Matching

Professor Song Yao
Olin Business School

Customer Analytics

Non-random Assigned Treatmed and Confounding Bias

Matching Example Revisit

EXAMPLE: For-profit education industry (E.g., Coursera, Udacity)

Student attrition is a major problem

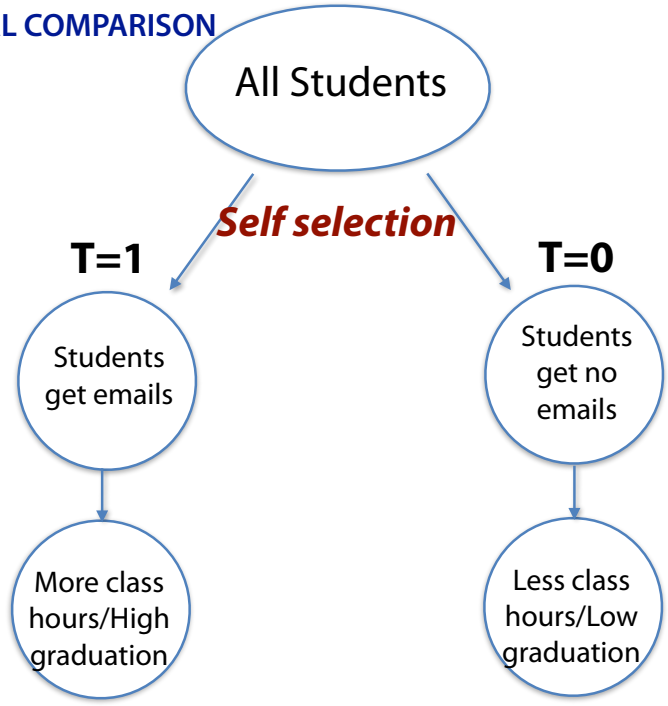
What keeps students on track?

One key relationship:

- Students who sign up weekly emails take more classes and are more likely to graduate.
- But is this causal?

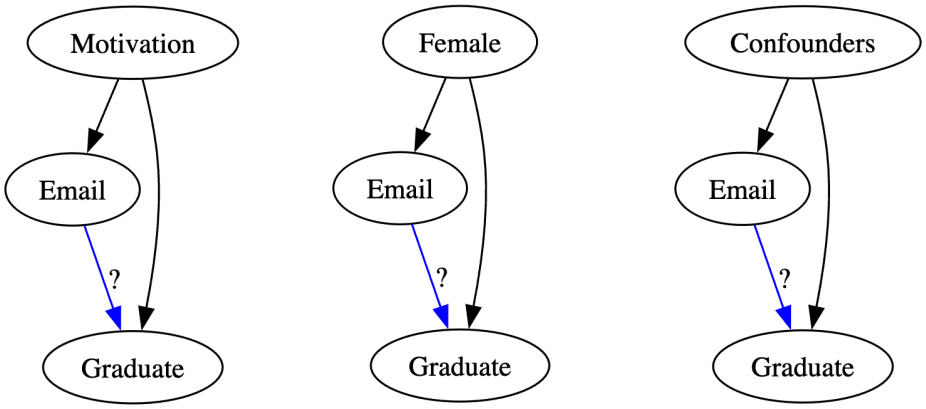
We cannot convincingly attribute the high graduation rate to Email (at least not completely)

NON-EXPERIMENTAL COMPARISON



We cannot convincingly attribute the high graduation rate to Email (at least not completely)

Confounding Bias



Consider a toy example

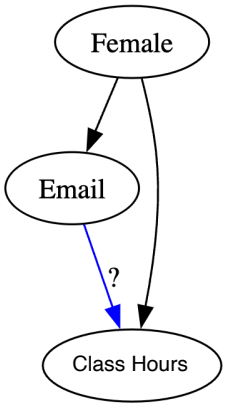
Effect of Email on Class Hours: Male 10hrs, Female 5hrs

```
email_example = pd.DataFrame(dict(
    gender= ["M","M","M","M","M","M", "F","F","F","F"],
    email=[1,1,0,0,0,0,  1,1,1,0],
    hours=[80,80,70,70,70,70,  90,90,90,85]
))

print(email_example.query('gender == "F"'))
print(email_example.query('gender == "M"'))
```

```
  gender  email  hours
6      F      1     90
7      F      1     90
8      F      1     90
9      F      0     85
  gender  email  hours
0      M      1     80
1      M      1     80
2      M      0     70
3      M      0     70
4      M      0     70
5      M      0     70
```

