project

December 10, 2025

1 Project: Estimating Causal Effect for RHC Dataset

Author: Zixiao Tan zt80@duke.edu

[ ]:
```
%reset
```

[ ]:
```
!pip install fancyimpute cvxpy tqdm
```

2 1. Data Preparation

2.1 1.0 RHC Dataset

Right heart catheterization (RHC) is a diagnostic procedure for directly measuring cardiac function in critically ill patients. In an influential study, Connors et al. (1996) studied the effectiveness of RHC with an observational study design. The study collected data on 5735 hospitalized adult patients; 2184 of them are assigned to the treatment (Z = 1), receipt of RHC within 24 hours of admission, and the remaining 3551 assigned to the control condition (Z = 0). The outcome was survival at 30 days after admission.

The goal of this project is to assess the causal effect of RHC on the binary outcome, death at 30 days after admission. I will show several methods: outcome modeling, propensity scores, weighting, double robust method for estimating ATE.

[ ]:
```
import pandas as pd
import numpy as np

rhc = pd.read_csv("rhc_demo.csv")
rhc.head(5)
```

Below table summarizes the interpretations of most of the covariates.

Name	Interpretations
age	Age (years)
sex	Female
cat1_copd	COPD
cat1_mosfsep	MOSF w/Sepsis
cat1_mosfmal	MOSF w/Malignancy
cat1_chf	CHF
cat1_coma	Coma

Name	Interpretations
cat1_cirr	Cirrhosis
cat1_lung	Lung Cancer
cat1_colon	Colon Cancer
cat2_mosfsep	MOSF w/Sepsis
cat2_coma	Coma
cat2_mosfmal	MOSF w/Malignancy
cat2_lung	Lung Cancer
cat2_cirr	Cirrhosis
cat2_colon	Colon Cancer
ca_yes	Cancer-localized
ca_meta	Cancer-metastatic
pafi1	PaO2/F102 ratio
wtkilo1	Weight
surv2md1	Estimate of prob. of surviving 2 months
dementhx	Dementia, stroke or cerebral infarct, Parkinson's disease
gastr	Gastrointestinal diagnosis
wblc1	WBC
temp1	Temperature
das2d3pc	DASI-Duke Activity Status Index
chfhx	Congestive Heart Failure
hema	Hematological diagnosis
chrpulhx	Chronic pulmonary disease, severe pulmonary disease
cardiohx	Cardiovascular symptoms
meta	Metabolic diagnosis

2.2 1.1 Missing Data

We use MICE(Multiple Imputation by Chained Equations) algorithm to deal with missing data in RHC dataset.

[30]:
```
import pandas as pd
import numpy as np
from fancyimpute import IterativeImputer

# select and drop the first two columns
rhc.drop(columns=rhc.columns[:2], inplace=True)

# Convert treatment and outcome variables to integer flags
rhc['treatment'] = (rhc['swang1'] == 'RHC').astype(int)  # 0 = No RHC, 1 = RHC
rhc['death']     = rhc['death'].astype(int)              # 0 = survived, 1 =␣
 ↪died


categorical = [
    'cat1', 'ca', 'dementhx', 'gastr', 'chfhx', 'sex',
    'hema', 'chrpulhx', 'cardiohx', 'meta', 'adld3p'
```

```
]
for col in categorical:
    rhc[col] = rhc[col].fillna('Missing').astype('category')

# Perform MICE imputation on cat2 separately

cat2 = rhc['cat2'].astype('category')
codes = cat2.cat.codes.astype(float)
codes[codes == -1] = np.nan


imp = IterativeImputer()
codes_imp = imp.fit_transform(codes.values.reshape(-1, 1))


codes_imp = np.round(codes_imp).astype(int).flatten()


rhc['cat2'] = pd.Categorical.from_codes(codes_imp, cat2.cat.categories)


print("Number of missing values in cat2:", rhc['cat2'].isnull().sum())
```

```
Number of missing values in cat2: 0
```

[31]:
```
print("missing values for each column ")
print(rhc.isnull().sum())
```

```
missing values for each column
survtime     0
cat1         0
cat2         0
ca           0
death        0
cardiohx     0
chfhx        0
dementhx     0
chrpulhx     0
age          0
sex          0
surv2md1     0
das2d3pc     0
wblc1        0
temp1        0
pafi1        0
swang1       0
wtkilo1      0
gastr        0
```

```
meta         0
hema         0
adld3p       0
urin1        0
treatment    0
dtype: int64
```

2.3 1.2 Propensity Score Estimation

2.3.1 1.2.1 Propensity Score Estimation

We fit the propensity score model by logistic regression and visually assess the overlap.

