CS 295B/CS 395B Systems for Knowledge Discovery

Lecture 4: SQL + CGMs

The University of Vermont

Topics

Data Frames vs. Databases

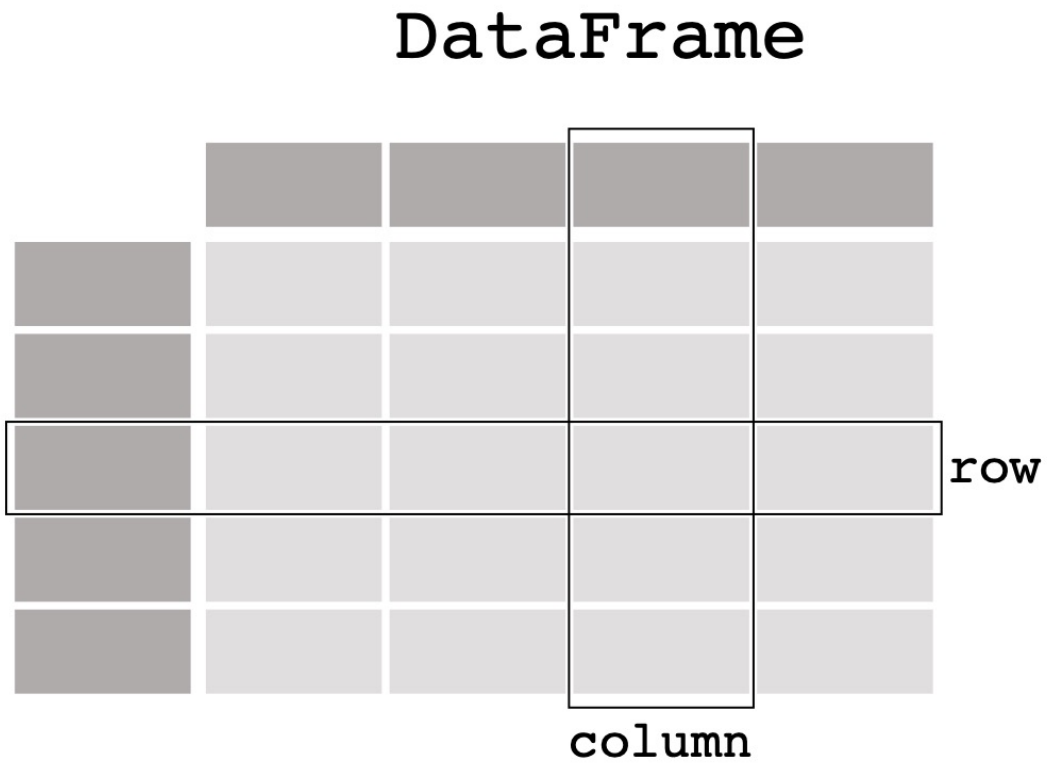
Structured Query Language (SQL)

Database Schemata

Database performance

Causal Reasoning

Dataframes: Pandas, R



jupyter tutorial Last Checkpoint: 3 minutes ago (autosaved)

File Edit View Insert Cell Kernel Widgets Help

PyCon 2018: Using pandas for Better (and Worse) Data Science

GitHub: https://github.com/justmarkham/pycon-2018-tutorial

In [1]:
```python
import matplotlib.pyplot as plt
import pandas as pd
pd.__version__
```
Out[1]: '0.24.1'

Dataset: Stanford Open Policing Project (video)

In [2]:
```python
# ri stands for Rhode Island
ri = pd.read_csv('police.csv')
```