Female,
1. More hours in the first place
2. More likely to sign up
3. Effect is smaller b/c their higher baseline



The Intuition behind Matching

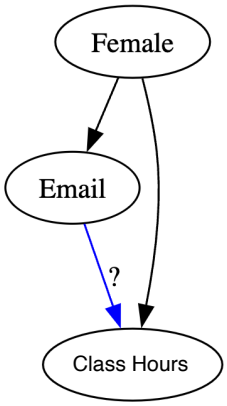
Effect of Email on Class Hours: Male 10hrs, Female 5hrs

```
Biased_Estimate = email_example.query("email==1")["hours"].mean() - \
    email_example.query("email==0")["hours"].mean()
print("Biased Estimate is", Biased_Estimate)
```

```
Biased Estimate is 13.0
```

```
# Calculate Average Treatment Effect (ATE)
## Effect on men is 10 hours and there are 6 men,
## Effect on women is 5 hours and there are 4 women.
ATE = (10*6 + 5*4)/10
print("ATE is", ATE)
```

```
ATE is 8.0
```



This ATE calculation is a "matching":

Male: Match each treated with a control, vice versa

Female: Match a treated with a control, vice versa

Take the average weighted by #obs of men and women, respectively

Diagnoses

How about OLS regressions? A good diagnositic tool

With and Without Controlling the Variables

```
# Run OLS regression of outcome on treatment without controlling gender
import statsmodels.formula.api as smf
ols_no_gender = smf.ols(formula='hours ~ email', data=email_example).fit()
print(ols_no_gender.summary().tables[1])

# Run OLS regression of outcome on treatment with controlling gender
import statsmodels.formula.api as smf
ols_with_gender = smf.ols(formula='hours ~ email + gender', data=email_example).fit()
print(ols_with_gender.summary().tables[1])
```

```
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
Intercept     73.0000      2.739     26.656      0.000      66.685      79.315
email         13.0000      3.873      3.357      0.010       4.069      21.931
==============================================================================
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
Intercept      82.6000      0.944     87.486      0.000      80.367      84.833
gender[T.M]   -12.0000      0.926    -12.961      0.000     -14.189      -9.811
email           8.2000      0.907      9.040      0.000       6.055      10.345
==============================================================================
```


Consider a more realistic (more complex) dataset

Email's effect on class hours

```
# Load the student email dataset
df = pd.read_csv('https://songyao21.github.io/course_data/students_email_hours.csv')
df.head()
```

	Student_ID	Email	Class_Hours	Female	Age	Income	GPA
0	1	1	77	1	27	81	3.55
1	2	0	106	0	27	100	3.30
2	3	1	63	0	24	110	4.00
3	4	1	59	1	32	75	3.34
4	5	1	50	1	30	116	4.00

Do We Have Random Assignment of Treatment (Email)?

Random assignment alleviates the confounding bias concern

If the email is randomly assigned, students' attributes/features should be similar between treatment and control groups

Variable	Treated Mean	Control Mean	Standardized Diff	p-value
Female	0.683	0.412	0.567	0.000
Age	29.228	32.457	-0.586	0.000
Income	92.234	91.533	0.053	0.019
GPA	3.764	3.659	0.304	0.000

What can we conclude from this table?

- NOTE: Creating the mean comparison table above is sometimes called *balance check* or *randomization check*

A naive (and most likely biased) estimate of ATE

Simple difference calculation of outcome between treated and control groups

```
biased_estimate = df.query("Email==1")["Class_Hours"].mean() - \
    df.query("Email==0")["Class_Hours"].mean()
print("ATE using unadjusted estimate is", biased_estimate.round(2))
```

```
ATE using unadjusted estimate is 10.05
```

The balance check shows there are potential confounding factors

- Treated and control groups differ significantly in their attributes/features

Use OLS to confirm the existence of bias

```
# Run OLS regressions with and without covariates
ols_model_no_covariates = smf.ols(formula='Class_Hours ~ Email', data=df).fit()
print(ols_model_no_covariates.summary().tables[1])

ols_model_with_covariates = smf.ols(formula='Class_Hours ~ Email + Female + Age + Income + GPA',
                                    data=df).fit()
print(ols_model_with_covariates.summary().tables[1])
```

```
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
Intercept     73.2226      0.531    137.879      0.000      72.182      74.264
Email         10.0510      0.621     16.196      0.000       8.835      11.267
==============================================================================
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
Intercept     44.7598      3.601     12.431      0.000      37.702      51.818
Email          8.9474      0.664     13.474      0.000       7.646      10.249
Female         2.3053      0.580      3.975      0.000       1.168       3.442
Age           -0.0052      0.049     -0.106      0.916      -0.101       0.090
Income         0.1746      0.021      8.370      0.000       0.134       0.215
GPA            3.1986      0.826      3.875      0.000       1.580       4.817
==============================================================================
```

