10718/94889
Tech Session 2: git & SQL

Agenda

1. Prerequisites
2. git/github: the basics
3. SQL: the basics
4. Troubleshooting in breakout rooms

Prerequisites

Git:

- Access to course github (can view https://github.com/dssg/test-mlpolicylab-private)
- Git command line interface set up on course server: When logged into the server, can run:

```
git clone git@github.com:dssg/test-mlpolicylab-private
```

SQL:

- Access to course database (can access `group_students_database` through psql or DBeaver)

Git & GitHub: a brief primer

What are Git/GitHub?

Git is Version Control Software - it tracks changes to files and folders

- Saves you from file version hell
- Review how a project has changed (how did this code work before?)
- Collaboration: Easy to combine new work from multiple teammates

GitHub hosts Git

- Organize teams
- Share a repository over the internet
- Other extra, non-git features

"FINAL".doc

FINAL.doc!

FINAL_rev.2.doc

FINAL_rev.6.COMMENTS.doc

FINAL_rev.8.comments5.
CORRECTIONS.doc

FINAL_rev.18.comments7.
corrections9.MORE.30.doc

FINAL_rev.22.comments49.
corrections.10.#@$%WHYDID
ICOMETOGRADSCHOOL????.doc

Git basics: Repositories

- Repository: A collection of files and folders making up a project in Git. Includes:
  - Your code & files
  - Other files used by Git to keep track of things
- Each team will use their repository to manage (and submit) their code this semester.
- Example: github.com/dssg/test-mlpolicylab-private is a repository.

Git basics: Getting a repository

Two ways to get a repository:

1. Create a new one with `git init`:
   - Create a folder to store your new repository
   - `cd` to it
   - Run `git init` inside
2. Download a pre-existing repository with `git clone`:
   - `cd` to a folder for storing projects, like `aaron/projects`
   - Run `git clone git@github.com:{username}/{project name}`
   - ex: `git clone git@github.com:dssg/test-mlpolicylab-private`
   - We'll do this one!

dssg / mlforpublicpolicylab

Code | Issues | Pull requests | Actions | Projects | Wiki | Security | Insights | Settings

master | 1 branch | 1 tag | Go to file | Add file | Code

rayidghani Update README.md

01 - Intro and Scoping	Update README.md	
02 - Case Studies and Acquiring Data	data discussion slide order	
03 - Data Exploration, Analytical For...	updating weekly readmes	
04 - Machine Learning Pipelines	updating weekly readmes	
05 - Features	updating weekly readmes	12 days ago
06 - Performance and Evaluation Pt 1	updating weekly readmes	12 days ago

Clone with SSH | Use HTTPS

Use a password protected SSH key.

git@github.com:dssg/mlforpublicpoli

Open with GitHub Desktop

Download ZIP

Git basics: Commits

Commit - A record of the state of a repository at one point in time

- A repository keeps a list of all of your commits
- Look at old commits to review change history
- Compare two commits and see what's changed

Example: Commits to our example repo

Git basics: Making commits

1. Stage files:
   - Staging collects the files you intend to update in the repository
   - In your repository, run: `git add <some_file>` for each file to be updated
   - Example: `git add test.py`
2. Once you've staged the desired files, commit your changes:
   - Committing your changes adds them to your repository
   - In your repository, run: `git commit -m "<some commit message>"`
   - Commit messages are short messages that summarize a commit.

Ex: `git commit -m "Update docstrings in feature engineering"`

(if interested: [commit message guidelines])

Git basics: sharing your changes

`git pull <remote> <branch>`: Update your local repository (and your code) to match a remote repository.

