Data Wrangling (I): Munging, Tidy Data, and Working with Multiple Data Tables

Nicholas Mattei, *Tulane University*

CMPS3660 – Introduction to Data Science – Fall 2019

https://rebrand.ly/TUDataScience

Many Thanks
Slides based off Introduction to Data Science from John P. Dickerson -
https://cmsc320.github.io/

Announcements

- Project1 and Milestone1 Updates
  - Reading really important here!
- Survey Results!
- Lab 4 + Lab 5
- Weekly Questions 4
- On to DATA!

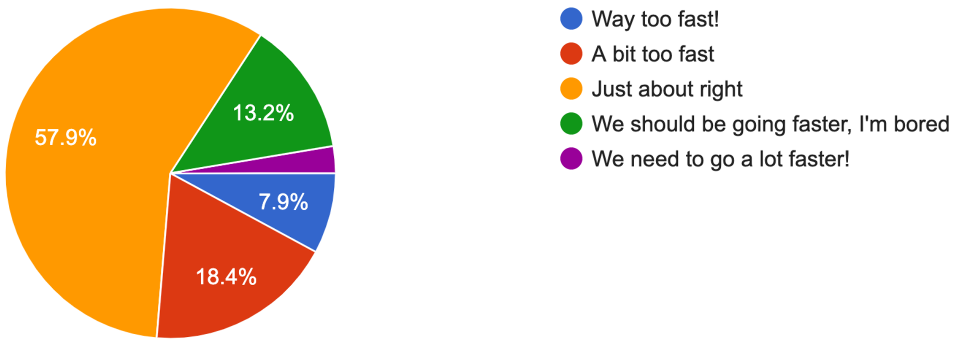
Survey Take Home Messages

- 28+ of you like or really like notebooks!
  - 2 of you really hate them!
- Some of you say I'm talking too fast!
- 6-7 of you Hate Docker!

- What we want (that I'll deliver):
  - More Depth, More Theory (5)
  - PPT/Lectures (5-7)?
    - Too much, Not enough; interesting, boring; clear, too muddled
  - More Feedback!
- What we want (I can't fix).
  - Class too early.
  - Too much programming.

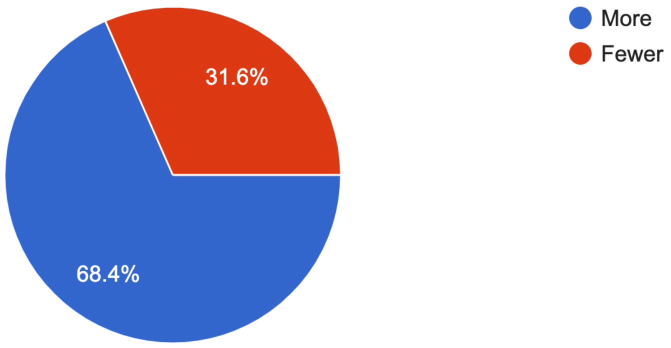
The course is moving...