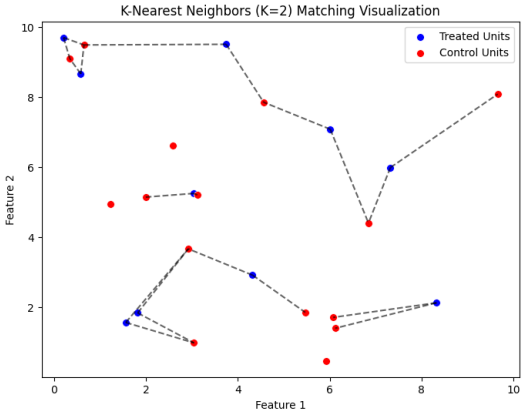

K-Nearest Neighbors

Most Basic Matching Method: K-Nearest Neighbors

The intuition: Find the nearest units in the other group (Euclidean distance)

In this example

- Each customer has two features
- Each treated is matched with two nearest neighbors in the control
  - Not showing control units matching with treated units (too cluttered)
  - With replacement. i.e., a control can match with multiple treated



K Nearest Neighbor Matching (KNN)

We use Python's "causalinference" package (manual implementation posted)

```
from causalinference import CausalModel

CM_matching = CausalModel(
    Y=df["Class_Hours"].values,
    D=df["Email"].values,
    X=df[["Female", "Age", "Income", "GPA"]].values,
)

CM_matching.est_via_matching(matches=5, bias_adj=True)

print(CM_matching.estimates)
```

```
Treatment Effect Estimates: Matching

                      Est.       S.e.          z      P>|z|      [95% Conf. int.]
--------------------------------------------------------------------------------
               ATE    10.248      0.963     10.638      0.000      8.360     12.136
               ATC     8.621      0.786     10.962      0.000      7.080     10.163
               ATT    10.843      1.153      9.407      0.000      8.583     13.102
```

Practical issues of KNN

- The matching may become low quality when number of features is large
- Even after matching, treated and control units may still differ substantially in some features (sometimes referred to as *poor balance* after matching)
- Propensity Score Matching (PSM) is an alternative to address these issues
  - Summarize many features into a single metric, the propensity score
  - We can check the balance of propensity score
    - Take actions if they are not balanced between treatment and control groups

Propensity Score Matching

Propensity Score Matching

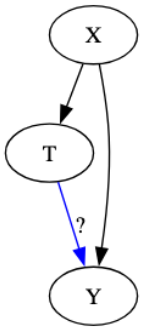
The intuition is similar to KNN

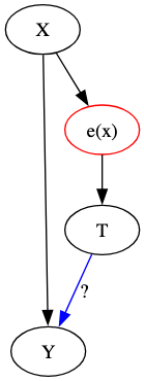
1. Use features to calibrate a propensity score for each customer (e.g., a logistic regression)
2. Use the propensity score to match
3. We can check the propensity score balance
   (1) Standardized Mean Difference (SMD) of Propensity Scores
   (2) Percentage outside common support region

Propensity Score Matching

The intuition is similar to KNN

1. Use features to calibrate a propensity score for each customer (e.g., a logistic regression)
2. Use the propensity score to match with nearest neighbors (i.e., only one feature, the propensity score, is used in matching)
3. But how is matching in one feature sufficient?





Propensity Score Matching: Step 1

Calibrate Propensity Score

```python
# Estimate propensity scores using logistic regression
import statsmodels.formula.api as smf

# Fit logistic regression to estimate propensity scores
ps_model = smf.logit(formula='Email ~ Female + Age + Income + GPA', data=df).fit()
print(ps_model.summary().tables[1])

# Add propensity scores to dataframe
df['propensity_score'] = ps_model.predict(df)
```

- We use logistic
- In practice, we may use more flexible methods (ML models)

Propensity Score Matching: Step 2

Matching with the Propensity Score (manual implementation code posted)

```python
cm = CausalModel(
    Y=df["Class_Hours"].values,
    D=df["Email"].values,
    X=df["propensity_score"].values
)

cm.est_via_matching(matches=5, bias_adj=True)

print(cm.estimates)
```