In [3]:
```python
# what does each row represent?
ri.head()
```
Out[3]:

	stop_date	stop_time	county_name	driver_gender	driver_age_raw	driver_age	driver_race	violation_raw	violation	search_
0	2005-01-02	01:55	NaN	M	1985.0	20.0	White	Speeding	Speeding	
1	2005-01-18	08:15	NaN	M	1965.0	40.0	White	Speeding	Speeding	
2	2005-01-23	23:15	NaN	M	1972.0	33.0	White	Speeding	Speeding	
3	2005-02-20	17:15	NaN	M	1986.0	19.0	White	Call for Service	Other	

```sql
PRINT 'After ROLLBACK example'
DECLARE @FlagINT INT
SET @FlagInt = 1
    PRINT @FlagInt ---- @FlagInt Value will be 1
BEGIN TRANSACTION
SET @FlagInt = 2
    PRINT @FlagInt ---- @FlagInt Value will be 2
ROLLBACK TRANSACTION
    PRINT @FlagInt ---- @FlagInt Value will be ?
GO

PRINT 'After COMMIT example'
DECLARE @FlagINT INT
SET @FlagInt = 1
    PRINT @FlagInt  ---- @FlagInt Value will be 1
BEGIN TRANSACTION
SET @FlagInt = 2
    PRINT @FlagInt ---- @FlagInt Value will be 2
COMMIT TRANSACTION
    PRINT @FlagInt ---- @FlagInt Value will be ?
GO
```

Messages

```
After ROLLBACK example
1
2
2   Value Remains the Same
After COMMIT example
1
2
2   Value Remains the Same
```

Data School

SQL Authority

Exploratory Data Analysis (EDA)

(Tukey, Tufte, Wickham)

One-off Tutorials

Reproducibility issues

Data School

Knowledge Discovery

Robust large-scale processing

Heavy-weight

SQL Authority

Cells vs. Transactions

PyCon 2018: Using pandas for Better (and Worse) Data Science

GitHub: https://github.com/justmarkham/pycon-2018-tutorial

In [1]:
```python
import matplotlib.pyplot as plt
import pandas as pd
pd.__version__
```

Out[1]: '0.24.1'

Dataset: Stanford Open Policing Project (video)

In [2]:
```python
# ri stands for Rhode Island
ri = pd.read_csv('police.csv')
```

In [3]:
```python
# what does each row represent?
ri.head()
```

Out[3]:

	stop_date	stop_time	county_name	driver_gender	driver_age_raw	driver_age	driver_race	violation_raw	violation	search_
0	2005-01-02	01:55	NaN	M	1985.0	20.0	White	Speeding	Speeding	
1	2005-01-18	08:15	NaN	M	1965.0	40.0	White	Speeding	Speeding	
2	2005-01-23	23:15	NaN	M	1972.0	33.0	White	Speeding	Speeding	
3	2005-02-20	17:15	NaN	M	1986.0	19.0	White	Call for Service	Other	

```sql
PRINT 'After ROLLBACK example'
DECLARE @FlagINT INT
SET @FlagInt = 1
    PRINT @FlagInt ---- @FlagInt Value will be 1
BEGIN TRANSACTION
SET @FlagInt = 2
    PRINT @FlagInt ---- @FlagInt Value will be 2
ROLLBACK TRANSACTION
    PRINT @FlagInt ---- @FlagInt Value will be ?
GO

PRINT 'After COMMIT example'
DECLARE @FlagINT INT
SET @FlagInt = 1
    PRINT @FlagInt  ---- @FlagInt Value will be 1
BEGIN TRANSACTION
SET @FlagInt = 2
    PRINT @FlagInt ---- @FlagInt Value will be 2
COMMIT TRANSACTION
    PRINT @FlagInt ---- @FlagInt Value will be ?
GO
```

Messages

```
After ROLLBACK example
1
2
2   Value Remains the Same
After COMMIT example
1
2
2   Value Remains the Same
```

Data School

SQL Authority

Databases: Transactions (all for one and one for all)

begin transaction

insert into credit_charges **values** (…)

delete from inventory **where**(…)

insert into shipping_requests **values**(…)

end transaction

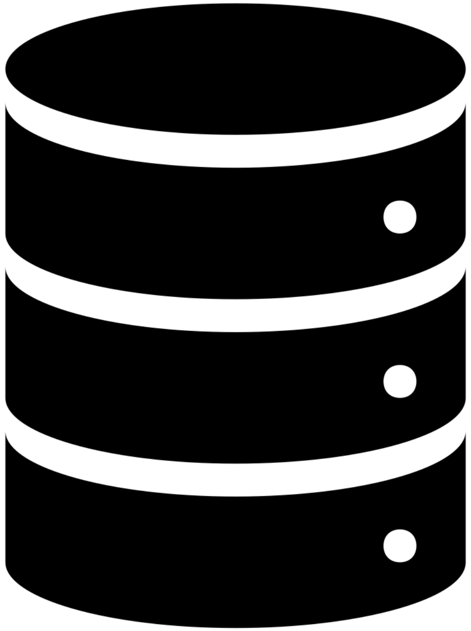
Queue up multiple statements that must be executed together; ***like***

