AI Saturdays Lagos Cohort 7 Practicals

Relational Data

Introduction to pandas
Optimized computational framework (CuDF)

Relational Data

The term “relation” can be interchanged with the standard notion we have of “tabular data,” say an instance of a “Person” relation

Rows are called tuples (or records), represent a single instance of this relation, and must be unique

Columns are called attributes, specify some element contained by each of the tuples

ID	Last Name	First Name	Role
1	Kolter	Zico	Instructor
2	Williams	Josh	TA
3	Dan	Chen	TA
4	Jain	Mayank	TA
5	Gates	William	Student
6	Musk	Elon	Student

ID	Last Name	First Name	Role
1	Kolter	Zico	Instructor
2	Williams	Josh	TA
3	Dan	Chen	TA
4	Jain	Mayank	TA
5	Gates	William	Student
6	Musk	Elon	Student

Primary and Foreign Keys

Primary key: unique ID for every tuple in a relation (i.e. every row in the table), each relation must have exactly one primary key

A foreign key is an attribute that points to the primary key of another relation If you delete a primary key, need to delete all foreign keys pointing to it

Person

ID	Last Name	First Name	Role ID
1	Kolter	Zico	1
2	Williams	Josh	2
3	Dan	Chen	2
4	Jain	Mayank	2
5	Gates	William	3
6	Musk	Elon	3

Person

ID	Last Name	First Name	Role ID
1	Kolter	Zico	1
2	Williams	Josh	2
3	Dan	Chen	2
4	Jain	Mayank	2
5	Gates	William	3
6	Musk	Elon	3

Entity relationship (multiple tables and relations)

1. One-to-one

Person

ID	Last Name	First Name	Role ID
1	Kolter	Zico	1
2	Williams	Josh	2

Andrew ID

Person ID	Andrew ID
1	zkolter
2	jnwillia

2. (One-to-zero/one)

Grades

Person ID	HW1 Grade	HW2 Grade
5	100	80
6	60	80

3. One-to-many (and many-to-one)

Person

ID	Last Name	First Name	Role ID
1	Kolter	Zico	1
2	Williams	Josh	2
3	Dan	Chen	2
4	Jain	Mayank	2
5	Gates	William	3
6	Musk	Elon	3

Role

ID	Name
1	Instructor
2	TA
3	Student

4. Many-to-many

Homework

ID	Name	388 Points	688 Points
1	HW 1	65	35
2	HW 2	75	25

Person Homework

Person ID	HW ID	Score
5	1	100
5	2	80
6	1	60
6	2	80

Indexes

Indexes are created as ways to “quickly” access elements of a table. Think of an index as a separate sorted table containing the indexed column and the tuple location

The primary key always has an index associated with it (so you can think of primary keys themselves as always being a fast way to access data) Indexes don’t have to be on a single column, can have an index over multiple columns (with some ordering)

Person

Location	ID	Last Name	First Name	Role ID
0	1	Kolter	Zico	1
100	2	Williams	Josh	2
200	3	Dan	Chen	2
300	4	Jain	Mayank	2
400	5	Gates	William	3
500	6	Musk	Elon	3

Last Name Index

Last Name	Location
Dan	200
Gates	400
Jain	300
Kolter	0
Musk	500
Williams	100

Pandas

Pandas is a “Data Frame” library in Python, meant for manipulating in-memory data with row and column labels (as opposed to, e.g., matrices, that have no row or column labels)

Pandas is a Python package for data analysis and exposes two new data structures: Dataframes and Series.

- Dataframes store tabular data consisting of rows and columns.
- Series are similar to Python's built-in list or set data types.

In this class, we will explore the structures that Pandas provides, and learn how to interact with them.

Creating A Dataframe and Basic Exploration

1. Importing Pandas

To import an external Python library such as Pandas, use Python's import function. To save yourself some typing later on, you can give the library you import an alias. Here, we are importing Pandas and giving it an alias of pd.

```
import pandas as pd
```

2. Loading a dataframe

We will load a CSV file as a dataframe using Pandas read_csv method. This will allow us to use Pandas' dataframe functions to explore the data in the CSV.

```
path = '../content/'
fileName = 'Default_Fin.csv'

df = pd.read_csv (path+fileName)
```

3. Basic Exploration with pandas functions

#This function allow you to see the shape of your dataset

```
df.shape
```

#To get some basic stats of the columns you can either use .describe() for discrete data or .value_counts for categorical data

```
df.describe()
df.value_counts()
```

```
#type checking (int, float, bool, string)
df.dtypes
```

4. Selecting Data

To examine a specific column of the DataFrame either at the beginning or end, we respectively see the following:

`df['Employed']` or `df['Employed'].head()`

#To examine specific rows and columns of a Dataframe, Pandas provides the iloc and loc methods to do so. iloc is used when you want to specify a list or range of indices, and .loc is used when you want to specify a list or range of labels.

```
# Get rows 1 through 3 and columns 0 through 5
df.iloc[1:3,:5]
# Get rows with index values of 2-4 and the columns basket_amount and activity
df.loc[2:4, ["Bank Balance", "Employed"]]
```

5. Checking for missing value

If we have missing data, is the missing data at random or not? If data is missing at random, the data distribution is still representative of the population. You can probably ignore the missing values as an inconvenience. However, if the data is systematically missing, the analysis you do may be biased. You should carefully consider the best way to clean the data, it may involve dropping some data.

```
df.isnull().sum().sum()
df.isnull().sum()
df.dropna()
df.fillna(method='ffill')
df.fillna(method='bfill')
```

The need for speed (optimization)

When you have a large amount of raw data and you may have to make useful information or generate features out of it. For this purpose, you may need to use one of the most popular python libraries for data analysis, Pandas.