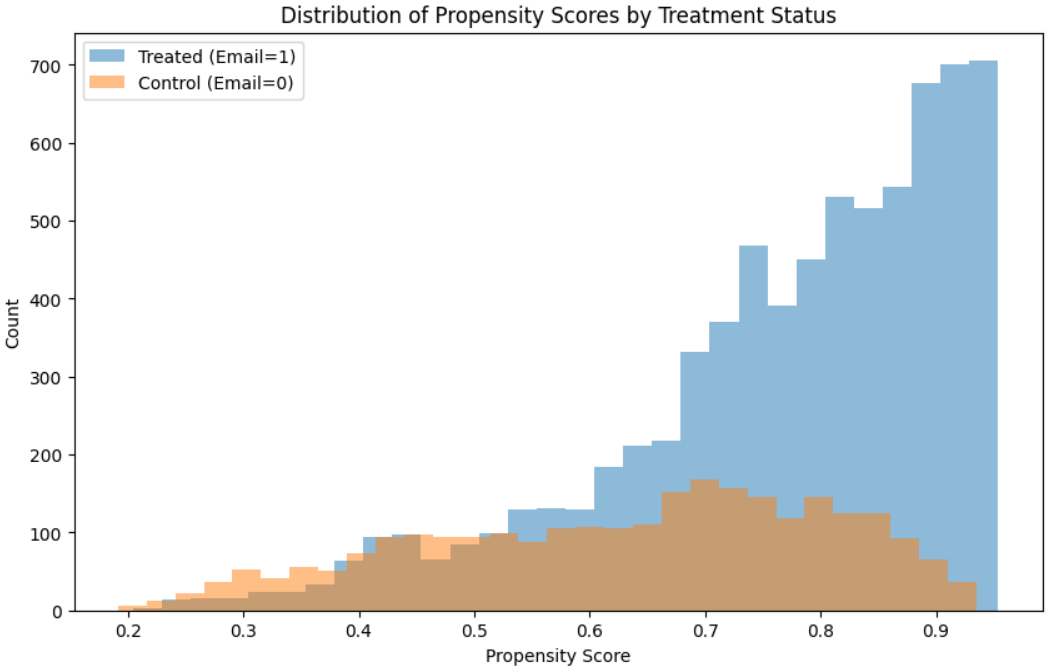
```
Treatment Effect Estimates: Matching

                Est.       S.e.          z      P>|z|      [95% Conf. int.]
-------------------------------------------------------------------------------
         ATE     9.600      1.300      7.385      0.000      7.052     12.147
         ATC     8.402      0.806     10.429      0.000      6.823      9.981
         ATT    10.037      1.654      6.068      0.000      6.795     13.280
```

- The ATE has become smaller (closer to the OLS result, 8.95).

Also need to check the propensity score's match quality

Do treatment and control groups have comparable propensity scores



Metric 1: Percentage of obs outside common support

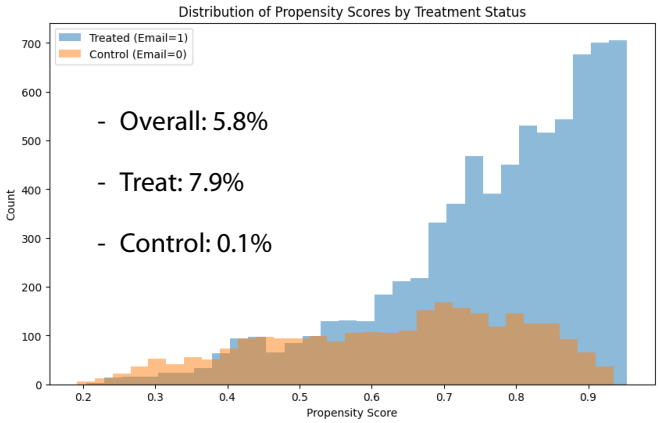
Do we have many observations that do not have neighbors nearby?
Rule of thumb: <5%

- Common support: The range of treatment/control's propensity score overlap
- Overall how many observations fall outside?
- How many observation fall outside by treatment status

```
Propensity Score Summary Statistics:

Treated Group:
count    7323.000000
mean        0.769346
std         0.149036
min         0.204110
25%         0.692859
50%         0.805577
75%         0.889994
max         0.953898
Name: propensity_score, dtype: float64

Control Group:
count    2677.000000
mean        0.630961
std         0.172188
min         0.191512
25%         0.502615
50%         0.658847
75%         0.770778
max         0.934831
Name: propensity_score, dtype: float64
```



Metric 2: Standardized Mean Difference (SMD)

Does the propensity score's distribution differ btw treatment and control groups?
Rule of thumb: <0.1 (some people use 0.25)

$$\text{SMD} = \frac{\bar{X}_T - \bar{X}_C}{\sqrt{\frac{\sigma_T^2 + \sigma_C^2}{2}}}$$

- Numerator: Difference in mean propensity score
- Denominator: Standard Deviation of that difference
- In our case, this metric is 0.859
- Note: We can do this check for each feature, not just PS. But when the number of features is large, this becomes cumbersome.