(1) charging someone's credit card

(2) removing from the inventory

(3) Shipping that item to them

Databases: SQL



Database tables a lot like dataframes, but can be related, and updated safely.

Databases can be distributed, but on a single system, famously promise to maintain ACID properties.

Dropping some ACID Guarantees

Atomicity – transactions are applied as a single unit in time

Consistency – all rules of the database are maintained across statements

Isolation – if transactions are running concurrently, they don't get in each other's way.

Durability – database is kept whole even in the face of power or system failures

ACID deals with ***mutability***; data frames and KDD often deal with immutable copies of the mutating database.

Two halves of SQL

Data Definition Language (DDL)

```
create table courses if not exists (
  instructor int,
  prefix text,
  number int,
  title text, …
);
```

Data Manipulation Language (DML)

```
select instructor, title
  from courses
  where prefix = 'CS'
    and number = 295
```

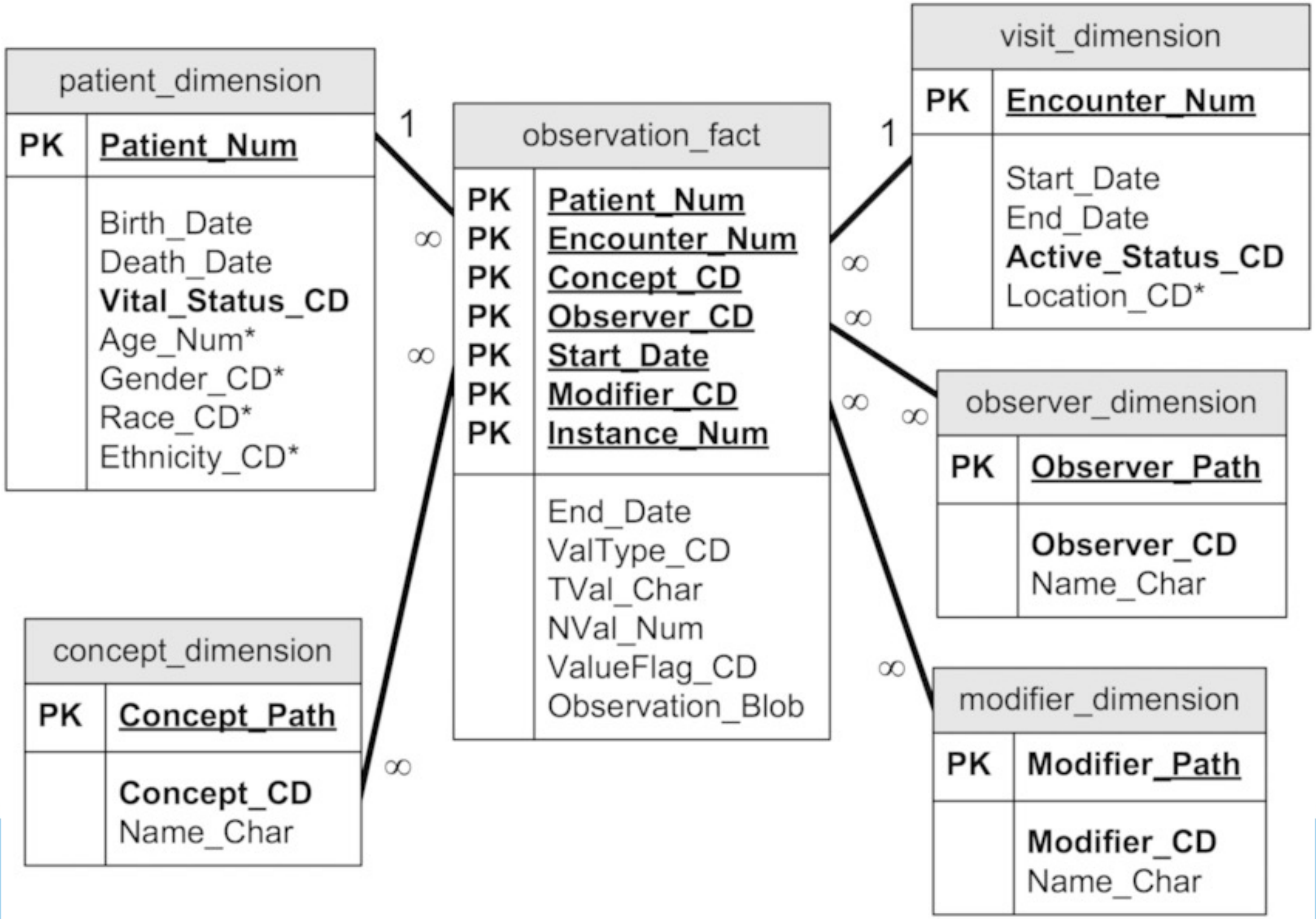
DML: the good parts

JOINS

- Extremely powerful (expressive)
- Extremely fast
- Many types, but WHERE is the easiest

Schema is important

Real World Schema: i2b2 database



Star-Schema: additional tables describe central table.

Many related tables working together.

https://www.w3schools.com/sql/sql_join.asp

Joining Tables

To figure out the students in a particular course, we may have to join the course_enroll table with the courses table.

```
