Week 4: Causality Wrap-Up, Python Sequences and Libraries

DSUA111: Data Science for Everyone, NYU, Fall 2020

TA Jeff, `jpj251@nyu.edu`

- This slideshow: [https://jjacobs.me/dsua111-sections/week-04 (https://jjacobs.me/dsua111-sections/week-04)](https://jjacobs.me/dsua111-sections/week-04)
- All materials: [https://github.com/jpowerj/dsua111-sections (https://github.com/jpowerj/dsua111-sections)](https://github.com/jpowerj/dsua111-sections)

Outline

1. HW1 Feedback
2. Causality Wrap-Up
3. Python Sequences
4. Python Libraries

0. HW1 Feedback

- [NYU Classes -> Gradebook (https://newclasses.nyu.edu/portal/site/609e7da1-32c6-4cf5-bf0a-d7e589f4f5c5)](https://newclasses.nyu.edu/portal/site/609e7da1-32c6-4cf5-bf0a-d7e589f4f5c5)

1. Causality Wrap-Up

Where we left off:

Fundamental Problem of Causal Inference: Forget Everything And Run?

Face Everything And Rise

- Find good **comparison** cases: **Treatment Group** and **Control Group**

- "Statistical Matching"
    - Don't worry about the details, but tldr is:
    - Find the two **most similar** people, put one in Treatment Group, the other in Control Group, and compare their outcomes
    - Bam. If we can measure and take into account all variables that may be related to our causal hypothesis, this is **as close as we can possibly get** to "solving" FPCI
    - [Not on the midterm or final, but relevant in case you're despairing about FPCI]

Controlled Experiments: How/Why Do They Help?

- Random Assignment: Vietnam War/Second Indochina War Draft
    - Key point: makes treatment and control groups similar, on average, without us having to do any work!
    - (e.g., don't need to worry about "pairing up" similar treatment+control units via statistical matching)
- No more Selection Effects
- Omitted variables are in BOTH Treatment and Control groups

Complications: Selection

- Tldr: **Why** did this person (unit) end up in the **treatment** group? **Why** did this other person (unit) end up in the **control** group?
    - Are there systematic differences?
- Vietnam/Indochina Draft: Why can't we just study **[men who join the military]** versus **[men who don't]**, and take the difference as a causal estimate?

Complications: Compliance

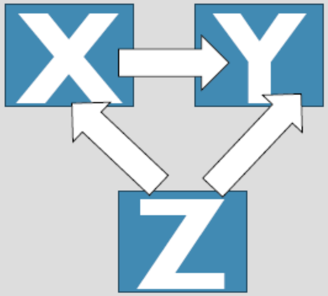
- We ideally want people **assigned** to the treatment group to **take** the treatment, and people **assigned** to the control group to **take** the control.
- "Compliance" is the degree to which this is actually true in your experiment
    - **High** compliance = most people actually took what they were assigned
    - **Low** compliance = lots of people who were assigned to treatment actually took control, and vice-versa
- What problems might there be with compliance in the Draft example?

The Biggest Complication: Observational Data

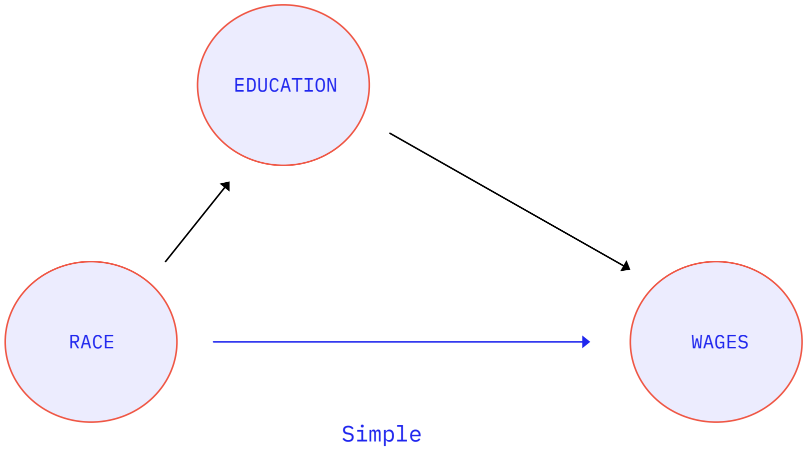
- In observational studies, researchers have no control over assignment to treatment/control 😨
- On the one hand... Forget Everything And Run, if you can.
- On the other hand... statisticians over the last ~4 centuries have developed fancy causal inference tools/techniques to help us Face Everything And Rise!

