1 Lalonde NSW Data

A. Load the Lalonde experimental dataset with the lalonde_data method from the module causalinference.utils. Using CausalModel from the module causalinference, provide summary statistics for the outcome variable and the covariates. Which covariate has the largest normalized difference?

In [29]:

```
from causalinference.utils import lalonde_data
```

In [30]:

```
# the dataset
data = lalonde_data()
```

In [31]:

```
from causalinference import CausalModel
```

In [32]:

```
Y = data[0]  # observed outcomes
D = data[1]  # treatment status indicators
X = data[2]  # the matrix of covariates
```

In [33]:

```
# CausalModel object
causal = CausalModel(Y, D, X)
```

In [34]:

```
# summary statistics for the outcome variable and the covariates
summary_stats = causal.summary_stats
print(summary_stats)
```

```
Summary Statistics

                       Controls (N_c=260)         Treated (N_t=185)
       Variable         Mean         S.d.         Mean         S.d.     Raw-diff
--------------------------------------------------------------------------------
              Y        4.555        5.484        6.349        7.867        1.794

                       Controls (N_c=260)         Treated (N_t=185)
       Variable         Mean         S.d.         Mean         S.d.     Nor-diff
--------------------------------------------------------------------------------
             X0        0.827        0.379        0.843        0.365        0.044
             X1        0.108        0.311        0.059        0.237       -0.175
             X2       25.054        7.058       25.816        7.155        0.107
             X3        0.154        0.361        0.189        0.393        0.094
             X4        0.835        0.372        0.708        0.456       -0.304
             X5       10.088        1.614       10.346        2.011        0.141
             X6        2.107        5.688        2.096        4.887       -0.002
             X7        0.750        0.434        0.708        0.456       -0.094
             X8        1.267        3.103        1.532        3.219        0.084
             X9        0.685        0.466        0.600        0.491       -0.177
```

In [35]:

```
# the summary_stats is a python dictionary, its keys are
summary_stats.keys()
```

Out[35]:

```
dict_keys(['N', 'K', 'N_c', 'N_t', 'Y_c_mean', 'Y_t_mean', 'Y_c_sd', 'Y_t_sd', 'rdiff', '
```

```
X_c_mean', 'X_t_mean', 'X_c_sd', 'X_t_sd', 'ndiff'])
```

In [36]:

```
# use numpy to locate the index of an element in an ndarray
import numpy as np
```

In [37]:

```
# the ndiff represents the normalized difference
normalized_diff = summary_stats['ndiff']

# the covariate with the largest normalized difference
max_ndiff = np.where(normalized_diff == [max(normalized_diff)])[0][0]

print(f"The covariate with the largest normalized difference is X{max_ndiff}")
```

```
The covariate with the largest normalized difference is X5
```

B. Estimate the propensity score using the selection algorithm est_propensity_s. In selecting the basic covariates set, specify E74, U74, E75, and U75. What are the additional linear terms and second-order terms that were selected by the algorithm?

In [38]:

```
# estimate the propensity score
# using 6,7,8,9 as the column numbers representing E74, U74, E75, U75
causal.est_propensity_s(lin_B=[6, 7, 8, 9])
```

In [39]:

```
# propensity score
propensity = causal.propensity
print(propensity)
```

```
Estimated Parameters of Propensity Score

                    Coef.       S.e.          z      P>|z|      [95% Conf. int.]
--------------------------------------------------------------------------------
     Intercept     -3.480      4.471     -0.778      0.436    -12.243      5.283
            X6      0.034      0.051      0.667      0.505     -0.066      0.133
            X7     -0.236      0.386     -0.611      0.541     -0.992      0.521
            X8      0.058      0.051      1.144      0.253     -0.041      0.158
            X9     -3.477      1.652     -2.104      0.035     -6.716     -0.238
            X4      7.329      4.255      1.723      0.085     -1.010     15.668
            X1     -0.653      0.385     -1.696      0.090     -1.409      0.102
            X5      0.290      0.370      0.783      0.433     -0.435      1.015
         X4*X5     -0.668      0.349     -1.915      0.056     -1.352      0.016
         X6*X4     -0.130      0.057     -2.286      0.022     -0.241     -0.018
         X9*X5      0.304      0.156      1.950      0.051     -0.002      0.609
```

