QSS20: Modern Statistical Computing

Unit 06: Reshaping and merging (exact)

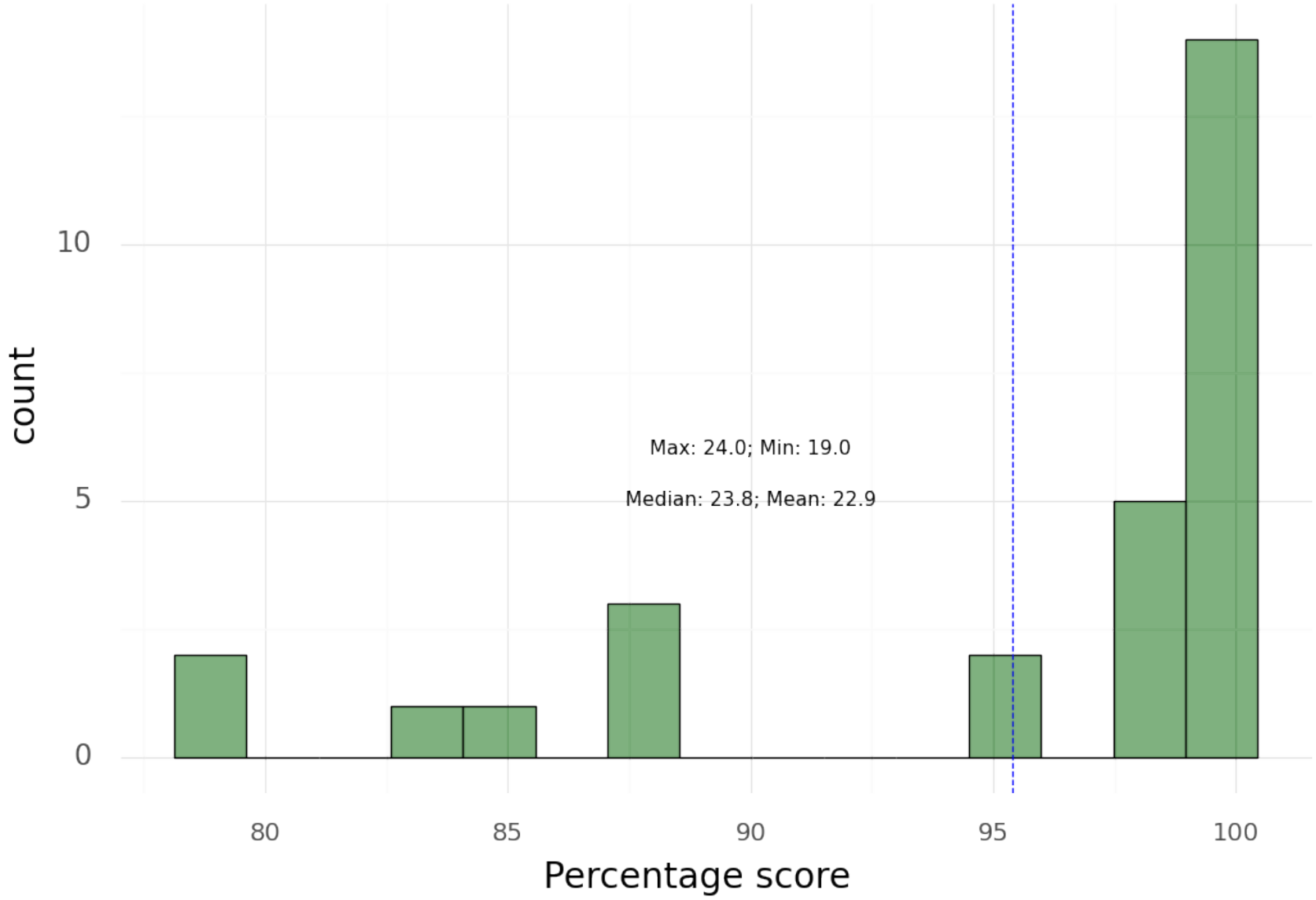
Goals for today

- Pset 1 grades and makeup
- Project shopping
- Recap of command line & git
- Reshaping data
- Merging data

