

Computing in Context: Fall 2024

Lecture 2 | Data in Practice: the 80:20 rule

” **80 percent** of a data scientist’s valuable time is spent simply finding, cleansing, and organizing data, leaving only 20 percent to actually perform analysis...

IBM Data Analytics

Cleaning Data

It's all relative!

“Structured” vs. “Clean” Data

From last lecture: “Less than 15 percent of health data in EHRs are entered in structured data fields.” (Roski et al., 2014)

Even when data is “structured,” this doesn’t mean it’s “clean”

- “**clean**”: when data is structured, edited, and/or formatted such that the desired use/analysis can be performed on it without further preparation

Reliability and Replicability in Data Cleaning

Reliable – error free and accurate, you can rely on it

Replicable – recreate the “clean” data again given the raw data

We want both

Reliability and Replicability in Data Cleaning

Code allows you to automate repetitive tasks and while reducing errors that emerge during manual edits.

GUI-based alternatives (like Excel) don't offer the flexibility, transparency, reliability, and replicability that code does

Code tells you exactly how a piece of data was transformed and executing it will give you the same results every time.

Unnecessary Variables in Data

Quick note:

Often, an analysis only requires a handful of variables to conduct.

When assembling your data for analysis, feel free to drop unnecessary variables along the way if you find it helpful.

If your data “big,” this will free up memory on your computer.

“Clean” is a Relative Term

“clean”: when data is structured, edited, and/or formatted such that the desired use/analysis can be performed on it without further preparation

“clean” is relative: you may find that a dataset might be clean for the purposes of one type of analysis, but not for another

Street Addresses Example

person_id	home_address
1	722 w 168 th st 10032
2	722 w 168 th ave, new york 10032
3	722 W 168 th street, new York, new york 10032
4	722 west 168 th , new York, new york 10032
5	722 West 168 th St, New York, New York 10032

Desired Format: 722 West 168th St, New York, NY 10032

Suppose that a hospital collects home address information for their patients. The hospital wants to adopt a standardized address format to reduce the rate of undelivered mail.

Street Addresses Example



Google Geocoding API

Now suppose researchers wanted to use the same address field to identify the Census Tract for each address, to determine SES of the patient population.

The street address would not be useful. It must instead be **“geocoded”** (assigned latitude and longitude coordinates).

Conclusion: the 80/20 Rule

Given the wide variety of uses for any data point, data preparation/cleaning important part of data science.

The “80/20 Rule” [one of many]: data scientists generally spend about 80% of their time cleaning data, and only about 20% of their time analyzing it

- Obviously, this can vary tremendously by setting/role

Data Merging

Levels of analysis and data merges

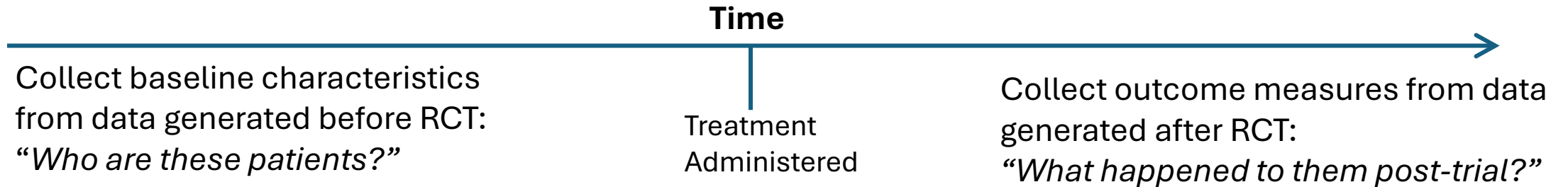
Level of Analysis

Before you begin preparing dataset(s) for analysis, you must determine the **level of analysis** of the study.

What is the **level of analysis**? Envision what your final (clean) DataFrame will look like – what do the rows represent?

Note: This may differ from the initial **unit(s) of observation** (rows) in the various data sources that you'll draw upon your analysis. You may need to restructure this raw data accordingly.

Level of Analysis



Example: Imagine that you are extracting EHR data from a hospital database relating to patients that took part in an RCT. You want to collect baselines (1 year pre-trial) and outcomes (1 year post-trial) to assess the efficacy of treatment in the RCT. E.g.

- Results from lab tests
- Procedures performed
- Diagnoses

Level of Analysis

Level of Analysis

Data in EHR systems like a lot of “big data,” are stored in **relational databases**. Each category of data will exist in its own table(s), with unique “keys” to link them to specific patients.

