Economics 691-06: Assignment 1 Solutions

Nihar Shah

Solutions are marked in blue.

Problem 1: Treatment Effects

Consider the following table.

Name	Y^1	Y^0	δ
Unit 1	22	12	1
Unit 2	26	14	1
Unit 3	20	18	1
Unit 4	23	16	1
Unit 5	21	19	1
Unit 6	17	18	1
Unit 7	35	30	0
Unit 8	33	31	0
Unit 9	27	28	0
Unit 10	42	36	0

```
# Load libraries: numpy and pandas
import numpy as np
import pandas as pd

data = pd.DataFrame([22, 26, 20, 23, 21, 17, 35, 33, 27, 42],
                    columns = ['y1'])
data['y0'] = [12, 14, 18, 16, 19, 18, 30, 31, 28, 36]
data['treatment'] = [1, 1, 1, 1, 1, 1, 0, 0, 0, 0]
```

1. **(10 points)** Compute each of the ATE, ATT, and ATU.

```
i = np.where(data['treatment'] == 1)[0]
j = np.where(data['treatment'] == 0)[0]

print('The ATE is: ' +
      str(round(np.average(data['y1'] - data['y0']), 2)))
print('The ATT is: ' +
      str(round(np.average(data.iloc[i,]['y1'] - data.iloc[i,]['y0']), 2)))
print('The ATU is: ' +
      str(round(np.average(data.iloc[j,]['y1'] - data.iloc[j,]['y0']), 2)))
```

```
The ATE is: 4.4
The ATT is: 5.33
The ATU is: 3.0
```

2. **(5 points)** Showing all work, derive a single equation that links these three concepts to each other, observed data, and selection bias.

   We start with the definition of the ATE, and then break it into the sub-population that is treated (which represents π of the population) and the sub-population that is not treated (which represents $1-\pi$).

   $$\text{ATE} = \mathbb{E}(Y^1) - \mathbb{E}(Y^0)$$
   $$\text{ATE} = \pi\mathbb{E}(Y^1|\delta_i = 1) + (1-\pi)\mathbb{E}(Y^1|\delta_i = 0) - \pi\mathbb{E}(Y^0|\delta_i = 1) - (1-\pi)\mathbb{E}(Y^0|\delta_i = 0)$$
   $$\text{ATE} - (1-\pi)\mathbb{E}(Y^1|\delta_i = 0) + \pi\mathbb{E}(Y^0|\delta_i = 1) - \pi\mathbb{E}(Y^0|\delta_i = 0) = \pi\mathbb{E}(Y^1|\delta_i = 1) - \mathbb{E}(Y^0|\delta_i = 0)$$

   Now we add $(1-\pi)\mathbb{E}(Y^1|\delta_i = 1)$ to both sides of the equation. The right-hand side becomes the observed effect (denoted as OE).

   $$\text{ATE} - (1-\pi)\mathbb{E}(Y^1|\delta_i = 0) + \pi\mathbb{E}(Y^0|\delta_i = 1) - \pi\mathbb{E}(Y^0|\delta_i = 0) + (1-\pi)\mathbb{E}(Y^1|\delta_i = 1) = \text{OE}$$

   Now we add and subtract the selection effect (denoted as SE) to the left-hand side, which is $\mathbb{E}(Y^0|\delta_i = 1) - \mathbb{E}(Y^0|\delta_i = 0)$:

   $$\begin{aligned}&\text{ATE} + \text{SE} - (1-\pi)\mathbb{E}(Y^1|\delta_i = 0) - (1-\pi)\mathbb{E}(Y^0|\delta_i = 1) + (1-\pi)\mathbb{E}(Y^0|\delta_i = 0) + \\ &(1-\pi)\mathbb{E}(Y^1|\delta_i = 1) = \text{OE}\end{aligned}$$

   Finally, we identify the ATT and ATU as $\mathbb{E}(Y^1|\delta_i = 1) - \mathbb{E}(Y^0|\delta_i = 1)$ and $\mathbb{E}(Y^1|\delta_i = 0) - \mathbb{E}(Y^0|\delta_i = 0)$ respectively to get the end result:

   $$\text{ATE} + \text{SE} + (1-\pi)(\text{ATT} - \text{ATU}) = \text{OE}$$

3. **(10 points)** In practice, which part of this equation is observable; and what value do you get? Which part of this equation is likely of interest; and what value do you get?