38 responses



I'd like to do [ANSWER] Lab Days

38 responses



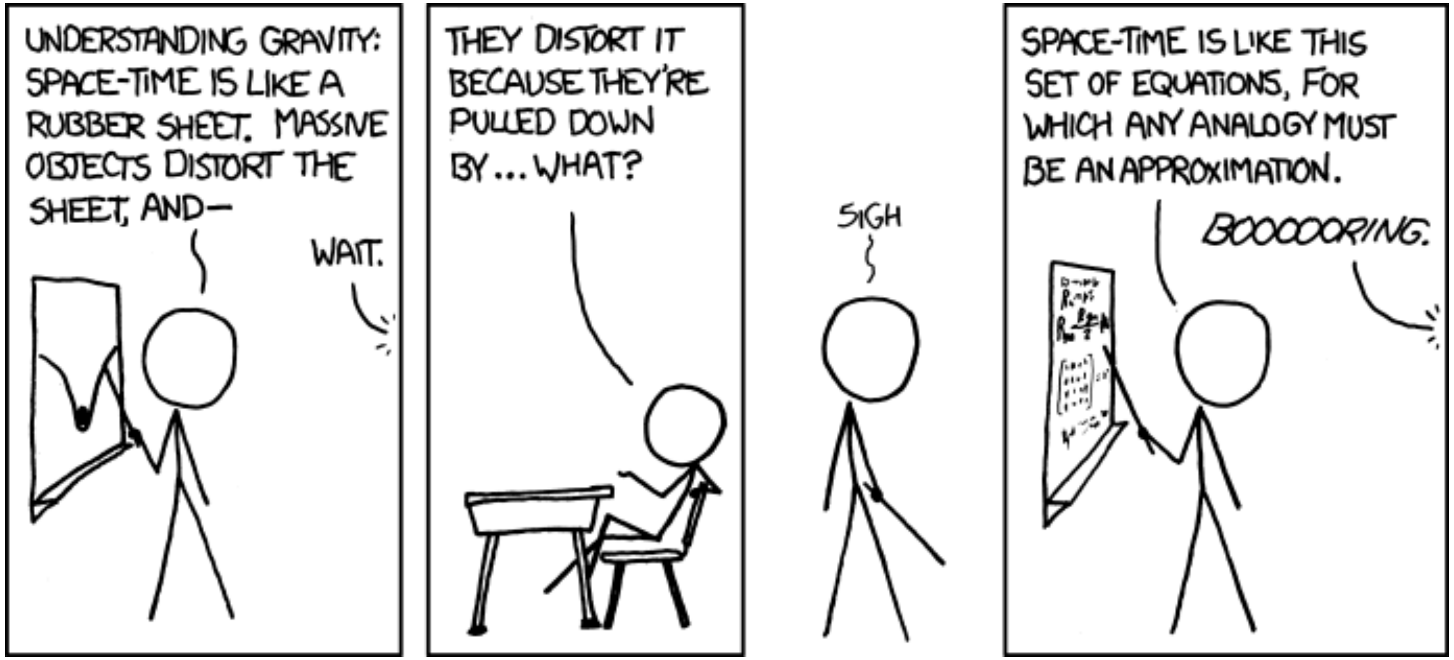
Next Couple of Lectures (Till Midterm)