select course_enroll.student_id
  from courses
  join course_enroll on courses.id = course_enroll.course
  where courses.number = 295 and courses.prefix = 'CS'
```

To then get the students' emails, we would need to involve the student table.

Data warehousing / normalizing databases

Sometimes for KDD style applications we ***simplify*** the database, since it's not being edited.

Databases sometimes adhere to normal forms; there are different tradeoffs for:

- making queries fast e.g., getting everything relevant into 1 table

- making it easy to add new data

https://en.wikipedia.org/wiki/Database_normalization

Embedded DSLs vs. standalone DSLs

- Pandas is arguably an embedded DSL
  - You ***can*** write whatever Python you want, but if you limit yourself to pandas operations they are much faster.
- SQL is a standalone DSL
  - You can write SQL in separate files, and database APIs execute strings of SQL code and provide results back in various forms
  - It is not uncommon to see a ***schema.sql*** file containing all the DDL for a database in a project.

SQL outside of DBs (I): Spark / Arrow / Drill

Apache Spark (paper for Wednesday!) offers SQL execution across large datasets that are maintained in-memory on a cluster

Formats in the Apache Arrow project are designed with the goal of SQL executing over these files even if not fully loaded into memory.

https://drill.apache.org/docs/querying-parquet-files/

SQL outside of DBs (II): C# LINQ

Embedded SQL DSL that supports all types of collections/arrays/etc.

