Lecture 21
Types of Joins

Dennis Sun
Stanford University
DATASCI / STATS 112

March 3, 2023

1 Review

2 Types of Joins

3 Many-to-Many Joins

4 Summary

5 Reminders

Joins

Sometimes data is spread across multiple datasets.

For example, suppose we have baby names in 1920 and 2020:

```
import pandas as pd
data_dir = "http://dlsun.github.io/pods/data/names/"

df_1920 = pd.read_csv(data_dir + "yob1920.txt", header=None,
                      names=["Name", "Sex", "Count"])
df_2020 = pd.read_csv(data_dir + "yob2020.txt", header=None,
                      names=["Name", "Sex", "Count"])
```

Joins

`df_1920`

	Name	Sex	Count
0	Mary	F	70975
1	Dorothy	F	36645
2	Helen	F	35098
3	Margaret	F	27997
4	Ruth	F	26100
...	...	...	...
10751	Zearl	M	5
10752	Zeferino	M	5
10753	Zeke	M	5
10754	Zera	M	5
10755	Zygmont	M	5

10756 rows × 3 columns

`df_2020`

	Name	Sex	Count
0	Olivia	F	17641
1	Emma	F	15656
2	Ava	F	13160
3	Charlotte	F	13065
4	Sophia	F	13036
...	...	...	...
31448	Zykell	M	5
31449	Zylus	M	5
31450	Zymari	M	5
31451	Zyn	M	5
31452	Zyran	M	5

31453 rows × 3 columns

We can *join* two datasets on a *key*.

We focused on the case where we join on a **primary key**.

In this case, we are joining the primary keys of two tables together. (We could also join the primary key to a **foreign key**.)

Joins

```
df_joined = df_1920.merge(df_2020, on=["Name", "Sex"])
df_joined
```

	Name	Sex	Count_x	Count_y
0	Mary	F	70975	2210
1	Dorothy	F	36645	562
2	Helen	F	35098	721
3	Margaret	F	27997	2190
4	Ruth	F	26100	1323
...	...	...	...	...
4473	Whitt	M	5	23
4474	Wyley	M	5	6
4475	Xavier	M	5	3876
4476	York	M	5	14
4477	Zeke	M	5	382

4478 rows × 4 columns

Missing Keys?

```
df_joined[df_joined["Name"] == "Maya"]
```

Name	Sex	Count_x	Count_y

Why isn't Maya in the joined data? How does Pandas determine which keys show up?

It is there in the 2020 data...

```
df_2020[df_2020["Name"] == "Maya"]
```

	Name	Sex	Count
60	Maya	F	3724
28914	Maya	M	6

...but not in the 1920 data.

```
df_1920[df_1920["Name"] == "Maya"]
```

Name	Sex	Count

In order to appear in the joined data, a key must be present in *both* tables.

1 Review

2 Types of Joins

3 Many-to-Many Joins

4 Summary

5 Reminders

Types of Joins

How can we customize the behavior of joins for missing keys?

This brings us to the first of today's topics: *types of joins*.

- By default, Pandas does an **inner join**, which only keeps keys that are present in *both* tables.
- An **outer join** keeps any key that is present in either table.
- A **left join** keeps all keys in the left table, even if they are not in the right table. But any keys that are only in the right table are dropped.
- A **right join** keeps all keys in the right table, even if they are not in the left table. But any keys that are only in the left table are dropped.

Code Example

We can customize the type of join using the `how=` parameter of `.merge()`. By default, `how="inner"`.

```
df_joined_outer = df_1920.merge(df_2020, on=["Name", "Sex"],
                                how="outer")
df_joined_outer[df_joined_outer["Name"] == "Maya"]
```

	Name	Sex	Count_x	Count_y
10771	Maya	F	NaN	3724.0
35372	Maya	M	NaN	6.0

Note the missing values for other columns, like Count, for 1920!