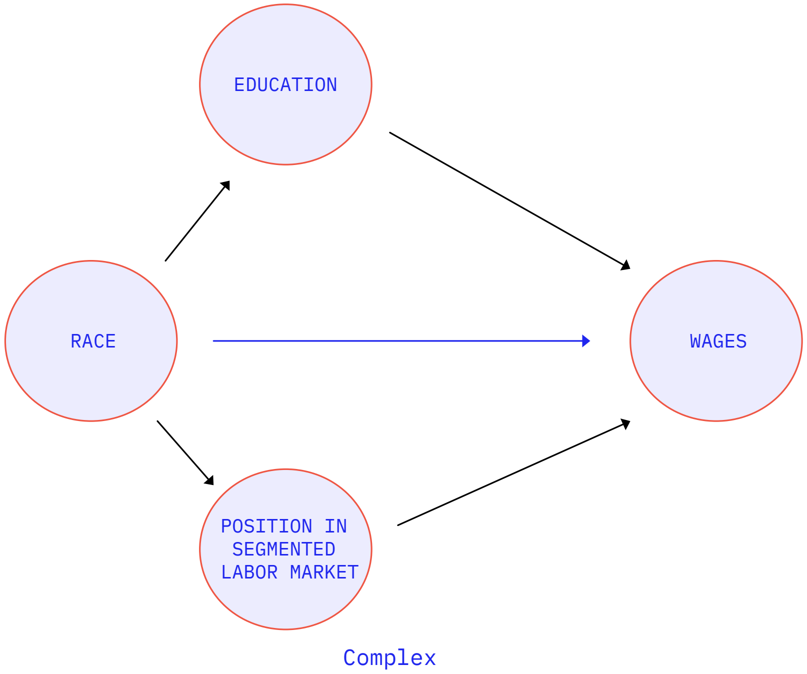
Causal Terminology for Observational Studies

- We have an outcome we want to explain. Call that the ***dependent variable*** or **Y**.
- We have a treatment/control that does the explaining. Call that the ***independent variable*** or ***X***.
- We may have a ***confounder***, ***Z*** which is causing/affecting both X and Y.
- In that case, there may be ***no causal relationship at all*** between X and Y. The relationship between X and Y may be ***spurious***.



(This is... genuinely important + relevant imo)