https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/concepts/linq/

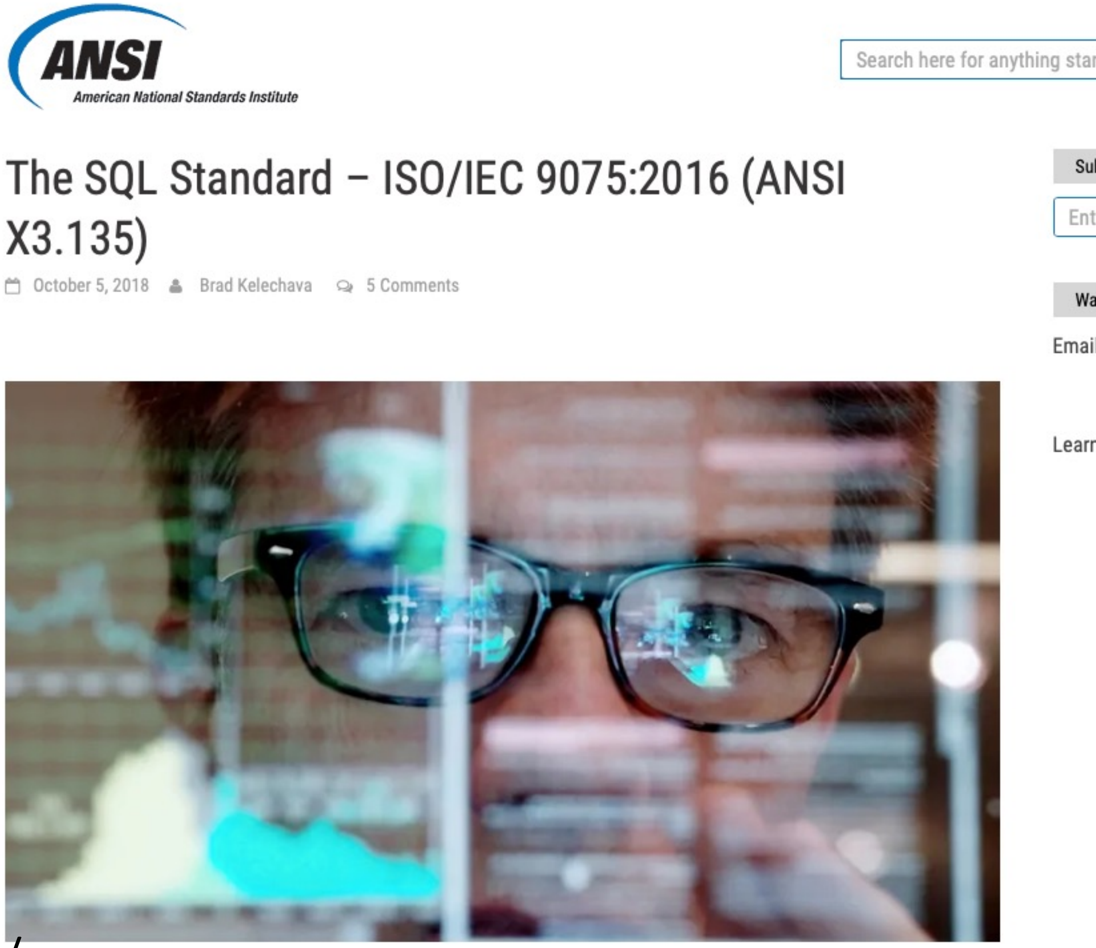
```
// Specify the data source.
int[] scores = new int[] { 97, 92, 81, 60 };

// Define the query expression.
IEnumerable<int> scoreQuery =
    from score in scores
    where score > 80
    select score;

// Execute the query.
foreach (int i in scoreQuery)
{
    Console.Write(i + " ");
}
```

Standard SQL vs. SQL in practice

- SQL standardization history quite interesting
- Different vendors have slightly different flavors
  - SQLite lacked an ***upsert*** operation (before 2018)
  - NULL values handled differently in some databases
    - MySQL uses IF/IFNULL, Posgres uses CASE
  - https://troels.arvin.dk/db/rdbms/



https://blog.ansi.org/2018/10/sql-standard-iso-iec-9075-2016-ansi-x3-135/

SQL as a domain-specific language (DSL)

Syntax (what it looks like)

```
ALTER INDEX { index_name | ALL }
    ON <object>
    { REBUILD
        [ [PARTITION = ALL]
                    [ WITH ( <rebuild_index_option> [ ,...n ] ) ]
          | [ PARTITION = partition_number
                [ WITH ( <single_partition_rebuild_index_option>
                        [ ,...n ] )
                ]
            ]
        ]
    | DISABLE
    | REORGANIZE
        [ PARTITION = partition_number ]
        [ WITH ( LOB_COMPACTION = { ON | OFF } ) ]
  | SET ( <set_index_option> [ ,...n ] )
    }
[ ; ]

<object> ::=
{
    [ database_name. [ schema_name ] . | schema_name. ]
        table_or_view_name
}
```

Semantics (how we assign meaning)