   In practice, we observed the "observed effect" (i.e. the average of the treated units' treated outcomes, minus the average of the untreated units' untreated outcomes). However, we are typically interested in the value of the average treatment effect, which is the effect of exposing the entire population to treatment. (In some cases, we may be more interested in the average treatment effect on the treated, i.e. the true treatment effect for those who opt into treatment; or in very rare cases, the average treatment effect on the untreated.) However, we observe none of these other terms. The values are computed below.

```
i = np.where(data['treatment'] == 1)[0]
j = np.where(data['treatment'] == 0)[0]

print('The observed effect is: ' +
      str(round(np.average(data.iloc[i,]['y1']) -
                np.average(data.iloc[j,]['y0']), 2)))
```

```
print('The ATE is: ' +
      str(round(np.average(data['y1'] - data['y0']), 2)))

The observed effect is: -9.75
The ATE is: 4.4
```

4. **(10 points)** Assuming that your empirical estimates are not due to sampling noise, which assumptions in the classic experimentation model are likely violated in this table? Does this explain the discrepancy in Question 3, and how does each assumption failure increase or decrease the discrepancy?

   There are five assumptions that underpin the classic experimentation model. For three of them – compliance (did all units in the treatment group take treatment?), SUTVA (do units' treatment statuses affect one another?), and observable responses (can we observe the response variable?) – we have no reason to believe there are any violations. However, we do have reasons to believe that the last two– heterogeneity and selection – have been violated. As a result, our ATE is substantially higher than the observed effect.

   (a) Heterogeneous Treatment Effect Bias: We know this is violated from the potential outcomes framewhere, where we see a clear positive difference between the ATT and the ATU. Interestingly, a positive bias (on its own) causes the observed effect to overstate the ATE.

   (b) Selection Bias: We know this is violated from a visual inspect of the data alone, and can compute it (below) to demonstrate its magnitude. Units who opt into treatment have far lower untreated outcomes than units who stay in the control group. This bias is so large that it causes the observed effect to massively understate the ATE.

```
i = np.where(data['treatment'] == 1)[0]
j = np.where(data['treatment'] == 0)[0]

print('The selection effect is: ' +
      str(round(np.average(data.iloc[i,]['y0']) -
                np.average(data.iloc[j,]['y0']), 2)))

The selection effect is: -15.08
```

Problem 2: Analysis on Classic Experiments

Suppose you are a data scientist at TikTok, and you are considering deploying a feature – richer color palettes – to increase time spent on the app. You run a small and short experiment to test its impact, with an equal treatment and control group.