- Tables in the Abstract
  - How, Why
  - Operations

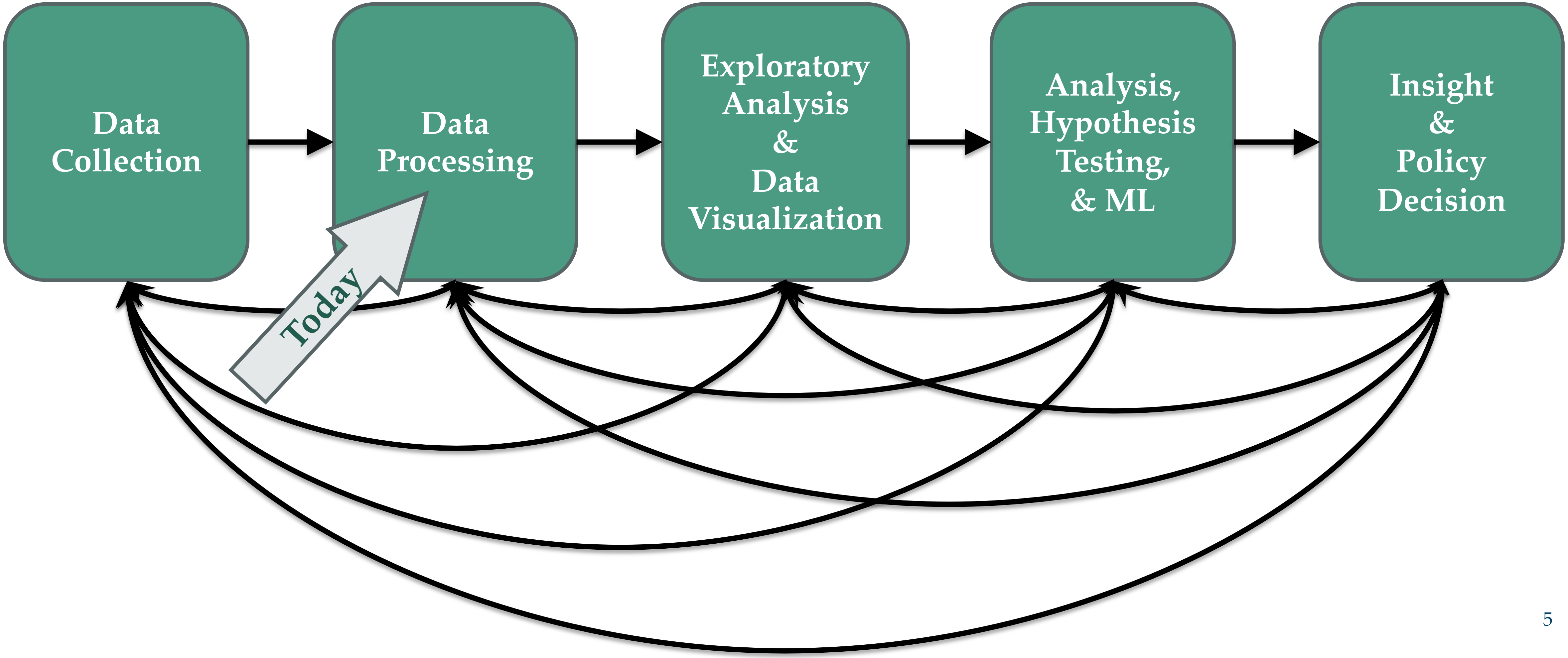
- Principles of Tidy Data

- Tables in Pandas
- Tables in SQL and RMDBS

- 2 More Labs.



The Data LifeCycle



Tulane University

Tables

Special Column, called “Index”, or “ID”, or “Key”
Usually, no duplicates Allowed

Variables
(also called Attributes, or Columns, or Labels)

Observations, Rows, or Tuples

ID	age	wgt_kg	hgt_cm
1	12.2	42.3	145.1
2	11.0	40.8	143.8
3	15.6	65.3	165.3
4	35.1	84.2	185.8

Entries or values

1. Select/Slicing

- Select only some of the rows, or some of the columns, or a combination.

ID	age	wgt_kg	hgt_cm
1	12.2	42.3	145.1
2	11.0	40.8	143.8
3	15.6	65.3	165.3
4	35.1	84.2	185.8

Only columns
ID and Age

ID	age
1	12.2
2	11.0
3	15.6
4	35.1

Only rows
with wgt > 41

ID	age	wgt_kg	hgt_cm
1	12.2	42.3	145.1
3	15.6	65.3	165.3
4	35.1	84.2	185.8

Both

ID	age
1	12.2
3	15.6
4	35.1

2. Aggregate/Reduce

- Combine values across a column into a single value

ID	age	wgt_kg	hgt_cm
1	12.2	42.3	145.1
2	11.0	40.8	143.8
3	15.6	65.3	165.3
4	35.1	84.2	185.8

SUM

73.9	**232.6**	**640.0**

MAX

35.1	**84.2**	**185.8**

SUM(wgt_kg^2 - hgt_cm)

14167.66

What about ID/Index column?

Usually not meaningful to aggregate across it

May need to explicitly add an ID column

Practical Interlude: *np.nan*

- We use numpy.nan to signify a value is missing or not a number.
- If we don't use NaN's then Pandas doesn't know how to handle the data.
- Breaks in all sorts of awful ways.
- (Demo Notebook)

ID	age	wgt_kg	hgt_cm
1	12.2	42.3	145.1
2	11.0	40.8	143.8
3	15.6	65.3	165.3
4	--	84.2	185.8

SUM

ERROR!

ID	age	wgt_kg	hgt_cm
1	12.2	42.3	145.1
2	11.0	40.8	143.8
3	15.6	65.3	165.3
4	np.NaN	84.2	185.8

SUM

38.8

Tulane University

3. Map

- Apply a function to every row, possibly creating more or fewer columns

ID	Address
1	College Park, MD, 20742
2	Washington, DC, 20001
3	Silver Spring, MD 20901

SPLIT(“,”)

ID	City	State	Zipcode
1	College Park	MD	20742
2	Washington	DC	20001
3	Silver Spring	MD	20901

Variations that allow one row to generate multiple rows in the output (sometimes called “flatmap” or “melt” as we’ll see later.)

4. Group By

- Group tuples together by column/dimension.

ID	A	B	C
1	foo	3	6.6
2	bar	2	4.7
3	foo	4	3.1
4	foo	3	8.0
5	bar	1	1.2
6	bar	2	2.5
7	foo	4	2.3
8	foo	3	8.0

By 'A' →

A = foo

ID	B	C
1	3	6.6
3	4	3.1
4	3	8.0
7	4	2.3
8	3	8.0

A = bar

ID	B	C
2	2	4.7
5	1	1.2
6	2	2.5

4. Group By

- Group tuples together by column/dimension.