Unfortunately, early on, Pandas had gotten a low reputation for being “slow”. However, the good news is that for most applications, well-written Pandas code is *fast enough*; and what Pandas lacks in speed, it makes up for in being powerful and user-friendly.

Optimized computational framework

The most essential framework which is widely used by data scientist is — **Pandas!** Pandas has made life easier with the help of its memory processing power and the use of dataframes.

Along with Pandas, there are other alternatives, one of which we are going to look at is a better python-based dataframe which can further boost up your computational power by optimizing the pandas workflows — **CuDF!**

CuDF

cuDF is **a Python-based GPU DataFrame library for working with data including loading, joining, aggregating, and filtering data and otherwise manipulating data**.

This makes it very easy for Data Scientists, Analysts, and Engineers to integrate it into their workflow.

Some basic exploration with CuDF in comparison with pandas

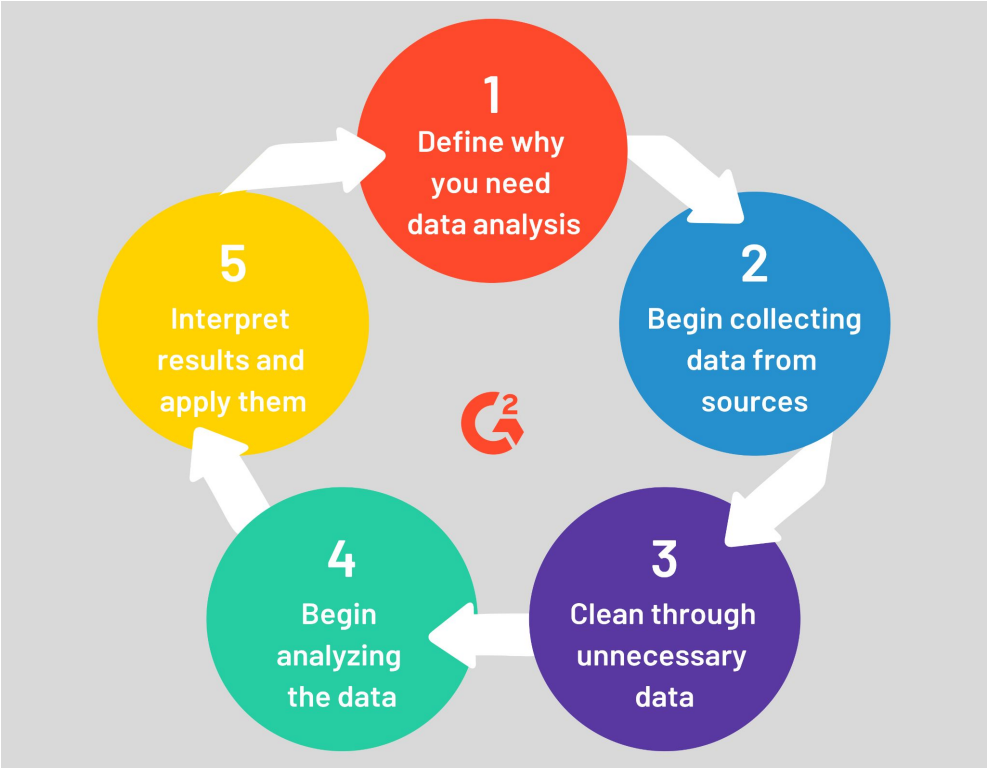
```
import cudf
import pandas as pd

#Loading the data
%time gdf = cudf.read_csv('./data/pop_1-03.csv')
gdf.shape

%time df = pd.read_csv('./data/pop_1-03.csv')
gdf.shape == df.shape
```

Some basic exploration with CuDF in comparison with pandas

```
#converting data types
%time gdf['age'] = gdf['age'].astype('float32')

%time df['age'] = df['age'].astype('float32')

#String operation
%time gdf['name'] = gdf['name'].str.title()

%time df['name'] = df['name'].str.title()
```

Add on:
Data analysis process

- Identify the problem to be solved
- Collection of data
- Data wrangling / Data cleaning
- Analyze data
- Interpret results



References

- https://github.com/rapidsai/cudf
- https://towardsdatascience.com/understanding-the-need-for-optimization-when-using-pandas-8ce23b83330c
- https://towardsdatascience.com/boost-up-pandas-dataframes-46944a93d33e
- https://docs.rapids.ai/api/cudf/stable/10min.html
- http://www.datasciencecourse.org/slides/relational_data.pdf
- https://www.g2.com/articles/data-analysis-process