In [40]:

```
# additional linear terms and second-order terms selected by the algorithm
print("Linear Terms:\n", propensity['lin']) # 4, 1, and 5
print("Second-order terms:\n", propensity['se'])
```

```
Linear Terms:
 [6, 7, 8, 9, 4, 1, 5]
Second-order terms:
 [4.47080778 0.05070979 0.38578491 0.05084816 1.65246964 4.2545994
 0.38537311 0.37005553 0.34887244 0.05672495 0.15585817]
```

C. Trim the sample using trim_s to get rid of observations with extreme propensity score values. What is the cut-off that is selected? How many observations are dropped as a result?

In [41]:

```
# trim the sample
```

```
# trim the sample
causal.trim_s()
```

In [42]:

```
# get the cutoff
causal.cutoff
```

Out[42]:

```
0.13104228016193686
```

In [43]:

```
#observe the data
print(causal.summary_stats)
```

```
Summary Statistics

                       Controls (N_c=256)         Treated (N_t=182)
       Variable         Mean         S.d.         Mean         S.d.     Raw-diff
--------------------------------------------------------------------------------
              Y        4.543        5.501        6.237        7.587        1.694

                       Controls (N_c=256)         Treated (N_t=182)
       Variable         Mean         S.d.         Mean         S.d.     Nor-diff
--------------------------------------------------------------------------------
             X0        0.828        0.378        0.841        0.367        0.034
             X1        0.109        0.313        0.060        0.239       -0.176
             X2       25.074        7.091       25.841        7.208        0.107
             X3        0.156        0.364        0.187        0.391        0.081
             X4        0.832        0.375        0.714        0.453       -0.283
             X5       10.105        1.609       10.297        1.964        0.107
             X6        1.675        4.435        1.795        3.876        0.029
             X7        0.762        0.427        0.714        0.453       -0.108
             X8        1.213        3.052        1.457        3.132        0.079
             X9        0.691        0.463        0.604        0.490       -0.182
```

From the summary above, N_c = 256 means that 4 observations are dropped as a result of the trim

D. Stratify the sample using stratify_s. How many propensity bins are created? Report the summary statistics for each bin.

In [44]:

```
# stratify the sample
causal.stratify_s()
```

In [45]:

```
# Stratification summary will give the number of bins created
print(causal.strata)
```

```
Stratification Summary

              Propensity Score         Sample Size     Ave. Propensity   Outcome
   Stratum      Min.      Max.  Controls   Treated  Controls   Treated  Raw-diff
--------------------------------------------------------------------------------
         1     0.131     0.379       153        67     0.327     0.332     0.788
         2     0.380     0.483        69        63     0.435     0.443     1.587
         3     0.487     0.852        34        52     0.596     0.619     3.044
```

The above summary indicates that 3 bins are created by the starify_s method.

E. Estimate the average treatment effect using OLS, blocking, and matching. For matching, set the number of matches to 2 and adjust for bias. How much do the estimates differ?

In [47]:

```
# Estimate average treatment by OLS
causal.est_via_ols()
print(causal.estimates)
```

```
Treatment Effect Estimates: OLS

                     Est.       S.e.          z      P>|z|      [95% Conf. int.]
--------------------------------------------------------------------------------
           ATE      1.467      0.638      2.299      0.022      0.216      2.718
           ATC      1.385      0.652      2.123      0.034      0.106      2.663
           ATT      1.583      0.651      2.432      0.015      0.307      2.858
```