ID	A	B	C
1	foo	3	6.6
2	bar	2	4.7
3	foo	4	3.1
4	foo	3	8.0
5	bar	1	1.2
6	bar	2	2.5
7	foo	4	2.3
8	foo	3	8.0

By 'B' →

B = 1

ID	A	C
5	bar	1.2

B = 2

ID	A	C
2	bar	4.7
6	bar	2.5

B = 3

ID	A	C
1	foo	6.6
4	foo	8.0
8	foo	8.0

B = 4

ID	A	C
3	foo	3.1
7	foo	2.3

4. Group By

- Group tuples together by column/dimension.

ID	A	B	C
1	foo	3	6.6
2	bar	2	4.7
3	foo	4	3.1
4	foo	3	8.0
5	bar	1	1.2
6	bar	2	2.5
7	foo	4	2.3
8	foo	3	8.0

By ‘A’, ‘B’ →

A = bar, B = 1

ID	C
5	1.2

A = bar, B = 2

ID	C
2	4.7
6	2.5

A = foo, B = 3

ID	C
1	6.6
4	8.0
8	8.0

A = foo, B = 4

ID	C
3	3.1
7	2.3

5. Group By Aggregate

- Group the aggregate per group.

ID	A	B	C
1	foo	3	6.6
2	bar	2	4.7
3	foo	4	3.1
4	foo	3	8.0
5	bar	1	1.2
6	bar	2	2.5
7	foo	4	2.3
8	foo	3	8.0

Group by 'B'
Sum on C

B = 1

ID	A	C
5	bar	1.2

B = 1

Sum (C)
1.2

B = 2

ID	A	C
2	bar	4.7
6	bar	2.5

B = 2

Sum (C)
7.2

B = 3

ID	A	C
1	foo	6.6
4	foo	8.0
8	foo	8.0

B = 3

Sum (C)
22.6

B = 4

ID	A	C
3	foo	3.1
7	foo	2.3

B = 4

Sum (C)
5.4

5. Group By Aggregate

- Final result usually seen as a table.

ID	A	B	C
1	foo	3	6.6
2	bar	2	4.7
3	foo	4	3.1
4	foo	3	8.0
5	bar	1	1.2
6	bar	2	2.5
7	foo	4	2.3
8	foo	3	8.0

Group by 'B'
Sum on C

B = 1

Sum (C)
1.2

B = 2

Sum (C)
7.2

B = 3

Sum (C)
22.6

B = 4

Sum (C)
5.4

B	SUM(C)
1	1.2
2	7.2
3	22.6
4	5.4

Tulane University

5.5 Pivot Tables (Data Cubes)

- Laying out the possible values of multiple axes and aggregating them.
  - Can have more than two dimensions, need hierarchal indexes (later).

ID	A	B	C
1	foo	3	6.6
2	bar	2	4.7
3	foo	4	3.1
4	foo	3	8.0
5	bar	1	1.2
6	bar	2	2.5
7	foo	4	2.3
8	foo	3	8.0

Index A, Columns B

Values C, Agg=Sum

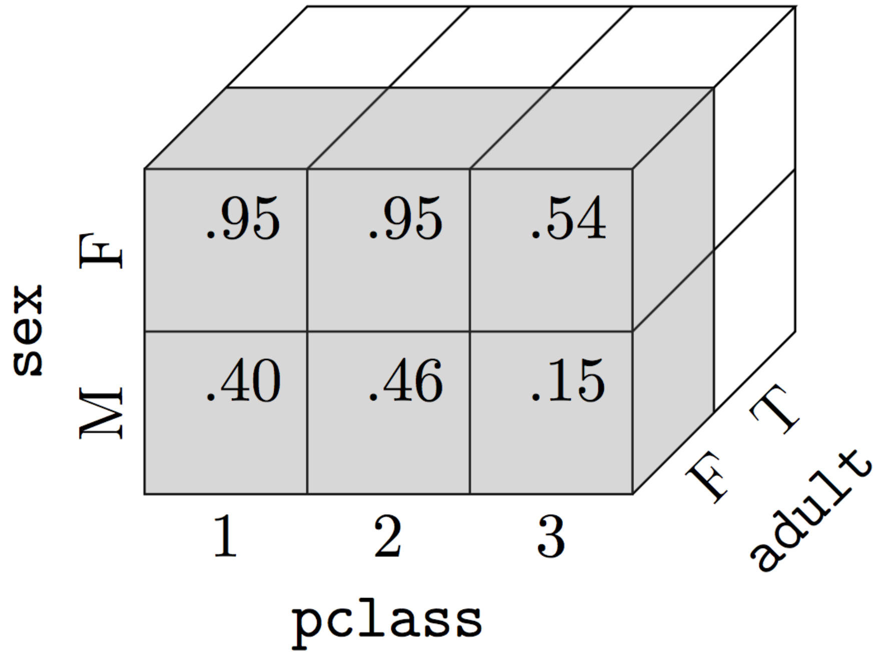
A B>	1	2	3	4
foo	0	0	22.6	5.4
bar	1.2	7.2	0	0