The data following the experiment is in the file data_assignment1.csv. The file has four columns and 1200 rows. Each row corresponds to a separate user, for which there are four columns: time spent on Tiktok (in seconds) during the experiment, an indicator for whether the user is in the treatment group (where 1 indicates treatment), the age of the user, and the number of channels (i.e. other accounts) to which the user previously subscribed.

```
# Load libraries: numpy, pandas, and linear regression
import numpy as np
import pandas as pd
import statsmodels.formula.api as sm

# Load in data
from google.colab import files
import io
uploaded = files.upload()
data = pd.read_csv(io.BytesIO(uploaded['data_assignment1.csv']))

# Define a custom-built function to do regression and Bootstrap
# This will make future analysis far easier
# The user enters in the linear regression formula, and a flag indicating
# whether they want the parameter estimate (0), the analytic standard error (1),
# the bootstrapped standard error (2), or the raw bootstrap samples (3).
# They also enter an optional parameter called "fancy" for formatting purposes

def regression(formula, flag = 0, fancy = True):

  # Set some fixed seed
  np.random.seed(0)

  # Run the regression model
  result = sm.ols(formula=formula, data = data).fit()

  # Extract either coefficient or analytic standard errors if desired
  if flag == 0:
    output = pd.DataFrame({'cols': result.params.index, 'par': result.params})
    effect = output.iloc[np.where(output['cols'] == 'treatment')[0][0],1]
    if fancy:
      return('The parameter estimate is ' + str(round(effect, 2)))
    return(effect)

  if flag == 1:
    output = pd.DataFrame({'cols': result.bse.index, 'par': result.bse})
    se = output.iloc[np.where(output['cols'] == 'treatment')[0][0],1]
    if fancy:
      return('The analytic SE is ' + str(round(se, 2)))
    return(se)

  # Continue onto bootstrap

  # First identify the treated and control units
  samples = []
  i = np.where(data['treatment'] == 1)[0]
  j = np.where(data['treatment'] == 0)[0]
```

```
# Run 2000 iterations of the bootstrap
for counter in range(2000):

  # Implement re-sampling; notice we respect 600 treated and 600 control units
  # at all times, to mimic the original experiment!
  i_sample = np.random.choice(i, len(i), replace = True)
  j_sample = np.random.choice(j, len(j), replace = True)
  index = np.append(i_sample, j_sample)

  # Run the regression model with the resampled data
  result = sm.ols(formula=formula, data = data.iloc[index,]).fit()

  # Extract coefficient
  output = pd.DataFrame({'cols': result.params.index, 'par': result.params})
  sample_effect = output.iloc[np.where(output['cols'] == 'treatment')[0][0],1]

  # Save coefficient
  samples.append(sample_effect)

# Either return the standard deviation of the samples, or the actual samples
if flag == 2:
  if fancy:
    return('The bootstrapped SE is ' + str(round(np.std(samples), 2)))
  return(np.std(samples))
else:
  return(samples)
```

1. **(10 points)** Without using regression techniques, what is your assessment of the experiment — including both its estimated impact and the significance of that estimate? To assess significance, use both analytic techniques and the bootstrap. Be very clear about precisely how you performed the bootstrap, and why you chose to do it that way.

```
# Compute difference in means
effect = (np.average(data.loc[data['treatment'] == 1,'timespent']) -
          np.average(data.loc[data['treatment'] == 0,'timespent']))

# Compute analytic standard deviation as sqrt(s_t^2/n_t + s_c^2/n_c)
v_t = (pow(np.std(data.loc[data['treatment'] == 1,'timespent']), 2)/
      sum(data['treatment'] == 1))
v_c = (pow(np.std(data.loc[data['treatment'] == 0,'timespent']), 2)/
      sum(data['treatment'] == 0))
s = pow(v_t + v_c, 0.5)

# Compute bootstrapped estimates

# First identify the treated and control units
samples = []
i = np.where(data['treatment'] == 1)[0]
```

```
j = np.where(data['treatment'] == 0)[0]

# Set some fixed seed
np.random.seed(0)

# Run 2000 iterations of the bootstrap
for counter in range(2000):

  # Implement re-sampling; notice we respect 600 treated and 600 control units
  # at all times, to mimic the original experiment!
  i_sample = np.random.choice(i, len(i), replace = True)
  j_sample = np.random.choice(j, len(j), replace = True)

  # Create resampled data
  resample = data.iloc[np.append(i_sample, j_sample),]

  # Compute difference in means in resampled data
  samples.append(
      np.average(resample.loc[resample['treatment'] == 1,'timespent']) -
      np.average(resample.loc[resample['treatment'] == 0,'timespent']))

print('The parameter estimate is ' + str(round(effect, 2)))
print('The analytic SE is ' + str(round(s, 2)))
print('The bootstrapped SE is ' + str(round(np.std(samples), 2)))
```

```
The parameter estimate is 4.24
The analytic SE is 3.07
The bootstrapped SE is 3.15
```

We can see that the t-statistic will fall well short of the needed 1.96 cutoff (under either set of standard errors), and so we cannot conclude that the treatment works from this first analysis.