[32]:
```
# 1.2
import statsmodels.api as sm
import statsmodels.formula.api as smf
import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np

ps_formula = (
    'treatment ~ C(cat1) + C(cat2) + C(ca) + pafi1 + wtkilo1 + surv2md1 + '
    'C(dementhx) + C(gastr) + wblc1 + temp1 + das2d3pc + age + '
    'C(chfhx) + C(sex) + urin1 + C(hema) + C(chrpulhx) + C(cardiohx) + '
    'C(meta) + C(adld3p) '
)
#
model_ps = smf.logit(ps_formula, data=rhc).fit(disp=False)

print(model_ps.summary())

#
rhc['pscore'] = model_ps.predict(rhc)
```

```
                           Logit Regression Results
==============================================================================
Dep. Variable:              treatment   No. Observations:                 5735
Model:                          Logit   Df Residuals:                     5699
Method:                           MLE   Df Model:                           35
Date:                Mon, 28 Apr 2025   Pseudo R-squ.:                 0.1452
Time:                        10:58:35   Log-Likelihood:                -3257.3
converged:                       True   LL-Null:                       -3810.7
Covariance Type:            nonrobust   LLR p-value:                1.053e-209
==============================================================================
================
                                 coef    std err          z      P>|z|
[0.025      0.975]
------------------------------------------------------------------------------
----------------
```

Intercept	1.7588	0.800	2.200	0.028	
0.192	3.326				
C(cat1)[T.CHF]	0.9199	0.146	6.300	0.000	
0.634	1.206				
C(cat1)[T.COPD]	-1.0592	0.172	-6.156	0.000	
-1.396	-0.722				
C(cat1)[T.Coma]	-1.1255	0.158	-7.128	0.000	
-1.435	-0.816				
C(cat1)[T.MOSF]	0.7884	0.078	10.162	0.000	
0.636	0.940				
C(cat1)[T.Other]	-0.9490	0.183	-5.199	0.000	
-1.307	-0.591				
C(cat2)[T.Colon Cancer]	2.2370	1.557	1.437	0.151	
-0.814	5.288				
C(cat2)[T.Coma]	-0.5066	0.476	-1.064	0.287	
-1.440	0.427				
C(cat2)[T.Lung Cancer]	0.4506	0.882	0.511	0.610	
-1.278	2.180				
C(cat2)[T.MOSF w/Malignancy]	0.6975	0.387	1.801	0.072	
-0.062	1.456				
C(cat2)[T.MOSF w/Sepsis]	1.3165	0.393	3.353	0.001	
0.547	2.086				
C(ca)[T.No]	1.1164	0.149	7.490	0.000	
0.824	1.409				
C(ca)[T.Yes]	0.3559	0.150	2.369	0.018	
0.061	0.650				
C(dementhx)[T.1]	-0.6643	0.113	-5.878	0.000	
-0.886	-0.443				
C(gastr)[T.Yes]	0.3312	0.088	3.779	0.000	
0.159	0.503				
C(chfhx)[T.1]	0.1686	0.097	1.740	0.082	
-0.021	0.359				
C(sex)[T.Male]	0.0464	0.063	0.740	0.459	
-0.076	0.169				
C(hema)[T.Yes]	-0.6423	0.138	-4.647	0.000	
-0.913	-0.371				
C(chrpulhx)[T.1]	-0.1238	0.094	-1.317	0.188	
-0.308	0.060				
C(cardiohx)[T.1]	0.1988	0.090	2.221	0.026	
0.023	0.374				
C(meta)[T.Yes]	-0.1295	0.147	-0.884	0.377	
-0.417	0.158				
C(adld3p)[T.1]	-0.5731	0.151	-3.789	0.000	
-0.869	-0.277				
C(adld3p)[T.2]	-0.4375	0.220	-1.987	0.047	
-0.869	-0.006				
C(adld3p)[T.3]	-0.5462	0.377	-1.451	0.147	
-1.284	0.192				

```
C(adld3p)[T.4]                  -0.8323      0.396     -2.100      0.036
-1.609      -0.055
C(adld3p)[T.5]                  -0.4229      0.303     -1.394      0.163
-1.017       0.172
C(adld3p)[T.6]                  -0.8525      0.368     -2.315      0.021
-1.574      -0.131
C(adld3p)[T.7]                  -0.6659      0.543     -1.226      0.220
-1.731       0.399
pafi1                           -0.0040      0.000    -12.848      0.000
-0.005      -0.003
wtkilo1                          0.0077      0.001      6.947      0.000
0.005       0.010
surv2md1                        -2.1795      0.249     -8.746      0.000
-2.668      -1.691
wblc1                            0.0025      0.003      0.969      0.332
-0.003       0.008
temp1                           -0.0561      0.018     -3.103      0.002
-0.092      -0.021
das2d3pc                         0.0004      0.006      0.069      0.945
-0.012       0.013
age                             -0.0068      0.002     -3.224      0.001
-0.011      -0.003
urin1                        2.816e-05   2.82e-05      0.997      0.319
-2.72e-05    8.35e-05
==============================================================================
================
```