Goals for today

- **Pset 1 grades and makeup**
- Project shopping
- Recap of command line & git
- Reshaping data
- Merging data

Pset 1 grades



Code for previous plot

```
plot_grades = (ggplot(grades_df, aes(x = 'pset1_prop')) +
geom_histogram(alpha = 0.5,
               color = "black",
               bins = 15,
               fill = 'darkgreen') +
theme_minimal(base_size = 20) +
xlab("Percentage score") +
geom_vline(xintercept = grades_df['pset1_prop'].mean(),
           linetype = "dashed",
           color = "blue") +
annotate("text", x = 90, y = 5,
         label = "Median: 23.8; Mean: 22.9") +
annotate("text", x = 90, y = 6,
         label = "Max: 24.0; Min: 19.0") +
theme(axis_text_x = element_text(size = 14)
      )
        )

plot_grades # show plot
plot_grades.save("pset1_grades_w23.png", width = 12,
                 height = 8, verbose = False) # save plot
```

Makeup for pset 1 problem 2 (list comprehension)

- Opportunity for folks who didn't do list comprehensions on problem 2 of pset 1 to demonstrate learning (and avoid point penalties)
- Another three problems to take the place of your score for problem 2
- Problems will be a little harder and likely also involve pandas, since working with DataFrames is a large reason we use list comprehensions
- Will be released to course repo Thursday (tomorrow)–not Canvas–and due this Sunday by 11:59 PM

Where we're headed

DataCamp deadlines:

- **Monday 01/30:** Chapter on regular expressions for matching
- **Wednesday 02/01:** Chapter on intro to HTML (essential for web-scraping)

TITLE	ASSIGNEES	STATUS	DUE BY	C	A	CR
Joining Data with pandas Data Merging Basics Chapter	Organization	DUE SOON	Jan 25, 15:30 EST	6	30	20%
Joining Data with pandas Merging Tables With Different Join Types Chapter	Organization	DUE SOON	Jan 25, 15:30 EST	5	30	16%
Regular Expressions in Python Regular Expressions for Pattern Matching Chapter	Organization	Active	Jan 30, 15:30 EST	2	30	6%
Web Scraping in Python Introduction to HTML Chapter	Organization	Active	Feb 1, 15:30 EST	0	30	0%

Class and problem sets:

- **Next class** (building on today): Basic regular expressions (regex) to clean join fields for merging datasets
- **Next Weds:** basic web-scraping for data collection and cleaning
- **Grades for pset 2:** hopefully this coming Monday
- **Next deadline in 11 days:** pset 3 due Sunday 02/05 at 11:59 PM

Where we are

- Pset 1 grades and makeup
- **Project shopping**
- Recap of command line & git
- Reshaping data
- Merging data

"Project shopping"

Goals: Connect with classmates around project ideas; start brainstorming!

Instructions: Group by option you're most interested in, discuss approaches/questions you're curious about, draw on whiteboard if you want, and **feel free to float around!**

Viewed from the door:

- **Left** side of room: SIP large dataset (SIRS)
- **Right** side of room: SIP medical training
- **Back** of room: Cook County sentencing dataset
- **Front** of room: Independent project

Where we are

- Pset 1 grades and makeup
- Project shopping
- **Recap of command line & git**
- Reshaping data
- Merging data

Recap of command line & git

What do you remember?