2. **(10 points)** Using regression techniques but without using the additional covariates, what is your assessment? Again, use both analytic techniques and the bootstrap to assess significance; and be clear about the bootstrap approach.

```
print(regression("timespent ~ treatment", 0))
print(regression("timespent ~ treatment", 1))
print(regression("timespent ~ treatment", 2))
```

```
The parameter estimate is 4.24
The analytic standard error is: 3.07
The bootstrapped standard error is: 3.15
```

Unsurprisingly, we get the same results, illustrating that simple regression and t-tests are identical. Note that the bootstrap respects the fact that the experiment aims to have exactly equal treatment and control groups, and so each group should have exactly six hundred units in every re-sampling iteration.

3. **(5 points)** Now incorporate the additional covariates in your regression model. What is your assessment of the experiment?

```
print(regression("timespent ~ treatment + age + channels", 0))
print(regression("timespent ~ treatment + age + channels", 1))
print(regression("timespent ~ treatment + age + channels", 2))

The parameter estimate is 3.05
The analytic SE is 0.74
The bootstrapped SE is 0.74
```

These additional covariates have greatly reduced the noise in the experiment, and now our t-statistics are well in excess of the 1.96 cutoff. We now can conclude the the treatment works in this refined analysis, and that its effect is to cause users to spend an extra three seconds on the app.

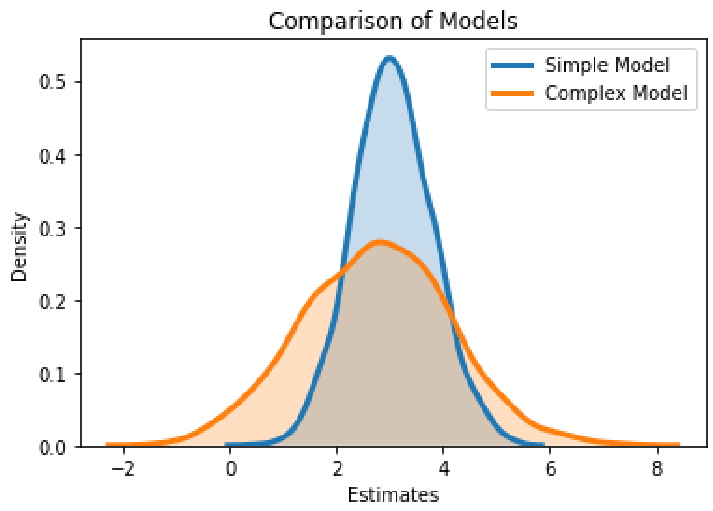
4. **(10 points)** Another data scientist at TikTok tells you that your model in Question 3 is too simple. She tells you that, rather than using the covariates as simple linear predictors, you should include some non-linear transformations of those covariates. Specifically, she recommends that you use the square of channels, the square of age, and the interaction between channels and treatment as additional covariates. You're less sure, as this model is substantially more complex and may have higher variance. Using the bootstrap only, compare the variance of the complex model versus your simple model.

```
data['channels2'] = pow(data['channels'], 2)
data['age2'] = pow(data['age'], 2)
print(regression("timespent ~ treatment + age + channels", 2))
print(regression("timespent ~ treatment + age + channels + age2" +
                 "+ channels2 + channels * treatment", 2))

The bootstrapped SE is 0.74
The bootstrapped SE is 1.38
```

Our suspicions are right: the complex model has much higher variance.