Inverse Propensity Weighting (IPW)

Solution 1: Inverse Propensity Weighting (IPW)

Intuition behind IPW

- Instead of matching treated and control units, IPW reweights the sample so that groups are comparable.
  - Individuals who are underrepresented in their group (e.g., low propensity to be treated but was treated) get higher weights.
  - Individuals who are overrepresented in their group (e.g., high propensity to be treated and was treated) get lower weights.
- For ATE, weights are the inverse of the propensity score:
  - Treated units

$$w_i = \frac{1}{e(X_i)}$$

  - Untreated (control) units

$$w_i = \frac{1}{1 - e(X_i)}$$

  - ATE: Treated's average outcome - Control's average outcome (weighted by the weights defined above)

Solution 1: Inverse Propensity Weighting (IPW)

Intuition behind IPW

- For ATT, weights are different from the ATE case:
  - Treated units

$$w_i = 1$$

  - Untreated (control) units

$$w_i = \frac{e(X_i)}{1 - e(X_i)}$$

- For ATC, weights are different from the ATE and ATT cases:
  - Treated units

$$w_i = \frac{1 - e(X_i)}{e(X_i)}$$

  - Untreated (control) units

$$w_i = 1$$

- ATT/ATC: Treated's average outcome - Control's average outcome (weighted by the weights defined above)

Solution 1: Inverse Propensity Weighting (IPW)

```
CM_ipw = CausalModel(
    Y=df["Class_Hours"].values,
    D=df["Email"].values,
    X=df[["Female", "Age", "Income", "GPA"]].values
)

# Estimate propensity scores using logistic regression
CM_ipw.est_propensity_s()

# Retrieve estimated propensity scores correctly
p_scores = CM_ipw.propensity['fitted']

# Clip extreme propensity scores
eps = 1e-2 # set the threshold to 0.01
# clip the propensity scores to be between 0.01 and 0.99
lower, upper = eps, 1-eps
clipped_ps = np.clip(p_scores, lower, upper)

# Manually replace the propensity scores in the internal dictionary
# CM_ipw.propensity['fitted'] = clipped_ps
CM_ipw.propensity._dict["pscore"] = clipped_ps

# Estimate treatment effects using IPW
CM_ipw.est_via_weighting()

# Report ATE
print("\nIPW Results from causalinference:")
print(CM_ipw.estimates)
```

Clipping to avoid extreme values

```
IPW Results from causalinference:

Treatment Effect Estimates: Weighting

                     Est.       S.e.          z      P>|z|      [95% Conf. int.]
--------------------------------------------------------------------------------
           ATE      9.475      0.767     12.359      0.000      7.972     10.977
```


The "causalinference" package does not report ATT/ATC

```
# Get data from the IPW model
Y = df["Class_Hours"].values
D = df["Email"].values
propensity = CM_ipw.propensity['fitted']  # Access fitted propensity scores

# Clip extreme propensity scores to avoid numerical issues
eps = 1e-2 # set the threshold to 0.01
# clip the propensity scores to be between 0.01 and 0.99
lower, upper = eps, 1-eps
propensity = np.clip(propensity, lower, upper)

# Get indices of treated and control units
D1_indices = D == 1
D0_indices = D == 0

# For ATT
Y1_treated = Y[D1_indices]  # Outcomes for treated group
ps_control = propensity[D0_indices]  # Propensity scores for control group
Y0_control = Y[D0_indices]  # Outcomes for control group

# Weight the control outcomes by ps/(1-ps)
weights_control_att = ps_control / (1 - ps_control)
weighted_control_outcomes = Y0_control * weights_control_att
att = np.mean(Y1_treated) - np.sum(weighted_control_outcomes) / np.sum(weights_control_att)

# For ATC
Y0_control = Y[D0_indices]  # Outcomes for control group
ps_treated = propensity[D1_indices]  # Propensity scores for treated group
Y1_treated = Y[D1_indices]  # Outcomes for treated group

# Weight the treated outcomes by (1-ps)/ps
weights_treated_atc = (1 - ps_treated) / ps_treated
weighted_treated_outcomes = Y1_treated * weights_treated_atc
atc = np.sum(weighted_treated_outcomes) / np.sum(weights_treated_atc) - np.mean(Y0_control)

print(f"ATT: {att:.4f}")
print(f"ATC: {atc:.4f}")
```

