PPOL564: Modern Statistical Computing

Unit 05: Reshaping and merging (exact)

Where today fits in

- **Today:**
  - **Reshaping data**
  - **Exact merging**: have identifier with exact matches between two datasets
- **Next week:** basic regular expressions (regex) to help:
  1. Clean join fields for exact matching/merges
  2. Clean join fields for fuzzy/probabilistic matching/merges
- **Later week or optional session:** fuzzy/probabilistic merging/record linkage

Where we are

- **Reshaping data**
- Merging data

Two general formats for data

1. **Wide format data:** each row is one unit (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 (eg a person at one point in time; a state with one economic summary measure); each unit might have multiple rows

Example contrast

Wide format:

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: repeated across years and type of statistic (truncated for student 2)

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
⋮			

"Pivoting" from long to wide

```
pd.pivot(longformat_df,
     index= 'col_unitinfo',
     columns= 'col_repeatmeasure',
     values = ['col1_value', 'col2_value', ...])
```

Breaking it down:

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

"Melting" from wide to long

```
pd.melt(coffee_df_wide, id_vars = 'shop_name')
```

Breaking it down:

- **id_vars:** the name of the column you're treating as the unit of analysis that has repeated measures
- See documentation for optional arguments that help us rename the output: https://pandas.pydata.org/docs/reference/api/pandas.melt.html

Pause for practice

Pause for practice

Reshaping section (section 1) of `06_reshaping_merging`

Where we are

- Reshaping data
- **Merging data**

Working example

- **Main or "left" dataset:** college data on student's high schools

Student	**Year**	**District**	**NCES ID**
Rebecca	2021	New Trier High School	1728200
Jennifer	2022	Bethesda High	3302670
Jason	2022	Homeschool	NA
⋮			

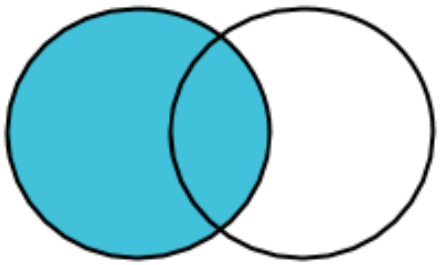
- **Auxiliary or "right" dataset**- FRPL = free or reduced price lunch eligible percentage; used as (one) measure of school district poverty

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

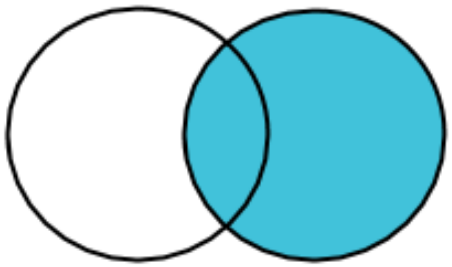
Possible join keys

- **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:** what's the likelihood that "New Trier district" and "New Trier HS" are the same entity?

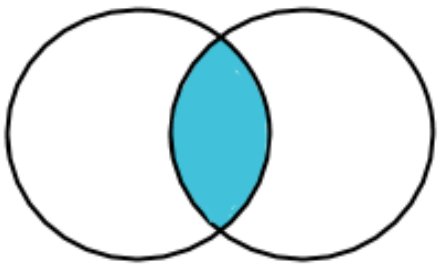
Conceptual overview of four types of joins



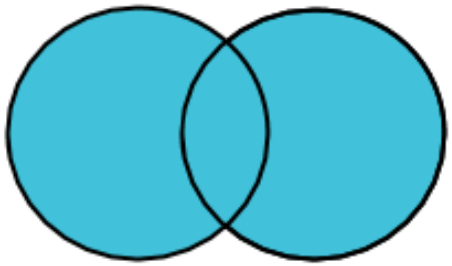
Left Join



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 Georgetown student data"

- **Main or "left" dataset**

Student	Year	District	NCES ID
Rebecca	2021	New Trier High School	1728200
Jennifer	2022	Bethesda High	3302670
Jason	2022	Homeschool	NA
⋮			

- **Auxiliary or "right" dataset**

District	NCES ID	% FRPL
New Trier HS	1728200	X%
Bethesda HS	3302670	Y%
Arundel 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**
Rebecca	2021	New Trier High School	1728200	X%
Jennifer	2022	Bethesda High	3302670	Y%
Jason	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
Rebecca	2021	New Trier High School	1728200
Jennifer	2022	Bethesda High	3302670
Jason	2022	Homeschool	NA
⋮			

- **Auxiliary or "right" dataset**

District	NCES ID	% FRPL
New Trier HS	1728200	X%
Bethesda HS	3302670	Y%
Arundel 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
Rebecca	2021	New Trier High School	1728200
Jennifer	2022	Bethesda High	3302670
Jason	2022	Homeschool	NA
⋮			

- **Auxiliary or "right" dataset**

District	NCES ID	% FRPL
New Trier HS	1728200	X%
Bethesda HS	3302670	Y%
Arundel 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 because it's more similar to base `R` syntax)
- 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 in practice? Example with left join and join key has same colname in both

```
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_mergestatus")
```

- `how`: argument to tell it inner, left, right, outer, or cross; defaults to inner
- `on`: name of join key (in this case single key)
- `indicator`: optional arg to add a col to the resulting data (string is what to call it) that helps diagnose merge status; good for post-merge dx

Example with inner join and join key has different name

```
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_mergestatus")
```

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_mergestatus",
12                        suffixes = ["_students",
13                                    "_schools"])
```

Non-exhaustive checklist of merge diagnostics

1. How many rows were in each data 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**
Rebecca	2021	New Trier High School
Jennifer	2022	Bethesda High
Jason	2022	Homeschool
⋮		

District	**% FRPL**
New Trier HS	X%
Bethesda HS	Y%
Arundel 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
2	1978039819	RSM US LLP	cd_1	541211	OFFICES OF CERTIFIED PUBLIC ACCOUNTANTS	6
3	1978042092	THORSNES BARTOLOTTA MCGUIRE LLP	cd_3	5411	LEGAL SERVICES	4
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 the San Diego small businesses

Pause for practice

Pause for practice

Merging section (section 2) of `06_reshaping_merging`