Recap of command line & git

Tips:

- Get used to terminal AND git! Baseline for most real-world coding
- If "no file exists", double-check where you are and where the file is
- Install things from command line, NOT inside a notebook

Useful commands:

```
pwd | cd | ls | cp | mv | rm # basic shell commands
touch | mkdir | rm -r folder | rm *.png # also useful
../ | ./ # up one level | current level

git clone | add | commit | push # basic git commands
git rm | pull | status # also useful
```

Where we are

- Pset 1 grades and makeup
- Project shopping
- Recap of command line & git
- **Reshaping data**
- Merging data

Two general formats for data

1. **Wide format data**
   - each row is one unit, e.g., a person, a company, a state
   - columns contain time or type-varying information about that unit
2. **Long format data**
   - each row is a snapshot of that unit, e.g., a person at one point in time, a state with one economic summary measure
   - each unit often has multiple rows

Example contrast

Wide format: one row per student

Student	gpa_2020	gpa_2021	ncourses_2020	ncourses_2021
1	3.8	3.9	5	3
2	3.6	3.6	6	7

Long format: each student repeated across years and type of statistic

Student	year	stat	value
1	2020	gpa	3.8
1	2021	gpa	3.9
1	2020	ncourses	5
1	2021	ncourses	3
2	2020	gpa	3.6
2	2021	gpa	3.4
⋮			

"Pivoting" from long to wide

```
pd.pivot(longformat_df,
    index = 'Student',
    columns = 'year',
    values = ['gpa', 'ncourses_2020', ...])
```

Breaking it down:

- **index**: name of column we want to treat as a row—in the previous example, we want one student per row
- **values:** name of column(s) that contain the values of data we want to "spread out"—in previous example, this is gpa and total courses
- **columns:** name of column(s) that describe the unit of variation—in previous example, the year column (2020 or 2021) and stat column (gpa or ncourses)

"Melting" from wide to long

```
pd.melt(wideformat_df,
    id_vars = 'Student')
```

Breaking it down:

- **id_vars**: name of column we want to treat as the unit of analysis that has repeated measures—in the previous example, each student can have multiple rows
- See documentation for optional arguments that help us rename the output

Pause for practice

Reshaping part (section 1) of `03_reshaping_merging.ipynb`

Where we are

- Pset 1 grades and makeup
- Project shopping
- Recap of command line & git
- Reshaping data
- **Merging data**

Working example

- **Main or "left" dataset:** data on Dartmouth students' high schools

Student	**Year**	**District**	**NCES ID**
Jeremy	2021	New Trier High School	1728200
Emma	2022	Hanover High	3302670
Esmeralda	2022	Homeschool	NA
⋮			

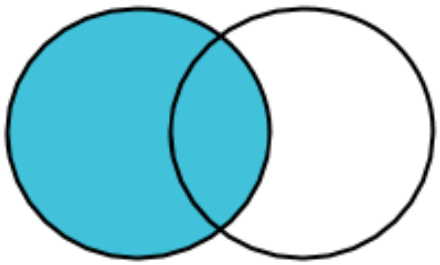
- **Auxiliary or "right" dataset:** percentage of students eligible for **FRPL** (free or reduced price lunch), a measure of school district poverty

District	**NCES ID**	**% FRPL**
New Trier HS	1728200	X%
Hanover HS	3302670	Y%
Lebanon HS	4107380	Z%
⋮		

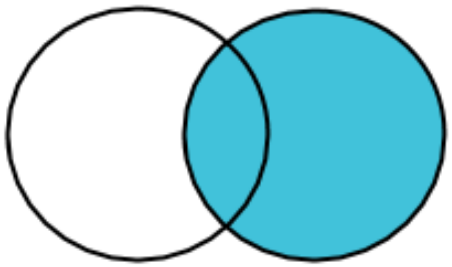
Possible join keys

- **Unique identifier:** used for "exact matching"— or a Yes/No match on that basis
  - E.g., is the NCES ID of New Trier found in the dataset of demographics?
- **Other identifiers:** can be used for either "exact match" or for "probabilistic/fuzzy matching"
  - **Probabilistic matching** (more advanced topic): what's the likelihood that "New Trier district" and "New Trier HS" are the same entity?