- We `pull` our changes from GitHub
- Might cause a `merge conflict` - read about that here
- Do this before you push!
- Example: `git pull origin master`

`git push <remote> <branch>`: Add your changes to a repository on a remote machine

- We use GitHub, so `push` sends our changes to GitHub's servers
- Your new commits are added to the remote repository
- Example: `git push origin master`

Git: Common workflow

When you start working:

The first time, clone an existing repo: `git clone`

Every time, get changes since last time: `git pull`

Add new files: `git add` or make changes to existing files

Make a local checkpoint: `git commit`

Push to the remote repository: first git pull, resolve any conflicts, then `git push`

More advanced cheatsheet is https://gist.github.com/jedmao/5053440

Git & GitHub: Exercises

Zoom Stuff:

- Complete these steps with your breakout group
- If your group gets stuck: click the “Ask for Help” button and we’ll come help

Getting the test repository

1. Use SSH to connect the class server:

   ```
   ssh {andrew_id}@mlpolicylab.dssg.io -i /path/to/id_rsa
   ```

2. Use `git clone` to get our test repository:

   ```
   git clone https://github.com/dssg/test-mlpolicylab-private.git
   ```

3. Change to the directory containing our repository:

   ```
   cd test-mlpolicylab-private
   ```

Making a commit

1. Create a new file, with some text:

   ```
   echo “{some message for your classmates}” > {your andrew_id}_f20.txt
   ```

2. Check to see that git recognizes your new file:

   ```
   git status
   ```

   Should look something like:

   ```
   adunmore@ip-10-0-1-213:~/test-mlpolicylab-private$ git status
   On branch master
   Your branch is up to date with 'origin/master'.

   Untracked files:
     (use "git add <file>..." to include in what will be committed)

           adunmore_f20.txt

   nothing added to commit but untracked files present (use "git add" to track)
   ```

Making a commit (continued)

3. Use `git add` to stage the file for committing:

   `git add {andrew_id}_f20.txt`

4. Check `git status` to see what's happened:

```
adunmore@ip-10-0-1-213:~/test-mlpolicylab-private$ git add adunmore_f20.txt
adunmore@ip-10-0-1-213:~/test-mlpolicylab-private$ git status
On branch master
Your branch is up to date with 'origin/master'.

Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

        new file:   adunmore_f20.txt
```

5. Use `git commit` to commit your change:

   `git commit -m "{a helpful commit message}"`

Pulling & Pushing

1. `pull` the class repository to make sure you're up to date:

   ```
   git pull origin master
   ```

2. `push` your change to the class repository to share it with your classmates:

   ```
   git push origin master
   ```

3. Check the class github to see your changes online!

PostgreSQL

- Only `select` based Data Manipulation Language (DML) covered to get you started with Level-1 data wrangling.
- There is more to DML along with Data Definition Language (DDL) which covers Drop, Alter, Create (which is not critical for now)

What is SQL and how do we use it?

SQL is a programming language to retrieve and analyze data that is in a database

We use SQL to ask the database what we want it to output and it produces that output. For example:

select firstname, lastname from voters will give us the two fields we specified (and all the rows) from the table called ‘voters’

Most of the queries we do will be a variation on this. For example, to limit rows to just PA:

select firstname, lastname from voters where state = ‘PA’

Deeper Dive: Select Anatomy

The SELECT statement has a lot more clauses:

1. Instead of getting all rows, you can select distinct rows using DISTINCT operator.
2. You can sort rows using ORDER BY clause.
3. Filter rows using WHERE clause.
4. Select a subset of rows from a table using LIMIT or FETCH clause so you don't get back 100 million rows
5. Aggregate data (rows) into groups using GROUP BY clause.
6. Filter groups using HAVING clause.
7. Join with other tables using joins such as INNER JOIN, LEFT JOIN, FULL OUTER JOIN, CROSS JOIN clauses.
8. Perform set operations using UNION, INTERSECT, and EXCEPT.