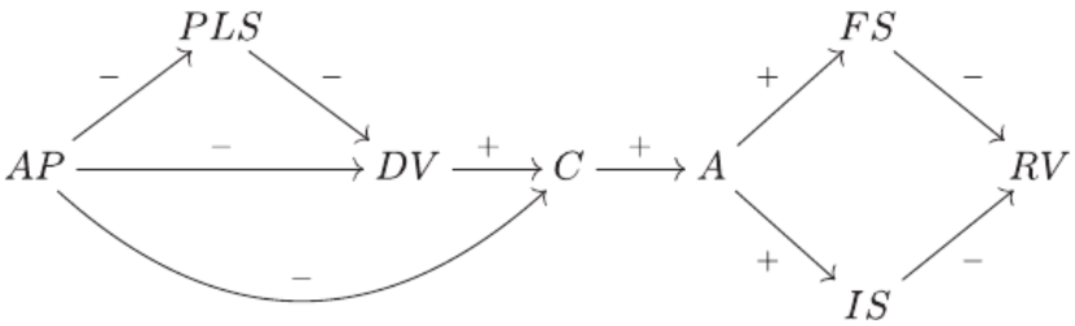




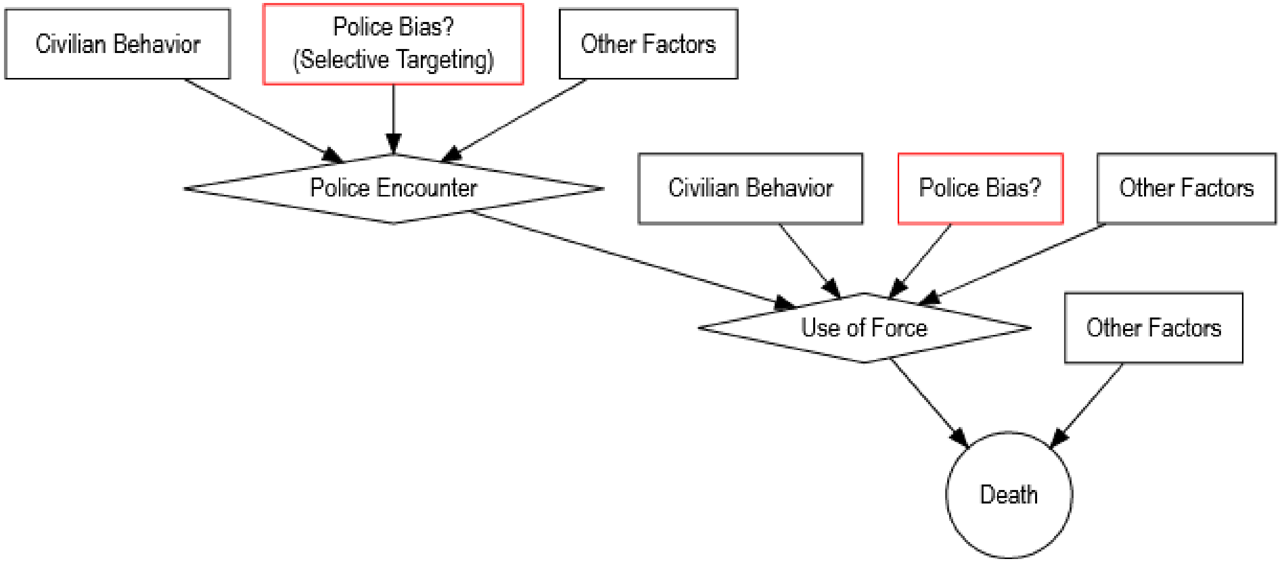
from Hu (2020), ["Direct Effects" (https://phenomenalworld.org/analysis/direct-effects)](https://phenomenalworld.org/analysis/direct-effects)

FIGURE 10

Combined Causal Graph of Domestic Violence



from Sampson, Winship, and Knight (2013), ["Translating Causal Claims: Principles and Strategies for Policy-Relevant Criminology" (https://onlinelibrary.wiley.com/doi/abs/10.1111/1745-9133.12028)](https://onlinelibrary.wiley.com/doi/abs/10.1111/1745-9133.12028)



from Bradford (2020), ["Observations on Police Shootings & Interracial Violence" (https://rpubs.com/johnbradford/policeShootingGraphs)](https://rpubs.com/johnbradford/policeShootingGraphs)

2. Python Sequences

Where we left off:

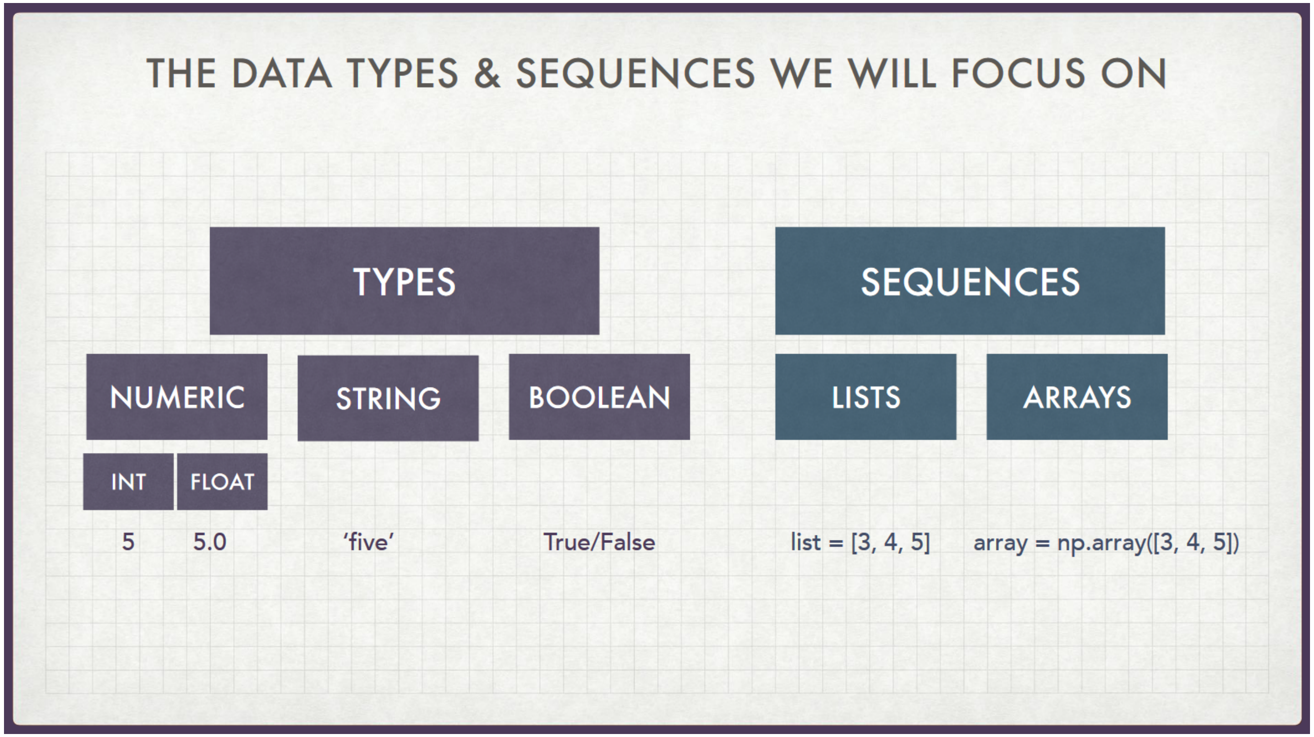
In [113]:

```
my_string = "Jeff"
my_float = 5.5
my_string * my_float
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-113-e6906dc4821b> in <module>
      1 my_string = "Jeff"
      2 my_float = 5.5
----> 3 my_string * my_float

TypeError: can't multiply sequence by non-int of type 'float'
```