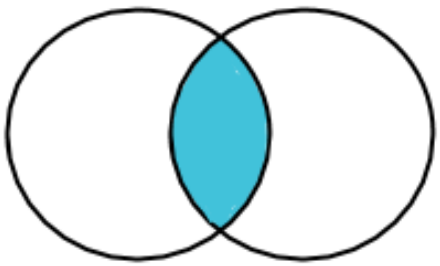
Conceptual overview of four types of joins



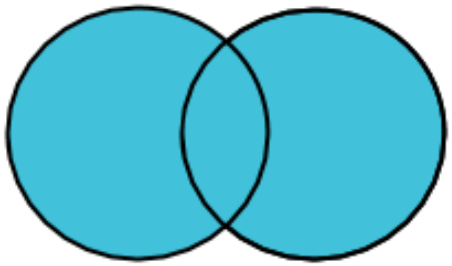
Left Join



Right Join



Inner Join



Full Outer Join

Source: Trifacta

Inner join in this context

In words: "drop all students whose districts don't appear in the demographics data; drop all districts that don't appear in the Dartmouth student data"

- **Main or "left" dataset**

Student	Year	District	NCES ID
Jeremy	2021	New Trier High School	1728200
Emma	2022	Hanover High	3302670
Esmeralda	2022	Homeschool	NA
⋮			

- **Auxiliary or "right" dataset**

District	NCES ID	% FRPL
New Trier HS	1728200	X%
Hanover HS	3302670	Y%
Lebanon HS	4107380	Z%
⋮		

Outer join in this context

In words: "keep all students from the student-level data; keep all schools from the school-level data—even if there's not an overlap"

Student	**Year**	**District**	**NCES ID**	**% FRPL**
Jeremy	2021	New Trier High School	1728200	X%
Emma	2022	Hanover High	3302670	Y%
Esmeralda	2022	Homeschool	NA	NA
NA	NA	NA	4107380	Z%
⋮				

Left join in this context

In words: "keep all students from the student-level data; drop any school from the school-level data that doesn't merge onto a student"

- **Main or "left" dataset**

Student	Year	District	NCES ID
Jeremy	2021	New Trier High School	1728200
Emma	2022	Hanover High	3302670
Esmeralda	2022	Homeschool	NA
⋮			

- **Auxiliary or "right" dataset**

District	NCES ID	% FRPL
New Trier HS	1728200	X%
Hanover HS	3302670	Y%
Lebanon HS	4107380	Z%
⋮		

Right join in this context

In words: "drop students who don't have a school in the school-level data; keep all schools from the student-level data—even those that don't merge onto any student"

- **Main or "left" dataset**

Student	Year	District	NCES ID
Jeremy	2021	New Trier High School	1728200
Emma	2022	Hanover High	3302670
Esmeralda	2022	Homeschool	NA
⋮			

- **Auxiliary or "right" dataset**

District	NCES ID	% FRPL
New Trier HS	1728200	X%
Hanover HS	3302670	Y%
Lebanon HS	4107380	Z%
⋮		

DataCamp versus slide syntax

- DataCamp modules generally used this syntax for merges:

```
merged_df = df1.merge(df2,
                      how = "left",
                      on = "something")
```

- Slides/solution code will tend to use this syntax:

```
merged_df = pd.merge(df1, df2,
                     how = "left",
                     on = "something")
```

- They produce identical answers so use whichever comes more naturally (I use latter)
- In addition, feel free to use self joins if useful but we won't be focusing a lot on those

How do we code these different types of joins? Example with left join and same colname for join key

```
1
2 ## perform a left join on the student data
3 ## and schools data
4 stud_wschool = pd.merge(students,
5                         schools,
6                         how = "left",
7                         on = "NCES ID",
8                         indicator = "student_mergesource")
```