SQL:	SELECT name, phone FROM Employees WHERE phone = 1122;
Relational Algebra:	$\pi_{(name,phone)} \sigma_{(phone=1122)}$ (Employees)
DataLog:	q(Name, Phone) :- Employees(Name, Phone) $\wedge$ (Phone=1122)

Causal Reasoning in CS: Background

- Two main traditions we care about:
    - PO: Potential outcomes (Rubin causal model)
    - CGM: Causal graphical models (Pearl causal model)

Potential Outcomes Example: A/B test

$$T = \{A, B\}$$

$$\text{Avg}(Y \mid T{=}A) - \text{Avg}(Y \mid T{=}B)$$

Procedure:

1. *Randomly* split "units" into 2 groups
2. Assign one group A, one group B
3. Take some measurements of Y, let time pass
4. Compute average Y for units receiving A, average Y for units receiving B

Remind me to DO STUFF ON THE BOARD

Many (potential – pun intended) sources of complexity

Highlight three big challenges

1. Treatments may be complex
2. Interference (i.e., breaking the “stable unit treatment value assumption or SUTVA)
3. Observational data

Remind me to DO STUFF ON THE BOARD

Causal Graphical Models Background

Assume your data lives in a single database table

- Graph entails probability simplex
- Any joint distribution can be refactored via basic operations (e.g., def. of conditional probability/chain rule/multiplication rule (these are all the same thing))
    - These factorizations all require the same amount of space
- Use independences entailed by the graph to use less space, make it tractable

Remind me to DO STUFF ON THE BOARD

Causal Graphical Models Background

- If you know the structure
    - Great! Simulate!
- If you don't know the structure
    - Need to learn independencies
        - This is where most of the research is

Some of this framing I learned from Cosma Shalizi's text, but that was updated this year and I ran out of time looking for the older version.

Why are CGMs useful?

Used for *simulating* experiments

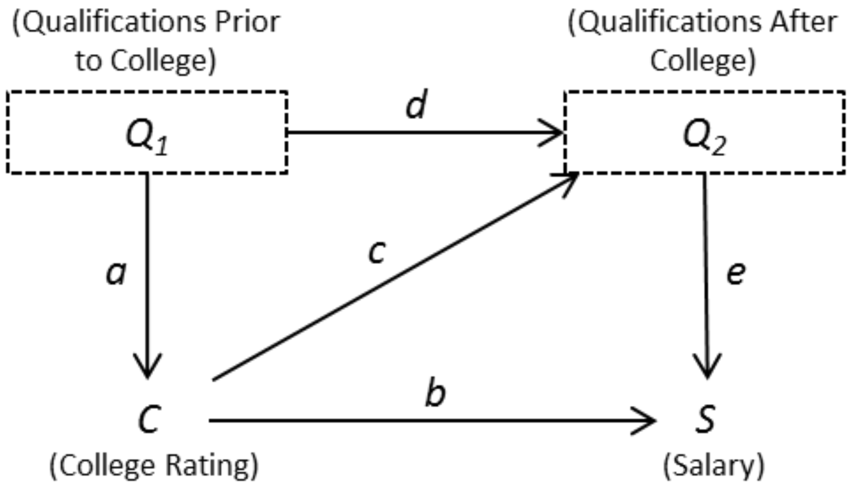
- Easy if we *have* the structure (just learn the parameters)
- Harder if we need to learn the structure
  - An active area of research

How to simulate?

- do-operator

CGMs as a domain-specific language (DSL)

Syntax (what it looks like)



Semantics (how we assign meaning)

P(Q1, Q2, C, S) = P(Q1)P(C|Q1)P(Q2|C, Q1)P(S|C, Q2)

+ parameters
+ any functional form assumptions
+ do-calculus

Why do we care?

For higher level modeling to be useful, we need to consider practical implications.

Sometimes this means *performance*.

Sometimes this means *measurable* or *verifiable* assumptions.

Sometimes this means verifying the *process*.

Themes

- Empirical study of performance
- Formal methods (e.g., logic, formal syntax and semantics) help ground assumptions in reality, not introduce bias
- Design for performance, discover generalizable findings