Sequences?



(Lecture 6.2 slides, p. 5)

Jeff's TLDR

- Lists: Ordered sequences of... anything (including lists themselves 🤯)

In [114]:
```
my_basic_list = [1, "one", 1.0]
print(my_basic_list)
```

```
[1, 'one', 1.0]
```

In [115]:
```
my_meta_list = [my_basic_list, 2, "two", 2.0, [3, "three", 3.0]]
print(my_meta_list)
```

```
[[1, 'one', 1.0], 2, 'two', 2.0, [3, 'three', 3.0]]
```

- This is **not** the same as

In [116]:
```
my_non_meta_list = my_basic_list + [2, "two", 2.0] + [3, "three", 3.0]
print(my_non_meta_list)
```

```
[1, 'one', 1.0, 2, 'two', 2.0, 3, 'three', 3.0]
```

- ^(!!!)

Ordered?

- We specify "ordered" only because there are also `sets`, which contain elements but have no notion of a "first", "second", "third" element

In [117]:
```
ordered_list = [4, 3, 2, 1]
set(ordered_list)
```

Out[117]:
```
{1, 2, 3, 4}
```

Arrays are just fancier lists (for doing fancier math)

In [118]:
```
ordered_array = np.array(ordered_list)
```

- What happened?

Remember to import NumPy first!

In [119]:

```
import numpy as np
ordered_array = np.array(ordered_list)
print(ordered_array)
print(ordered_list)
```

```
[4 3 2 1]
[4, 3, 2, 1]
```

In [120]:

```
ordered_list.mean()
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-120-46a02c4663bc> in <module>
----> 1 ordered_list.mean()

AttributeError: 'list' object has no attribute 'mean'
```

In [121]:

```
ordered_array.mean()
```

Out[121]:

```
2.5
```

Important difference, though: NumPy arrays require that all elements be the **same type**!

...so what happens if they're different?

In [122]:
```
no_bueno = np.array([1,"two",3.0])
no_bueno
```

Out[122]:
```
array(['1', 'two', '3.0'], dtype='<U11')
```

In [123]:
```
print(type(no_bueno[0]))
print(type(no_bueno[1]))
print(type(no_bueno[2]))
```

```
<class 'numpy.str_'>
<class 'numpy.str_'>
<class 'numpy.str_'>
```

Bracket Madness

- [ ] (**square** brackets): `list`
- { } (**curly** brackets): `set` (or `dict`)
- ( ) (**parentheses**): `tuple`

In [124]:
```
type([1, 2, 3])
```
Out[124]:
```
list
```

In [125]:
```
type({1, 2, 3})
```
Out[125]:
```
set
```

In [126]:
```
type({1: "one", 2: "two", 3: "three"})
```
Out[126]:
```
dict
```

In [127]:
```
type((1, 2, 3))
```
Out[127]:
```
tuple
```

3. Python Libraries

(From most basic to most fancy)

1. NumPy: `import numpy as np`

$\longrightarrow$ Math with **arrays**

2. Pandas: `import pandas as pd`

$\longrightarrow$ Math with **Tables** (DataFrames)