5. **(5 points)** Using the same output as in Question 4, depict the comparison of variances graphically.

```
import matplotlib.pyplot as plt
import seaborn as sns

sns.distplot(simple, hist = False, kde = True,
                 kde_kws = {'shade': True, 'linewidth': 3},
                 label = 'Simple Model')
sns.distplot(complex, hist = False, kde = True,
                 kde_kws = {'shade': True, 'linewidth': 3},
                 label = 'Complex Model')
plt.title('Comparison of Models')
plt.xlabel('Estimates')
plt.ylabel('Density')
```



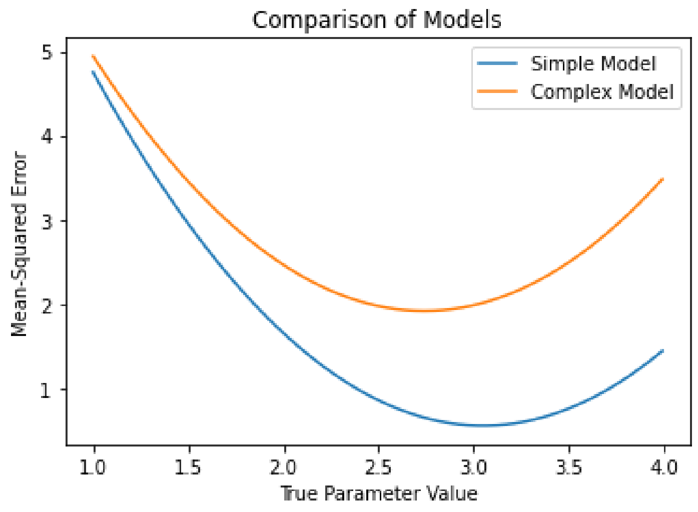
6. **(10 points)** The data scientist tells you that variance is only part of the story; it is also important to find a model with lower bias. In general, why can we not assess model bias using data only? Despite that, can we still holistically assess the simple versus complex model using data only?

   In general, to compute bias, we need to know the true treatment effect. This is impossible in a sample; after all, if we knew the treatment effect, we would not need to develop estimators!

   That being said, we actually can make progress by iterating through potential values for the true treatment effect. We can then compute the mean-squared error under each potential value, as we sum up variance (computed in the Question 3) and the square of the bias between the hypothetical true effect and our sample effect. We then depict the mean-squared error, and notice that for any plausible treatment effect, the simple model always performs better! The variance is so much higher with the complex model that no amount of realistic bias reduction can be worth it.

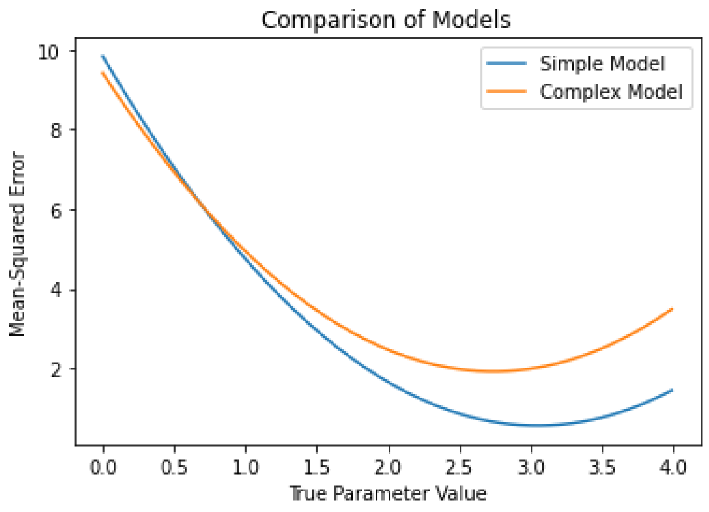
```
simple_e = regression("timespent ~ treatment + age + channels", 0, False)
simple_se = regression("timespent ~ treatment + age + channels", 2, False)
complex_e = regression("timespent ~ treatment + age + channels + age2" +
                       "+ channels2 + channels * treatment", 0, False)
complex_se = regression("timespent ~ treatment + age + channels + age2" +
                        "+ channels2 + channels * treatment", 2, False)

def mse(bottom):

  values = []
  simple_output = []
  complex_output = []

  # Iterate through possible parameter values
  for v in range(bottom * 100, 400):
    values.append(v/100)
    simple_output.append(pow(v/100 - simple_e, 2) + pow(simple_se, 2))
```

```
        complex_output.append(pow(v/100 - complex_e, 2) + pow(complex_se, 2))

    sns.lineplot(values, simple_output, label = 'Simple Model')
    sns.lineplot(values, complex_output, label = 'Complex Model')
    plt.title('Comparison of Models')
    plt.xlabel('True Parameter Value')
    plt.ylabel('Mean-Squared Error')

mse(1)
```