In [48]:

```
# Estimate average treatment by blocking
causal.est_via_blocking()
print(causal.estimates)
```

```
Treatment Effect Estimates: OLS

                     Est.       S.e.          z      P>|z|      [95% Conf. int.]
--------------------------------------------------------------------------------
           ATE      1.467      0.638      2.299      0.022      0.216      2.718
           ATC      1.385      0.652      2.123      0.034      0.106      2.663
           ATT      1.583      0.651      2.432      0.015      0.307      2.858

Treatment Effect Estimates: Blocking

                     Est.       S.e.          z      P>|z|      [95% Conf. int.]
--------------------------------------------------------------------------------
           ATE      1.542      0.641      2.406      0.016      0.286      2.798
           ATC      1.402      0.654      2.145      0.032      0.121      2.683
           ATT      1.739      0.663      2.623      0.009      0.440      3.039
```

In [50]:

```
# Estimate average treatment by matching
causal.est_via_matching(matches=2, bias_adj=True)
print(causal.estimates)
```

```
Treatment Effect Estimates: OLS

                     Est.       S.e.          z      P>|z|      [95% Conf. int.]
--------------------------------------------------------------------------------
           ATE      1.467      0.638      2.299      0.022      0.216      2.718
           ATC      1.385      0.652      2.123      0.034      0.106      2.663
           ATT      1.583      0.651      2.432      0.015      0.307      2.858

Treatment Effect Estimates: Blocking

                     Est.       S.e.          z      P>|z|      [95% Conf. int.]
--------------------------------------------------------------------------------
           ATE      1.542      0.641      2.406      0.016      0.286      2.798
           ATC      1.402      0.654      2.145      0.032      0.121      2.683
           ATT      1.739      0.663      2.623      0.009      0.440      3.039

Treatment Effect Estimates: Matching

                     Est.       S.e.          z      P>|z|      [95% Conf. int.]
--------------------------------------------------------------------------------
           ATE      1.400      0.888      1.576      0.115     -0.341      3.140
           ATC      1.316      0.971      1.356      0.175     -0.587      3.219
           ATT      1.517      0.935      1.623      0.105     -0.315      3.350
```

Estimates by matching are the least followed by estimates by OLS then Blocking produces the largest estimates.

2 Document Classification

A. From the module sklearn.datasets, load the training data set using the method fetch_20newsgroups. This dataset comprises around 18000 newsgroups posts on 20 topics. Print out a couple sample posts and list out all the topic names.