5.5 Pivot Tables (Data Cubes)

- Laying out the possible values of multiple axes and aggregating them.
  - Can have more than two dimensions, need hierarchal indexes (later).

```
survivors_cube = titanic_df.pivot_table(
    index="sex", columns=["adult", "pclass"],
    values="survived", aggfunc=np.mean)
survivors_cube
```

adult	False			True		
pclass	1	2	3	1	2	3
sex						
female	0.947368	0.952381	0.536364	0.968000	0.870588	0.443396
male	0.400000	0.464286	0.147059	0.326389	0.083916	0.155709



6. Union/Intersection/Difference

- Set operations – only if the two tables have identical attributes/columns

ID	A	B	C
1	foo	3	6.6
2	bar	2	4.7
3	foo	4	3.1
4	foo	3	8.0

U

ID	A	B	C
5	bar	1	1.2
6	bar	2	2.5
7	foo	4	2.3
8	foo	3	8.0

→

ID	A	B	C
1	foo	3	6.6
2	bar	2	4.7
3	foo	4	3.1
4	foo	3	8.0
5	bar	1	1.2
6	bar	2	2.5
7	foo	4	2.3
8	foo	3	8.0

Similarly Intersection and Set Difference manipulate tables as Sets

IDs may be treated in different ways, resulting in somewhat different behaviors

7. Merge or Join

- Combine rows/tuples across two tables *if they have the same key.*

- This example is called an *Inner Join*

ID	A	B
1	foo	3
2	bar	2
3	foo	4
4	foo	3

$\bowtie$

ID	C
1	1.2
2	2.5
3	2.3
5	8.0

→

ID	A	B	C
1	foo	3	1.2
2	bar	2	2.5
3	foo	4	2.3

What about IDs not present in both tables?

Often need to keep them around

Can “pad” with NaN (depends on software!)

7. Merge or Join (Outer or Full Join)

- Combine rows/tuples across two tables if they have the same key.
- Outer joins can be used to ”pad” IDs that don’t appear in both tables
  - Three variants: LEFT, RIGHT, FULL
  - SQL Terminology -- Pandas has these operations as well