Adjusting the weights

```
ATT: 9.8855
ATC: 8.5353
```

Solution 2: Stratified Propensity Score Matching (SPSM)

- Divide customers into groups (10 or 5) based on their propensity scores
  - E.g., Group 1: *ps<10%*; Group 2: *10%<=ps<20%*; …
- Within each group, implement PSM
- Obtain the average treatment effect using average (weighted by group size)
- One key reason why this approach helps
  - Within each stratum, treated and control units are more balanced:
    - Each stratum contains a more homogeneous subset of the data.
    - Covariates within each stratum are expected to be more similar.

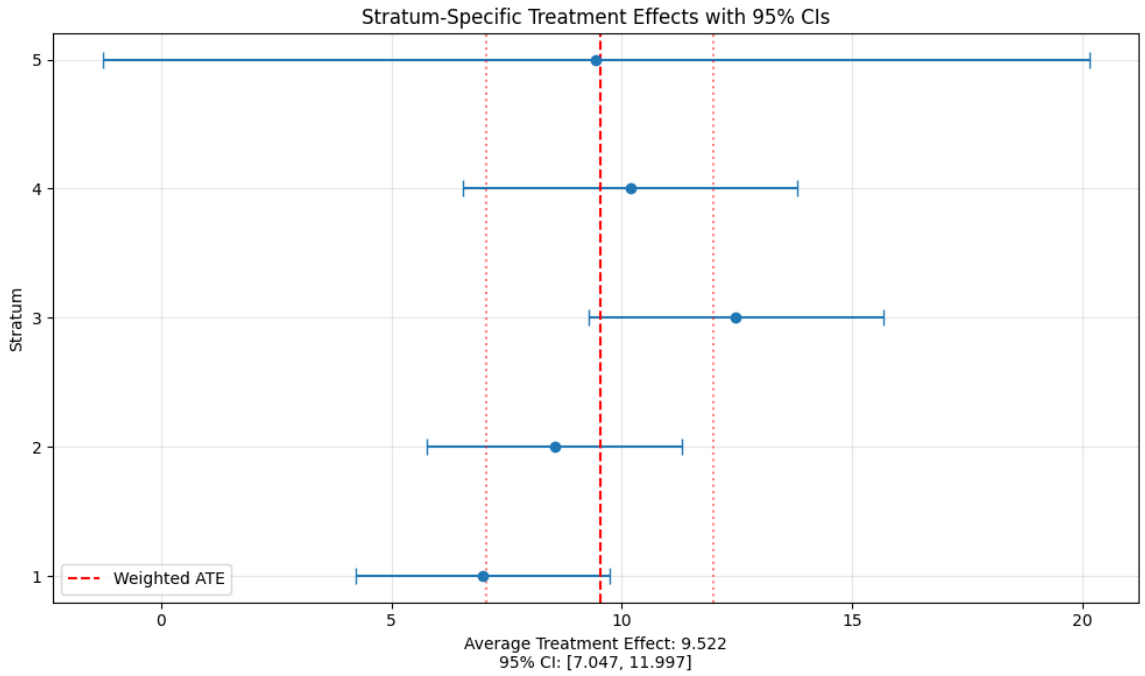
Solution 2: Stratified Propensity Score Matching (SPSM)

```
Weighted Average Treatment Effect: 9.522
Weighted Standard Error: 1.263
95% CI: [7.047, 11.997]
```

Stratum-Specific Results:

Stratum	ATE	Std Err	N	Treated	Control
1	6.98	1.41	2000	948	1052
2	8.54	1.41	2000	1317	683
3	12.48	1.63	2000	1513	487
4	10.18	1.85	2002	1651	351
5	9.44	5.46	1998	1894	104

Solution 2: Stratified Propensity Score Matching (SPSM)



Which one to use?

My rule of thumb is

- Start with an OLS or PSM to establish some benchmark
  - For PSM, remember to check % outside common support and SDM
- IPW is more robust and used widely
- SPSM
  - When we need to explicitly check balance
  - When there are many propensity scores near 0 or 1
    - IPW will have extreme weights in this case.