In [4]:

```python
from sklearn.datasets import fetch_20newsgroups
```

In [5]:

```python
dataset = fetch_20newsgroups(subset='train')
```

In [6]:

```python
data = dataset.data
```

In [7]:

```python
from pprint import pprint
```

In [8]:

```python
# print a couple (say 5) sample posts
for post in data[:5]:
    pprint(post)
    print()
```

```
("From: lerxst@wam.umd.edu (where's my thing)\n"
 'Subject: WHAT car is this!?\n'
 'Nntp-Posting-Host: rac3.wam.umd.edu\n'
 'Organization: University of Maryland, College Park\n'
 'Lines: 15\n'
 '\n'
 ' I was wondering if anyone out there could enlighten me on this car I saw\n'
 'the other day. It was a 2-door sports car, looked to be from the late 60s/\n'
 'early 70s. It was called a Bricklin. The doors were really small. In '
 'addition,\n'
 'the front bumper was separate from the rest of the body. This is \n'
 'all I know. If anyone can tellme a model name, engine specs, years\n'
 'of production, where this car is made, history, or whatever info you\n'
 'have on this funky looking car, please e-mail.\n'
 '\n'
 'Thanks,\n'
 '- IL\n'
 '   ---- brought to you by your neighborhood Lerxst ----\n'
 '\n'
 '\n'
 '\n'
 '\n')

('From: guykuo@carson.u.washington.edu (Guy Kuo)\n'
 'Subject: SI Clock Poll - Final Call\n'
 'Summary: Final call for SI clock reports\n'
 'Keywords: SI,acceleration,clock,upgrade\n'
 'Article-I.D.: shelley.1qvfo9INNc3s\n'
 'Organization: University of Washington\n'
 'Lines: 11\n'
 'NNTP-Posting-Host: carson.u.washington.edu\n'
 '\n'
 'A fair number of brave souls who upgraded their SI clock oscillator have\n'
 'shared their experiences for this poll. Please send a brief message '
 'detailing\n'
 'your experiences with the procedure. Top speed attained, CPU rated speed,\n'
 'add on cards and adapters, heat sinks, hour of usage per day, floppy disk\n'
 'functionality with 800 and 1.4 m floppies are especially requested.\n'
 '\n'
 'I will be summarizing in the next two days, so please add to the network\n'
```

```
'I will be summarizing in the next two days, so please add to the network\n'
"knowledge base if you have done the clock upgrade and haven't answered this\n"
'poll. Thanks.\n'
'\n'
'Guy Kuo <guykuo@u.washington.edu>\n')

('From: twillis@ec.ecn.purdue.edu (Thomas E Willis)\n'
'Subject: PB questions...\n'
'Organization: Purdue University Engineering Computer Network\n'
'Distribution: usa\n'
'Lines: 36\n'
'\n'
'well folks, my mac plus finally gave up the ghost this weekend after\n'
"starting life as a 512k way back in 1985.  sooo, i'm in the market for a\n"
'new machine a bit sooner than i intended to be...\n'
'\n'
"i'm looking into picking up a powerbook 160 or maybe 180 and have a bunch\n"
'of questions that (hopefully) somebody can answer:\n'
'\n'
'* does anybody know any dirt on when the next round of powerbook\n'
"introductions are expected?  i'd heard the 185c was supposed to make an\n"
'appearence "this summer" but haven\'t heard anymore on it - and since i\n'
"don't have access to macleak, i was wondering if anybody out there had\n"
'more info...\n'
'\n'
'* has anybody heard rumors about price drops to the powerbook line like the\n'
"ones the duo's just went through recently?\n"
'\n'
"* what's the impression of the display on the 180?  i could probably swing\n"
"a 180 if i got the 80Mb disk rather than the 120, but i don't really have\n"