The **unit of observation** in each table can vary. E.g., each row of the *lab results* table relates to an individual lab test result, each row of the *procedures* table relates to an individual procedure performed. Each patient may have all, some, or none of these.

Your goal is to analyze the impact of the RCT on *patients* (your **level of analysis**), so you will have to summarize or collapse this information to the level of the individual patient!

Level of Analysis

Units of observation (labs/procedures) don't always match to your level of analysis (patients)!

patient_id	DOB	sex	group
1	09/06/84	M	Treatment
2	07/05/85	F	Control
3	06/27/57	F	Treatment
4	01/28/87	M	Control
5	10/30/16	M	Control

patient_id	lab_id	date	lab_value
1	123836	1/12/2018	0.512
1	211687	4/26/2018	300
1	135818	9/02/2018	1.24
3	123836	3/12/2018	51
3	365581	5/26/2018	987
4	981048	7/02/2018	1547
5	873871	8/15/2018	62
patient_id	proc_id	date	proc_code
1	5181	1/12/2018	0.512
2	2116	4/26/2018	300
4	5581	9/02/2018	1.24
4	1238	3/12/2018	51
5	5581	5/26/2018	987
5	1048	7/02/2018	1547
5	8871	8/15/2018	62

labs

procedures

Merging data

patient_df

patient_id	DOB	sex	group
1	09/06/84	M	Treatment
2	07/05/85	F	Control
3	06/27/57	F	Treatment
4	01/28/87	M	Control
5	10/30/16	M	Control

labs_df

patient_id	lab_id	date	lab_value
1	123836	1/12/2018	0.512
1	211687	4/26/2018	300
1	135818	9/02/2018	1.24
3	123836	3/12/2018	51
3	365581	5/26/2018	987
4	981048	7/02/2018	1547
7	873871	8/15/2018	62

Example: you want to combine your patient and labs DFs, but there some patients that appear only in one and not the other. Moreover, some patients have multiple labs. How can this data be merged?

Merging Data

4 types of merges between two DataFrames, (one “left” and one “right”).
E.g., ‘patient_id’ is the **key** that you would use to merge these tables:

Left Joins: “Use only keys from the left DF”

- only keeps observations from right DF if keys exist in left DF

Right Joins: “Use only keys from the right DF”

- only keeps observations from left DF if keys exist in right DF

Inner Joins [default]: “Use only keys common to both DFs”

- only keeps observations with keys present in both DFs

Outer Joins: “Use all keys”

- keeps all observations from both DFs

Left Joins

Left Joins: “Use only keys from the left DF”

– only keeps obs from right DF if keys exist in left DF

```
left_joined_df = patient_df.merge(labs_df, on = 'patient_id', how = 'left')
```

patient_df

patient_id	DOB	sex	group
1	09/06/84	M	Treatment
2	07/05/85	F	Control
3	06/27/57	F	Treatment
4	01/28/87	M	Control
5	10/30/16	M	Control



labs_df

patient_id	lab_id	date	lab_value
1	123836	1/12/2018	0.512
1	211687	4/26/2018	300
1	135818	9/02/2018	1.24
3	123836	3/12/2018	51
3	365581	5/26/2018	987
4	981048	7/02/2018	1547
7	873871	8/15/2018	62



left_joined_df

patient_id	DOB	sex	group	lab_id	date	lab_value
1	9/6/1984	M	Treatment	123836	1/12/2018	0.512
1	9/6/1984	M	Treatment	211687	4/26/2018	300
1	9/6/1984	M	Treatment	135818	9/2/2018	1.24
2	7/5/1985	F	Control	NaN	NaN	NaN
3	6/27/1957	F	Treatment	123836	3/12/2018	51
3	6/27/1957	F	Treatment	365581	5/26/2018	987
4	1/28/1987	M	Control	981048	7/2/2018	1547
5	10/30/2016	M	Control	NaN	NaN	NaN