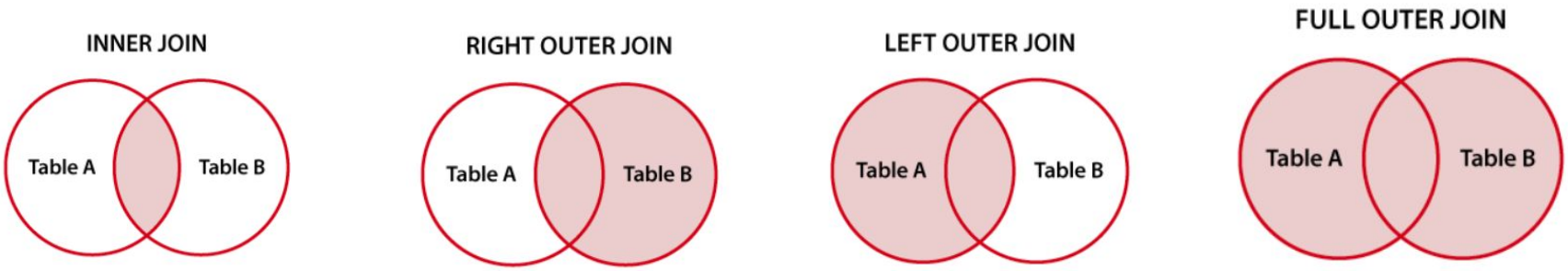
Aggregations: Group By

- The GROUP BY statement is often used with aggregate functions (COUNT, MAX, MIN, SUM, AVG) to group the result-set by one or more columns.

```
SELECT column_name(s)
FROM table_name
WHERE condition
GROUP BY column_name(s)
ORDER BY column_name(s)
```

- Aggregated groups can thave conditions applied using the HAVING clause

Joins



- A JOIN clause is used to combine rows from two or more tables, based on a related column between them.

General join Syntax:

```sql
SELECT *
FROM TableA
LEFT OUTER JOIN TableB
  ON tableA.name = tableB.name;
```

Left Join Example

TableA

a_id	name
1	apple
2	orange
3	tomato
4	cucumber

TableB

b_id	name
A	apple
null	*null*
null	*null*
B	banana
C	cucumber
D	dill

The resulting table will be as follows:

a_id	TableA.name	b_id	TableB.name
1	apple	A	apple
2	orange	*null*	*null*
3	tomato	*null*	*null*
4	cucumber	C	cucumber

Disclaimer: TableB does not have two null values. It’s only to highlight how it joins when there are no corresponding values for TableA in TableB.

NULLS and empty strings in PostgreSQL

```
CREATE TABLE test (
        id numeric(3,0) PRIMARY KEY,
        content varchar(255)
)

INSERT INTO test (id, content) VALUES (1, NULL);
INSERT INTO test (id, content) VALUES (2, ‘’);
INSERT INTO test(id, content) VALUES (3, ‘ ‘);
INSERT INTO test(id, content) VALUES (4, ‘x’);
```

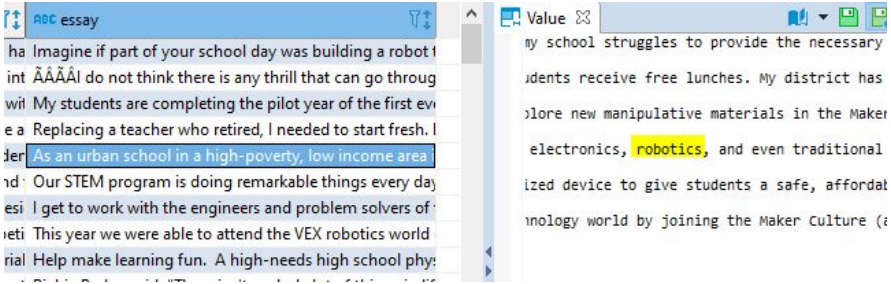
id	content	isnull	isempty	blank
1	(null)	1	0	0
2		0	1	0
3		0	0	1
4	x	0	0	0

SQL: Exercises

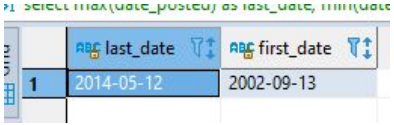
Dataset

- We will use a Kaggle dataset to play around and perform some basic analysis for a warm up with SQL commands.
- The dataset details are available here.
- donorschoose.org, a non-profit organisation provides a crowdfunding platform for public school teachers to get funds for their class projects directly from individuals.
- There are four tables under the donorschoose schema:
  1) projects - Contains information about each project.
  2) donations - Contains information about the donations to each project.
  3) essays - Contains project text posted by the teachers.
  4) resources - Contains information about the resources requested for each project.

Q1 -- Select projects that are related to `robotics` from the ‘essays’ table based on the short description provided by teachers.