'a feel for how much "better" the display is (yea, it looks great in the\n'
'store, but is that all "wow" or is it really that good?).  could i solicit\n'
'some opinions of people who use the 160 and 180 day-to-day on if its worth\n'
'taking the disk size and money hit to get the active display?  (i realize\n'
"this is a real subjective question, but i've only played around with the\n"
'machines in a computer store breifly and figured the opinions of somebody\n'
'who actually uses the machine daily might prove helpful).\n'
'\n'
'* how well does hellcats perform?  ;)\n'
'\n'
"thanks a bunch in advance for any info - if you could email, i'll post a\n"
'summary (news reading time is at a premium with finals just around the\n'
'corner... :( )\n'
'--\n'
'Tom Willis  \\  twillis@ecn.purdue.edu    \\    Purdue Electrical '
'Engineering\n'
'---------------------------------------------------------------------------\n'
'"Convictions are more dangerous enemies of truth than lies."  - F. W.\n'
'Nietzsche\n')

('From: jgreen@amber (Joe Green)\n'
'Subject: Re: Weitek P9000 ?\n'
'Organization: Harris Computer Systems Division\n'
'Lines: 14\n'
'Distribution: world\n'
'NNTP-Posting-Host: amber.ssd.csd.harris.com\n'
'X-Newsreader: TIN [version 1.1 PL9]\n'
'\n'
'Robert J.C. Kyanko (rob@rjck.UUCP) wrote:\n'
'> abraxis@iastate.edu writes in article '
'<abraxis.734340159@class1.iastate.edu>:\n'
'> > Anyone know about the Weitek P9000 graphics chip?\n'
"> As far as the low-level stuff goes, it looks pretty nice.  It's got this\n"
'> quadrilateral fill command that requires just the four points.\n'
'\n'
"Do you have Weitek's address/phone number?  I'd like to get some "
'information\n'
'about this chip.\n'
'\n'
'--\n'
'Joe Green\t\t\t\tHarris Corporation\n'
'jgreen@csd.harris.com\t\t\tComputer Systems Division\n'
'"The only thing that really scares me is a person with no sense of humor."\n'
```

```
 '"The only thing that really scares me is a person with no sense of humor."\n'
 '\t\t\t\t\t\t-- Jonathan Winters\n')

('From: jcm@head-cfa.harvard.edu (Jonathan McDowell)\n'
 'Subject: Re: Shuttle Launch Question\n'
 'Organization: Smithsonian Astrophysical Observatory, Cambridge, MA,  USA\n'
 'Distribution: sci\n'
 'Lines: 23\n'
 '\n'
 'From article <C5owCB.n3p@world.std.com>, by tombaker@world.std.com (Tom A '
 'Baker):\n'
 '>>In article <C5JLwx.4H9.1@cs.cmu.edu>, ETRAT@ttacs1.ttu.edu (Pack Rat) '
 'writes...\n'
 '>>>"Clear caution & warning memory.  Verify no unexpected\n'
 '>>>errors. ...".  I am wondering what an "expected error" might\n'
 '>>>be.  Sorry if this is a really dumb question, but\n'
 '> \n'
 '> Parity errors in memory or previously known conditions that were '
 'waivered.\n'
 '>    "Yes that is an error, but we already knew about it"\n'
 "> I'd be curious as to what the real meaning of the quote is.\n"
 '> \n'
 '> tom\n'
 '\n'
 '\n'
 "My understanding is that the 'expected errors' are basically\n"
 'known bugs in the warning system software - things are checked\n'
 "that don't have the right values in yet because they aren't\n"
 'set till after launch, and suchlike. Rather than fix the code\n'
 'and possibly introduce new bugs, they just tell the crew\n'
 "'ok, if you see a warning no. 213 before liftoff, ignore it'.\n"
 '\n'
 ' - Jonathan\n'
 '\n'
 '\n')
```