- `how`: argument to tell it inner, left, right, or outer; defaults to inner
- `on`: name of join key (in this case single key)
- `indicator`: optional arg to add a col to result (default name "_merge") indicating source ("left_only", "right_only", or "both"); helpful for merge status & post-merge diagnostics

Example with inner join and different colname for join key

```
1
2 ## perform a left join on the student data
3 ## and schools data
4 stud_wschool = pd.merge(students,
5                         schools,
6                         how = "inner",
7                         left_on = "NCES ID",
8                         right_on = "ncesnumeric")
```

Example with left join and multiple join keys

```
1
2 ## perform a left join on the student data
3 ## and schools data
4 stud_wschool = pd.merge(students,
5                         schools,
6                         how = "left",
7                         left_on = ["NCES ID",
8                         "Dist name"],
9                         right_on = ["ncesnumeric",
10                        "distnamechar"],
11                        indicator = "student_mergesource")
```

Example with left join, multiple join keys, and some overlapping, non-join columns that we want to differentiate

```
1
2 ## perform a left join on the student data
3 ## and schools data
4 stud_wschool = pd.merge(students,
5                         schools,
6                         how = "left",
7                         left_on = ["NCES ID",
8                         "Dist name"],
9                         right_on = ["ncesnumeric",
10                        "distnamechar"],
11                        indicator = "student_mergesource",
12                        suffixes = ("_students",
13                                    "_schools"))
```

Non-exhaustive checklist of merge diagnostics

1. How many rows were in each dataset before the merge? What about after?
2. If doing a left join, did we properly retain all left-hand side rows?
3. **For strings as join keys:** if a lot of rows were lost in a merge, could that be due to spelling/punctuation variations in a character join key?
4. **For numeric identifiers as join keys:** if a lot of rows were lost in a merge, could that be due to things like the id having leading or lagging zeros and those being stripped at some stage? (e.g., one dataset identifies an entity as 002548; another as 2548)

Next up: basic regex to improve match rates for strings as join keys

- In example below, what if we didn't have the NCES ID numeric identifier? Ways to improve match rates for spelling variations (sometimes called `entity resolution`)

Student	**Year**	**District**
Jeremy	2021	New Trier High School
Emma	2022	Hanover High
Esmeralda	2022	Homeschool
⋮		

District	**% FRPL**
New Trier HS	X%
Hanover HS	Y%
Lebanon HS	Z%
⋮	

Overview of activity data

- `public_data/sd_df.csv`: sample of business tax certificates for San Diego-based businesses—each row represents one unique business; cols for industry (6-digit NAICS code)

	account_key	dba_name	council_district	naics_code	naics_description	naics_nchar
	1974000448	ERNST & YOUNG LLP	cd_1	541211	OFFICES OF CERTIFIED PUBLIC ACCOUNTANTS	6
	1974011093	HECHT SOLBERG ROBINSON GOLDBERG & BAGLEY LLP	cd_3	5411	LEGAL SERVICES	4
	1978039819	RSM US LLP	cd_1	541211	OFFICES OF CERTIFIED PUBLIC ACCOUNTANTS	6
	1978042092	THORSNES BARTOLOTTA MCGUIRE LLP	cd_3	5411	LEGAL SERVICES	4
	1979046817	KORENIC & WOJDOWSKI LLP	cd_7	5412	ACCOUNTING/TAX PREP/BOOKKEEP/PAYROLL SERVICES	4

- `public_data/naics_df.csv`: exhaustive listing of all 6-digit NAICS codes from the Census Bureau with added information

naics	naics_description
111140	Wheat Farming
111160	Rice Farming
111150	Corn Farming
111110	Soybean Farming
111120	Oilseed (except Soybean) Farming

- **General goal:** match the two to investigate things like which industries are *not* represented in San Diego small businesses

Pause for practice

Merging part (section 2) of `03_reshaping_merging.ipynb`

- If interested, notebook where data was cleaned/prepped to make easier to analyze: `03_helper_merging_dataprep.ipynb`