I saw *plausible* and *realistic* because as you start to go out into the extreme tails of potential values for the treatment effect, the complex model technically starts to do better. However, it seems highly unlikely that the treatment effects are so far into the tails, given our sample estimates from both models.

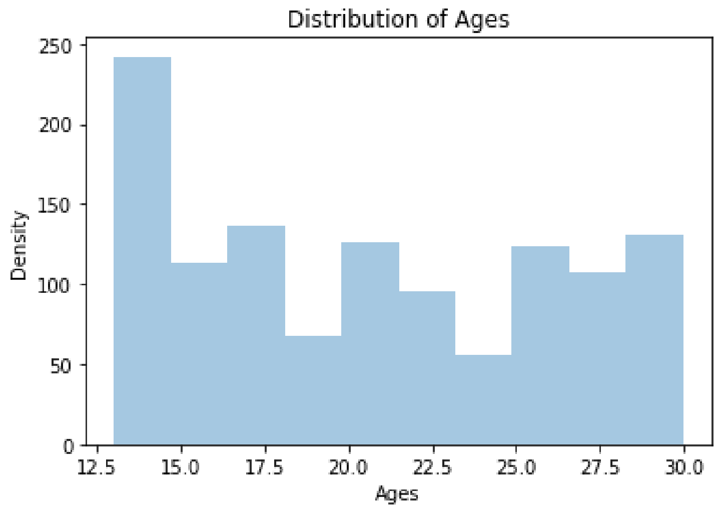
```
mse(0)
```



7. **(5 points)** Suppose we return to the simple model. After examining the data for interesting or suspicious patterns, can you identify and justify any potential improvements to this model?

   There are many answers that could receive credit here. However, I did embed one particular flaw in the data. You may notice that the age distribution is highly skewed: it is largely uniform, except with a large spike at the minimum age (13). This is not uncommon with tech products: younger users often avoid age restrictions by declaring themselves to be at the minimum age. As such, one reasonable approach would be to simply drop users who identified as thirteen, as some of them are actually younger users and so their covariates are incorrectly measured.

```
sns.distplot(data['age'], hist = True, kde = False)
plt.title('Distribution of Ages')
plt.xlabel('Ages')
plt.ylabel('Density')
```



```
data = data.drop(np.where(data['age'] == 13)[0])
print(regression("timespent ~ treatment + age + channels", 0))
print(regression("timespent ~ treatment + age + channels", 2))
```

```
The parameter estimate is 2.49
The bootstrapped SE is 0.77
```

   You'll notice that we deleted nearly 200 rows, and still have approximately the same sized-standard errors.[1] This suggests that this approach is valuable in estimating the true treatment effect, perhaps in conjunction with the full sample.

[1]Note that we have to be a bit careful with the bootstrap here, as it is unclear how the experimenter would have handled skipping users who identify as age thirteen: would she identify it *ex ante* and thus hold fixed treatment and control groups, or would she identify it *ex post* and thus introduce randomness into the size of the groups?