Right Joins

Right Joins: “Use only keys from the right DF”
– only keeps obs from left DF if keys exist in right DF

```
right_joined_df = patient_df.merge(labs_df, on = 'patient_id', how = 'right')
```

patient_df

patient_id	DOB	sex	group
1	09/06/84	M	Treatment
2	07/05/85	F	Control
3	06/27/57	F	Treatment
4	01/28/87	M	Control
5	10/30/16	M	Control



labs_df

patient_id	lab_id	date	lab_value
1	123836	1/12/2018	0.512
1	211687	4/26/2018	300
1	135818	9/02/2018	1.24
3	123836	3/12/2018	51
3	365581	5/26/2018	987
4	981048	7/02/2018	1547
7	873871	8/15/2018	62



right_joined_df

patient_id	DOB	sex	group	lab_id	date	lab_value
1	9/6/1984	M	Treatment	123836	1/12/2018	0.512
1	9/6/1984	M	Treatment	211687	4/26/2018	300
1	9/6/1984	M	Treatment	135818	9/2/2018	1.24
3	6/27/1957	F	Treatment	123836	3/12/2018	51
3	6/27/1957	F	Treatment	365581	5/26/2018	987
4	1/28/1987	M	Control	981048	7/2/2018	1547
7	NaN	NaN	NaN	873871	8/15/2018	62

Inner Joins

Inner Joins: “Use only keys common to both DFs”

– only keeps obs with keys present in both DFs

```
inner_joined_df = patient_df.merge(labs_df, on = 'patient_id', how = 'inner')
```

patient_df

patient_id	DOB	sex	group
1	09/06/84	M	Treatment
2	07/05/85	F	Control
3	06/27/57	F	Treatment
4	01/28/87	M	Control
5	10/30/16	M	Control



labs_df

patient_id	lab_id	date	lab_value
1	123836	1/12/2018	0.512
1	211687	4/26/2018	300
1	135818	9/02/2018	1.24
3	123836	3/12/2018	51
3	365581	5/26/2018	987
4	981048	7/02/2018	1547
7	873871	8/15/2018	62



inner_joined_df

patient_id	DOB	sex	group	lab_id	date	lab_value
1	9/6/1984	M	Treatment	123836	1/12/2018	0.512
1	9/6/1984	M	Treatment	211687	4/26/2018	300
1	9/6/1984	M	Treatment	135818	9/2/2018	1.24
3	6/27/1957	F	Treatment	123836	3/12/2018	51
3	6/27/1957	F	Treatment	365581	5/26/2018	987
4	1/28/1987	M	Control	981048	7/2/2018	1547

Outer Joins

Outer Joins: “Use all keys” – keeps all obs from both DFs

```
outer_joined_df = patient_df.merge(labs_df, on = 'patient_id', how = 'outer')
```

patient_df

patient_id	DOB	sex	group
1	09/06/84	M	Treatment
2	07/05/85	F	Control
3	06/27/57	F	Treatment
4	01/28/87	M	Control
5	10/30/16	M	Control



labs_df

patient_id	lab_id	date	lab_value
1	123836	1/12/2018	0.512
1	211687	4/26/2018	300
1	135818	9/02/2018	1.24
3	123836	3/12/2018	51
3	365581	5/26/2018	987
4	981048	7/02/2018	1547
7	873871	8/15/2018	62



outer_joined_df

patient_id	DOB	sex	group	lab_id	date	lab_value
1	9/6/1984	M	Treatment	123836	1/12/2018	0.512
1	9/6/1984	M	Treatment	211687	4/26/2018	300
1	9/6/1984	M	Treatment	135818	9/2/2018	1.24
2	7/5/1985	F	Control			
3	6/27/1957	F	Treatment	123836	3/12/2018	51
3	6/27/1957	F	Treatment	365581	5/26/2018	987
4	1/28/1987	M	Control	981048	7/2/2018	1547
5	10/30/2016	M	Control	NaN	NaN	NaN
7	NaN	NaN	NaN	873871	8/15/2018	62

Merging Data: 1-to-1 or 1-to-Multiple?

Merging DFs on a 1-to-1 basis –one row in a table corresponds to only one row in another DF – is straightforward.

If rows in one DF correspond to multiple rows in another DF, you may need to collapse these rows to your **level of analysis**.

We will use Pandas DataFrame tools like *group.by()*, *melt()*, *pivot()*, and others to solve this.

Missing Data & Outliers

Why, what, when?

Missing Data

It is *very* common to have missing values in your data – that is, for some rows (observations), one or more variables (columns) will be “empty”

- In Pandas, this is represented by cells containing **NaN** or **None***

- 1) Why might you have missing data?
- 2) What should you do about it?

The answer to 2) may depend on 1).

The use of NaN is preferred in Pandas – for consistency, try to avoid using None.

Why is this Data Missing?

Some typical reasons why data is missing:

- User forgot to fill in a field
- Data was lost while transferring manually from a legacy database, or from a non-digital source
- There was a programming error
- Users chose not to fill out a field

Always ask yourself: is the missing-ness random? If not, and it's correlated with outcomes of interest, consider how this may change your analysis.

Why is this Data Missing?