In [9]:

```python
# list out all topic names
pprint(list(dataset.target_names))
```

```
['alt.atheism',
 'comp.graphics',
 'comp.os.ms-windows.misc',
 'comp.sys.ibm.pc.hardware',
 'comp.sys.mac.hardware',
 'comp.windows.x',
 'misc.forsale',
 'rec.autos',
 'rec.motorcycles',
 'rec.sport.baseball',
 'rec.sport.hockey',
 'sci.crypt',
 'sci.electronics',
 'sci.med',
 'sci.space',
 'soc.religion.christian',
 'talk.politics.guns',
 'talk.politics.mideast',
 'talk.politics.misc',
 'talk.religion.misc']
```

B. Convert the posts (blobs of texts) into bag-of-word vectors. What is the dimensionality of these vectors? That is, what is the number of words that have appeared in this data set?

In [10]:

```python
# Model for creting a bag-of-words vector
from sklearn.feature_extraction.text import TfidfVectorizer
```

In [11]:

```
# instantiate the text to vector transform object
vectorizer = TfidfVectorizer(max_df=0.5, min_df=2, stop_words='english')
```

In [12]:

```
# create the bag of word vector
vectors = vectorizer.fit_transform(dataset.data)
print(vectors[:5])
```

```
  (0, 36254)	0.14365581894428547
  (0, 12514)	0.10213076203339368
  (0, 27452)	0.10684301809548442
  (0, 49936)	0.0601729255075507
  (0, 32912)	0.06540016969537528
  (0, 32219)	0.07543341899600832
  (0, 23695)	0.1701883013418222
  (0, 28064)	0.07985173645101952
  (0, 26362)	0.09109018303700456
  (0, 40860)	0.11821340713192517
  (0, 55580)	0.06619015166091179
  (0, 47259)	0.11840797817328158
  (0, 20932)	0.10607029880882898
  (0, 34893)	0.096567268121596
  (0, 49706)	0.19316713537009014
  (0, 30669)	0.04674013229588189
  (0, 11933)	0.0951701162387341
  (0, 43333)	0.08921734695029614
  (0, 45542)	0.10776256116529917
  (0, 12728)	0.145005721890891
  (0, 7903)	0.10410634610867979
  (0, 46669)	0.08217269860173604
  (0, 42366)	0.061870276832261296
  (0, 19464)	0.12669421106937861
  (0, 12395)	0.18248258600997347
  :	:
  (4, 37297)	0.10470101009575183
  (4, 9941)	0.12926268019328424
  (4, 46716)	0.12632823936080279
  (4, 31282)	0.1764594089841088
  (4, 46124)	0.09269312093643926
  (4, 33620)	0.12072229015841428
  (4, 25755)	0.08394905709969103
  (4, 14088)	0.11657256546626804
  (4, 25888)	0.06993095113747155
  (4, 29252)	0.1486404598718769
  (4, 29628)	0.18538624187287853
  (4, 15313)	0.05985326574720713
  (4, 54655)	0.08923774730252867
  (4, 50469)	0.15377215825175872
  (4, 41731)	0.10139809473262283
  (4, 42346)	0.056276528069909605
  (4, 29880)	0.035012259199459335
  (4, 19419)	0.0356850345326232
  (4, 21783)	0.16193300008641678
  (4, 36398)	0.04239455266466661
  (4, 52380)	0.04803759873641572
  (4, 19143)	0.03955515332763504
  (4, 9698)	0.05676754849852974
  (4, 42366)	0.04760864928254053
  (4, 54588)	0.07462852605769851
```

C. Use your favorite dimensionality reduction technique to compress these vectors into ones of K = 30 dimensions.

In [13]:

```
# Truncated SVD for dimensionality reduction
from sklearn.decomposition import TruncatedSVD
```

In [14]:

```
# the dimensionality reduction SVD object
K = 30
svd = TruncatedSVD(n_components=K, algorithm='randomized')
```

In [15]:

```
# perform the dimensionality reduction with transform
svd_vectors = svd.fit_transform(vectors)
```

D. Use your favorite supervised learning model to train a model that tries to predict the topic of a post from the vectorized representation of the post you obtained in the previous step.

In [16]:

```
# Use DecisionTreeClassifeir for the model
from sklearn.tree import DecisionTreeClassifier
```

In [17]:

```
# create an instance of the model
clf = DecisionTreeClassifier(random_state=0, max_depth=3)
```

In [18]:

```
# training the model
clf.fit(svd_vectors, dataset.target)
```

Out[18]:

```
DecisionTreeClassifier(max_depth=3, random_state=0)
```

In [19]:

```
# make a prediction
clf.predict(svd_vectors[0].reshape(1, -1))
```

Out[19]:

```
array([10])
```

E. Use the test data to tune your model. Make sure to include K as a hyperparameter as well. Use accuracy_score from sklearn.metrics as your evaluation metric. What is the highest accuracy you are able to achieve?

In [20]:

```
# create an instance of the model with K max features hyperparameter
clf = DecisionTreeClassifier(random_state=0, max_depth=3, max_features=K)
```

In [21]:

```
clf.fit(svd_vectors, dataset.target)
```

Out[21]:

```
DecisionTreeClassifier(max_depth=3, max_features=30, random_state=0)
```

In [22]:

```
# the test data
test_dataset = fetch_20newsgroups(subset='test')
```

In [23]:

```
# create a bag of words vector for the test data
vectors = vectorizer.fit_transform(test_dataset.data)
```

In [24]:

```
# perform dimensionality reduction
svd_vectors = svd.fit_transform(vectors)
```

In [25]:

```
# accuracy score metric
from sklearn.metrics import accuracy_score
```

In [26]:

```
y_true = test_dataset.target  # the true target values in the dataset
y_pred = clf.predict(svd_vectors)  # the predicted target values
```

In [27]:

```
# the accuracy score
score = accuracy_score(y_true, y_pred)
```

In [28]:

```
print("Accuracy Score %0.2f" %score)
```

```
Accuracy Score 0.17
```

In [ ]:

```

```