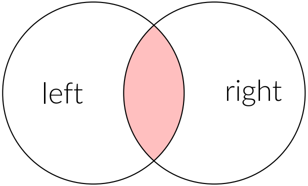
What other type of join would have produced this output?

```
df_joined_right = df_1920.merge(df_2020, on=["Name", "Sex"],
                                how="right")
df_joined_right[df_joined_right["Name"] == "Maya"]
```

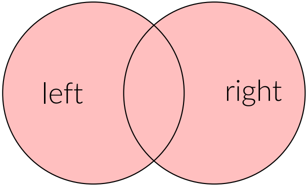
	Name	Sex	Count_x	Count_y
60	Maya	F	NaN	3724
28914	Maya	M	NaN	6

Summary of Types of Joins

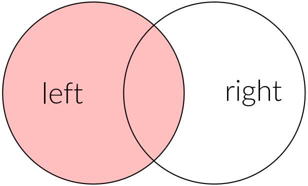
INNER JOIN



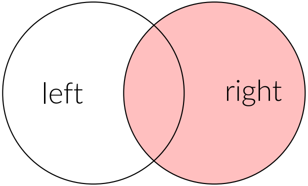
(FULL) OUTER JOIN



LEFT JOIN



RIGHT JOIN



Exercises

Which type of join would be best suited for each case?

1. We want to determine the names that have increased in popularity the most between 1920 and 2020.

```
df_1920.merge(df_2020, on=["Name", "Sex"], how=...)
```

`how="right"` (to include names that didn't appear at all in the 1920 data)

2. We want to graph the popularity of names over time.

```
df = pd.read_csv(data_dir + "yob1981.txt",
                 header=None,
                 names=["Name", "Sex", 1981])

for year in range(1982, 2021):
  df_year = pd.read_csv(data_dir + f"yob{year}.txt",
                        header=None,
                        names=["Name", "Sex", year])
  df = df.merge(df_year, on=["Name", "Sex"], how=...)
```

`how="outer"` (to include rare names that go in and out of the data)

1 Review

2 Types of Joins

3 Many-to-Many Joins

4 Summary

5 Reminders

Many-to-Many Relationships

So far, the keys we've joined on have been the primary key of (at least) one table.

- If we join to the primary key of another table, then the relationship is **one-to-one** (since primary keys uniquely identify rows).
- If we join to the foreign key of another table, then the relationship is **one-to-many**.

What if we join on a key that is not a primary key?
That is, what if the key does not uniquely identify rows in either table so that each value of the key might appear multiple times?

This type of join is called **many-to-many**.

Example

What if we only joined on the name?

```
df_1920.merge(df_2020, on="Name")
```

	Name	Sex_x	Count_x	Sex_y	Count_y
0	Mary	F	70975	F	2210
1	Mary	F	70975	M	5
2	Mary	M	195	F	2210
3	Mary	M	195	M	5
4	Dorothy	F	36645	F	562
...	...	...	...	...	...
6158	Xavier	M	5	F	7
6159	Xavier	M	5	M	3876
6160	York	M	5	F	6
6161	York	M	5	M	14
6162	Zeke	M	5	M	382

6163 rows × 5 columns

Why does Mary appear 4 times in this data?

A Diagram

`df_1920`

	Name	Sex	Count
0	Mary	F	70975
⋮	⋮	⋮	⋮
6195	Mary	M	195
⋮	⋮	⋮	⋮
⋮	⋮	⋮	⋮

`df_2020`

	Name	Sex	Count
⋮	⋮	⋮	⋮
122	Mary	F	2210
⋮	⋮	⋮	⋮
30759	Mary	M	5
⋮	⋮	⋮	⋮

There are 4 matches, only 2 of which are desirable.

Preventing Bugs

Most of the time, many-to-many joins are a bug, caused by a misunderstanding about the primary key.

Pandas allows us to specify the relationship we are expecting. It will fail with an error if the relationship is a different kind.

For example, suppose we thought that "name" was the primary key of the baby name tables.

```
df_1920.merge(df_2020, on="Name",
              validate="one_to_one")
```

```
MergeError: Merge keys are not unique in either left or right dataset;
            not a one-to-one merge
```

Errors are (sometimes) your friend. They can prevent you from making even bigger mistakes!

1. Review
2. Types of Joins
3. Many-to-Many Joins
4. Summary
5. Reminders

What We've Learned Today

We've discussed two kinds of complications with joins:

- when a key doesn't appear in one table (outer, left, right join)
- when a key appears multiple times in both tables (many-to-many joins)

You should be equipped to do Assignment 7, due next Friday. (Don't forget Assignment 6, due tonight.)

1. Review

2. Types of Joins

3. Many-to-Many Joins

4. Summary

5. Reminders

Exam 2

- Exam 2 is in class next Monday. Same policy as last time (1 page of handwritten notes allowed).
- The exam includes material up to Monday (hierarchical clustering). It does not include material from Wednesday or today.
- I have posted a practice exam. Solutions are posted as well.

Final Project

- Sign up for a final project presentation here: [link to form].
- The final project files are due on Canvas on Wednesday 3/22 at 11:59 PM.