3. Matplotlib: `import matplotlib.pyplot as plt`

$\longrightarrow$ Visualizing NumPy/Pandas objects

4. **Statsmodels**: `import statsmodels.formula.api as smf`

$\longrightarrow$ Statistical hypothesis testing

5. **Seaborn**: `import seaborn as sns`

$\longrightarrow$ Visualizing statistical hypothesis tests

6. **Scikit-learn**: `import sklearn`

$\longrightarrow$ Fancy machine learning things

NumPy

In [142]:

```
import numpy as np
cool_array = np.array([1, 2, 3, 4, 5])
cool_array.std()
```

Out[142]:

```
1.4142135623730951
```

Pandas

In [143]:

```
import pandas as pd
cool_df = pd.DataFrame(
    {'x':[1,2,3,4,5,6,7,8],
     'y':[2,4,5,5,7,9,10,13],
     'z':[0,0,1,0,1,1,1,1]}
)
cool_df.head()
```

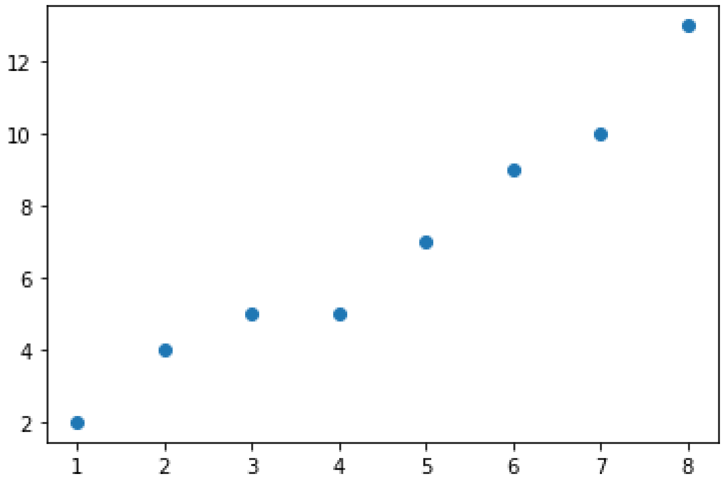
Out[143]:

	x	y	z
0	1	2	0
1	2	4	0
2	3	5	1
3	4	5	0
4	5	7	1

Matplotlib

In [144]:

```
import matplotlib.pyplot as plt
plt.scatter(cool_df['x'], cool_df['y'])
plt.show()
```



Statsmodels

In [166]:

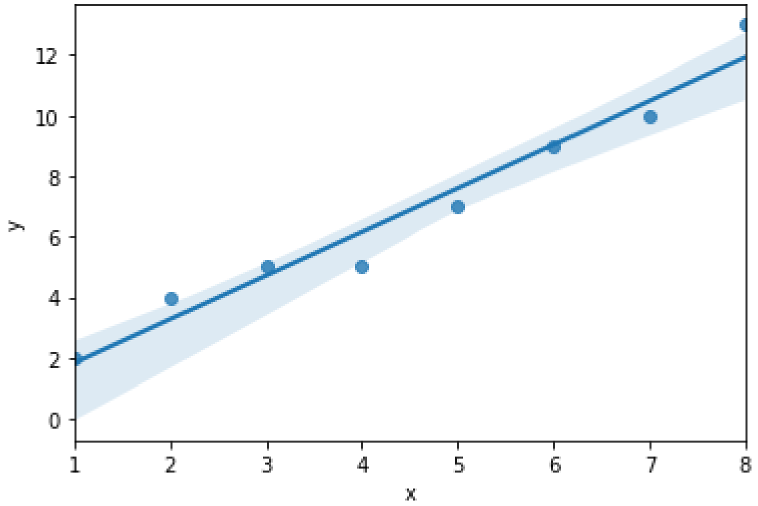
```
import statsmodels.formula.api as smf
result = smf.ols('y ~ x', data=cool_df).fit()
print(result.summary().tables[1])
```

```
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
Intercept      0.3929      0.614      0.640      0.546      -1.110       1.895
x              1.4405      0.122     11.846      0.000       1.143       1.738
==============================================================================
```

Seaborn

In [146]:

```
import seaborn as sns
sns.regplot(x='x', y='y', data=cool_df);
```



Scikit-learn

In [147]:

```
import sklearn
```

In [181]:

```
import scikitplot as skplt
skplt.metrics.plot_confusion_matrix(classes, predictions, normalize=True)
plt.show()
```