ID	A	B
1	foo	3
2	bar	2
3	foo	4
4	foo	3

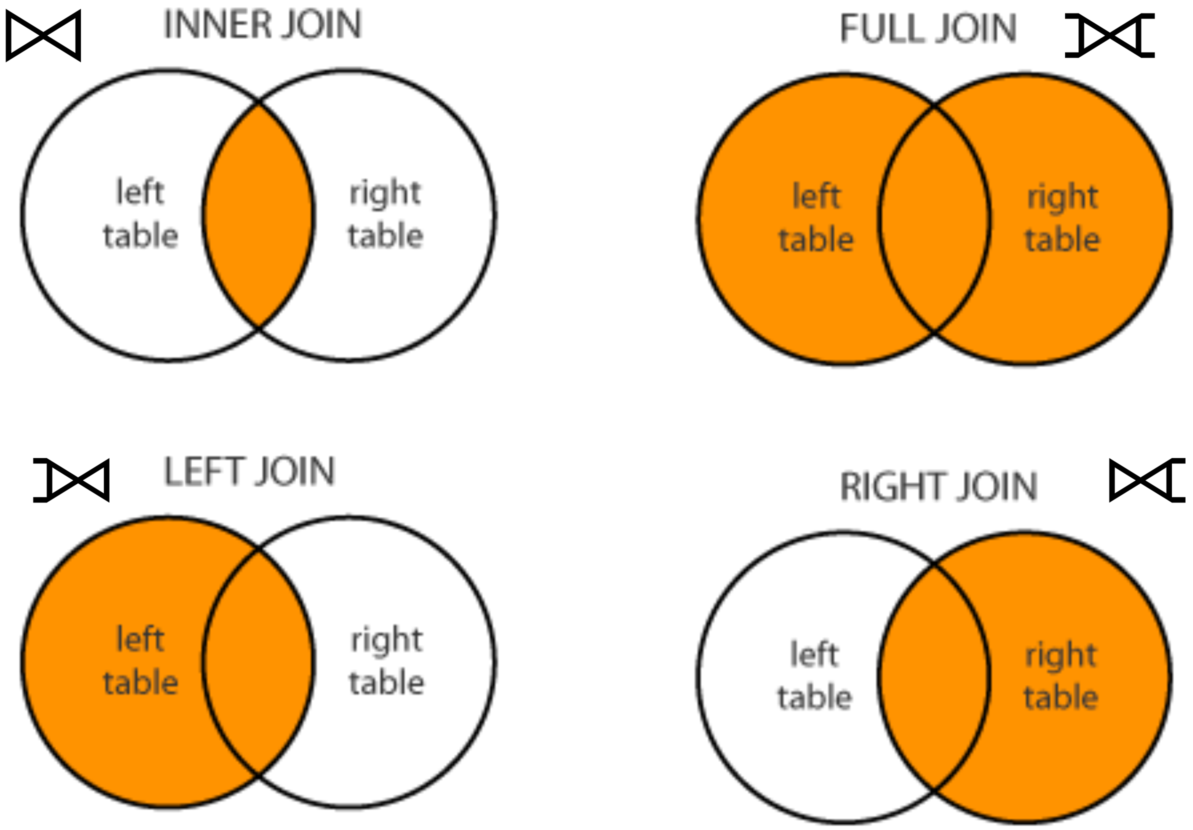
$\bowtie$

ID	C
1	1.2
2	2.5
3	2.3
5	8.0

→

ID	A	B	C
1	foo	3	1.2
2	bar	2	2.5
3	foo	4	2.3
4	foo	3	NaN
5	NaN	NaN	8.0

Types of Joins



In Pandas this is called a FULL OUTTER JOIN!

Image credit: http://www.dofactory.com/sql/join

7. Merge or Join (Left Join)

- Combine rows/tuples across two tables if they have the same key.
- Outer joins can be used to ”pad” IDs that don’t appear in both tables
  - Three variants: LEFT, RIGHT, FULL
  - SQL Terminology -- Pandas has these operations as well

ID	A	B
1	foo	3
2	bar	2
3	foo	4
4	foo	3

$\bowtie$

ID	C
1	1.2
2	2.5
3	2.3
5	8.0

$\rightarrow$

ID	A	B	C
1	foo	3	1.2
2	bar	2	2.5
3	foo	4	2.3
4	foo	3	NaN

7. Merge or Join (Right Join)

- Combine rows/tuples across two tables if they have the same key.
- Outer joins can be used to ”pad” IDs that don’t appear in both tables
  - Three variants: LEFT, RIGHT, FULL
  - SQL Terminology -- Pandas has these operations as well

ID	A	B
1	foo	3
2	bar	2
3	foo	4
4	foo	3

⟖

ID	C
1	1.2
2	2.5
3	2.3
5	8.0

ID	A	B	C
1	foo	3	1.2
2	bar	2	2.5
3	foo	4	2.3
5	NaN	NaN	8.0

Quick Review

- Tables: A simple, common abstraction
  - Subsumes a set of "strings" – a common input, or a list of lists, or a list of dicts with the same keys.

- Operations on tables:
  - Select, Map, Aggregate, Reduce, Join/Merge, Union/Concat, Group By

- ***These may have different names!*** In Pandas it's a *merge* while in SQL it's a *join.*
  - Actually, this isn't quite right -- Pandas has a *join* command that will only join based on the *index!* It also has a *merge* command that allows for more options – see Lab 7!
  - Pandas also uses *merge* as we'll see in lab while SQL uses Union

- There can be subtle variations in implementation on different data systems. Remember I'm giving you the high level but you need to *read the docs for your software* when you use this stuff!