Q2 -- Find the first and last project post dates available in the dataset.



	last_date	first_date
1	2014-05-12	2002-09-13

Q3 -- Find all categories of resource-types that can enable a project.

	resource_type
1	Other
2	Trips
3	Visitors
4	Supplies
5	Books
6	Technology

Q4 -- Find the total number of project submissions for the month of April in the year 2014.

	count
1	8,065

Q5 --What is the cumulative donation from the state of Pennsylvania ?

	donor_state	sum
1	PA	3,145,306.880000006

Q6 -- Show state-wise cumulative donation in descending order.

	donor_state	total
9	FL	5,904,958
10	NC	5,672,861
11	OK	4,787,662
12	IN	3,738,491
13	MA	3,475,936
14	VA	3,335,513
15	PA	3,145,306
16	MD	3,093,103
17	GA	3,092,347
18	CT	2,416,763
19	AZ	2,335,497
20	DC	2,250,185
21	MI	2,026,364
22	TN	1,950,049
23	CO	1,931,521
24	SC	1,911,372
25	OH	1,543,416
26	MO	1,509,681
27	NV	1,442,208

Q7 -- Find the distribution percentage of different payment methods for donations from Pennsylvania.

	donor_state	payment_method	transactions	total_transactions	pct
1	PA	paypal	6,910	52,451	13.1742006825
2	PA	no_cash_received	18,875	52,451	35.9859678557
3	PA	double_your_impact_match	323	52,451	0.6158128539
4	PA	creditcard	23,549	52,451	44.8971420945
5	PA	check	500	52,451	0.9532706717
6	PA	amazon	2,294	52,451	4.3736058416

Expected output snippet

Q8 -- For all the projects submitted in the dataset find the total financial aid obtained.

	title	overall_donation
17	Tuned in!	423.34
18	Reading and Writing in Color	363.6
19	You Ought to Be in Pictures:" Film as Personal Memoir	350.59
20	Re-String Please!	277.38
21	Ready - Set - Respond!	150
22	Give My First Graders the Gift of Reading!	559.46
23	Play Time! Encourage Reading Skills through Literacy Cer	35
24	HP Powerful Printing!	332.91
25	Concentrate, Collaborate and Learn	[NULL]
26	Butterfly Preserve	937.4
27	Living Pictures: Comprehension through Drama	10
28	Mastering More Than Facts!	1,090.8
29	Independent Reading Center	45
30	Literature Sleuths	5
31	Second Language Advantage	[NULL]
32	Getting Fit and Having Fun With Pedometers in P.E.	[NULL]

Expected output snippet

SOLUTIONS

A1

```
select title, essay from donorschoose.essays
where essay like '%robot%'
```

A2

```
select
    max(date_posted) as last_date, min(date_posted) as first_date
from
    donorschoose.projects
```

A3

```
select
        distinct(resource_type)
from
        donorschoose.projects
where
        resource_type is not null
```

A4

```
        SELECT
          count(*)
        FROM
          donorschoose.projects
        WHERE
                date_posted >= '2014-04-01'
              AND date_posted <=  '2014-04-30'
```

A5

```
select
        donor_state, sum(donation_total)
from
        donorschoose.donations
group by
        donor_state
having
        donor_state = 'PA
```

A6

```
 select
        donor_state, trunc(sum(donation_total)) as total
from
        donorschoose.donations
group by
        donor_state
order by
        total desc
```

A7

```
select
        dd.donor_state, dd.payment_method, dd.transactions, sub.total_transactions, (100.0 *
        dd.transactions) / sub.total_transactions as Pct
from (
        select donor_state, payment_method, count(payment_method) as transactions
        from donorschoose.donations
        group by donor_state, payment_method
        having donor_state = 'PA'
) dd
join (
        select donor_state, count(*) as total_transactions
        from donorschoose.donations
        group by donor_state
        having donor_state = 'PA'
) sub
on
        dd.donor_state = sub.donor_state
```

A8

```
SELECT
        title, overall_donation
FROM
    (   SELECT
            e.projectid as projid, title
        FROM
            donorschoose.essays e
        JOIN
            donorschoose.projects p on e.projectid = p.projectid
    ) t
LEFT JOIN
    (   select projectid, sum(donation_total) as overall_donation
        from
            donorschoose.donations
        group by projectid
    ) d
ON
    t.projid = d.projectid
```