Example: Suppose that a health clinic asks its patients to fill out a short questionnaire at the beginning of each visit.

“Since your last visit, have you consumed any drugs recreationally?”

1% responded “Yes,” 90% responded “No,” 9% left their answer blank.

Is it reasonable to assume that these missing values are unrelated to patients’ drug use?

Probably not.

Some Common Missing Data Solutions

Drop the observations with any missing values from the dataset

Exclude observations with a missing values on a column-by-column basis:

Impute missing values

Sensitivity Analysis – Never a Bad Idea

A **sensitivity analysis** helps to see whether your conclusions change under various assumptions.

How? See what happens to your estimates when you replace missing values with a variety of different values.

Estimating the mean age of patients who visit a health clinic, but some of the values are missing? What happens to your estimate of this mean when you replace missing values with the 1st %-tile of age? The 99th %-tile? The 75th? The 25th?

“Standard” vs “Non-Standard” Missing Values

So far we’ve been implicitly discussing missing data as though Pandas functions will always recognize missing values, and label them appropriately – **standard missing values**.

However, it’s often the case that a dataset has missing values that are clear to the human eye, but not necessarily to Pandas, which can result in observations inappropriately being categorized as non-missing – **non-standard missing values**.

“Non-Standard” Missing Values

ST_NUM	ST_NAME	NUM_BEDROOMS	OWN_OCCUPIED
104	PUTNAM	3	Y
197	LEXINGTON	3	N
	LEXINGTON	n/a	N
201	BERKELEY	1	12
203	BERKELEY	3	Y
207	BERKELEY	NA	Y
NA	WASHINGTON	2	
213	TREMONT	--	Y
215	TREMONT	na	Y



```
# Looking at the NUM_BEDROOMS column  
print df['NUM_BEDROOMS']  
print df['NUM_BEDROOMS'].isnull()
```

Out:

```
0      3  
1      3  
2    n/a  
3      1  
4      3  
5    NaN  
6      2  
7     --  
8     na
```

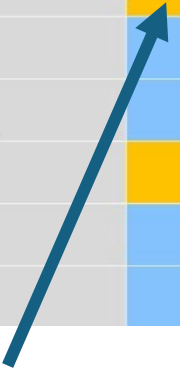
Out:

```
0    False  
1    False  
2    False  
3    False  
4    False  
5    True  
6    False  
7    False  
8    False
```

By default, Pandas recognized ‘NA’ as a missing value (and replaced it with **NaN**), but not the others. You’ll need to code these as missing yourself.

Unexpected Missing Values

ST_NUM	ST_NAME	NUM_BEDROOMS	OWN_OCCUPIED
104	PUTNAM	3	Y
197	LEXINGTON	3	N
	LEXINGTON	n/a	N
201	BERKELEY	1	12
203	BERKELEY	3	Y
207	BERKELEY	NA	Y
NA	WASHINGTON	2	
213	TREMONT	--	Y
215	TREMONT	na	Y



Other times, an observation may have nonsensical values – usually code these as missing as well.

Should familiarize yourself with your data before attempting to conduct any analysis to identify data issues.

Tools like the “Variable Explorer” in Spyder can help, especially for small datasets.

Summarizing each variable (using tools like the DataFrame `describe()`, `count()`, and `value_counts()` methods) will also help you to identify unexpected values in the data.

Outliers

Outliers

Outliers – extreme values that fall well outside of the norm for a given type of data. They can have a meaningful impact on estimates generated from the data.

How do you handle this?

Example: Examining a part of a hospital's EHR data that was hand-entered during patient intake. You come across a female patient with a recorded height of 86 inches (7'2")

Outliers

Outliers in data generally fall into one of two categories:

1. The outlier is an accurate representation of a rare phenomenon (e.g. this woman really is 7'2")
2. The outlier was generated erroneously (e.g. meant to type "68 inches," not "86 inches" in the patient's EHR).

#1 then you might want to leave this observation in the data as-is.

#2, change the outlier to another value, or record it as missing?

Outliers

Outliers unduly skew your estimates? But contain information that you don't want to do away, **winsorizing** a good approach

Example: estimating the mean lab test value from an imprecise blood test. Values are partially informative, but noisy.

Winsorizing: restrict the minimum and/or maximum values of numeric variable e.g., replace all values above the 99th percentile with the value corresponding to the 99th percentile.

Transparency

No single solution for cleaning “dirty” data.

Most important is to be *transparent* about the issue, and what you did to address it.

- This enables others to appropriately interpret your analysis.
- Document, document, document!

Questions?