site-packages/xgboost/core.py in __init__(self, data, label, weight, base_margin, missing,
 silent, feature_names, feature_types, nthread, group, qid, label_lower_bound, label_upper_bound, feature_weights, enable_catego
rical)
    641             return
    642 
--> 643         handle, feature_names, feature_types = dispatch_data_backend(
    644             data,
    645             missing=self.missing,

/opt/conda/envs/rapids/lib/python3.9/site-packages/xgboost/data.py in dispatch_data_backend(data, missing, threads, feature_name
s, feature_types, enable_categorical)
    894         )
    895     if _is_cudf_df(data) or _is_cudf_ser(data):
--> 896         return _from_cudf_df(
    897             data, missing, threads, feature_names, feature_types, enable_categorical
    898         )

/opt/conda/envs/rapids/lib/python3.9/site-packages/xgboost/data.py in _from_cudf_df(data, missing, nthread, feature_names, featu
re_types, enable_categorical)
    687         data, feature_names, feature_types, enable_categorical
    688     )
--> 689     interfaces_str = _cudf_array_interfaces(data, cat_codes)
    690     handle = ctypes.c_void_p()
    691     config = bytes(json.dumps({"missing": missing, "nthread": nthread}), "utf-8")

/opt/conda/envs/rapids/lib/python3.9/site-packages/xgboost/data.py in _cudf_array_interfaces(data, cat_codes)
    601         for i, col in enumerate(data):
    602             if is_categorical_dtype(data[col].dtype):
--> 603                 codes = cat_codes[i]
    604                 interface = codes.__cuda_array_interface__
    605             else:

IndexError: list index out of range
```