[10]:
```python
# Check the rank of Design matrix
X = model_ps.model.exog
colnames = model_ps.model.exog_names

import numpy as np

# Compute the rank of the design matrix
rank = np.linalg.matrix_rank(X)

# Get the dimensions of the matrix
n_rows, n_cols = X.shape

print(f"Design matrix shape: {n_rows} × {n_cols}")
print(f"Matrix rank: {rank}")
print("Full rank:", "Yes" if rank == n_cols else "No")
```

```
Design matrix shape: 5735 × 36
Matrix rank: 36
Full rank: Yes
```

2.3.2 1.2.2 Propensity Score Distribution

[ ]:
```
plt.figure(figsize=(8, 4))
sns.kdeplot(data=rhc, x='pscore', hue='treatment', common_norm=False)
plt.title('Propensity Score Density by Treatment Group')
plt.xlabel('Propensity Score')
plt.ylabel('Density')
plt.tight_layout()
plt.show()

#
plt.figure(figsize=(8, 4))
sns.histplot(data=rhc, x='pscore', hue='treatment', bins=20, element='step')
plt.title('Propensity Score Histogram by Treatment Group')
plt.xlabel('Propensity Score')
plt.ylabel('Count')
plt.tight_layout()
plt.show()
```

2.4 1.3 Balance Check and Standardized Mean Difference (SMD)

2.4.1 1.3.1 Standardized Mean Difference (SMD)

[33]:
```
import numpy as np
import pandas as pd
from numpy import average
import patsy


#  compute IPW & overlap weights ---
rhc['ipw_weight'] = np.where(rhc['treatment']==1,
                             1/rhc['pscore'],
                             1/(1-rhc['pscore']))
rhc['ow_weight']  = np.where(rhc['treatment']==1,
                             1-rhc['pscore'],
                             rhc['pscore'])

# build the exact design-matrix you'll use in the logit ---

y, X = patsy.dmatrices(ps_formula, data=rhc, return_type='dataframe')
#    drop the intercept column; we don't balance that.
X = X.drop('Intercept', axis=1)

#  helper functions for SMDs
def smd_unw(df, treat, col):
    g0 = df.loc[df[treat]==0, col]; g1 = df.loc[df[treat]==1, col]
    m0, m1 = g0.mean(), g1.mean()
    s0, s1 = g0.std(),  g1.std()
```

```
    return (m1-m0)/np.sqrt((s0**2+s1**2)/2)

def smd_w(df, treat, col, wcol):
    x0 = df.loc[df[treat]==0, col]; w0 = df.loc[df[treat]==0, wcol]
    x1 = df.loc[df[treat]==1, col]; w1 = df.loc[df[treat]==1, wcol]
    m0 = average(x0, weights=w0); m1 = average(x1, weights=w1)
    v0 = average((x0-m0)**2, weights=w0); v1 = average((x1-m1)**2, weights=w1)
    return (m1-m0)/np.sqrt((v0+v1)/2)


import numpy as np
from numpy import average

#
df_bal = X.copy()
df_bal['treatment']   = rhc['treatment']
df_bal['ipw_weight']  = rhc['ipw_weight']
df_bal['ow_weight']   = rhc['ow_weight']

#
out = []
for col in X.columns:
    out.append({
        'covariate':      col,
        'SMD_unweighted': smd_unw(df_bal, 'treatment',   col),
        'SMD_IPW':        smd_w( df_bal, 'treatment',   col, 'ipw_weight'),
        'SMD_OW':         smd_w( df_bal, 'treatment',   col, 'ow_weight'),
    })

smd_compare = (
    pd.DataFrame(out)
      .assign(abs_SMD_OW=lambda d: d['SMD_OW'].abs())
      .sort_values('abs_SMD_OW', ascending=False)
)

print(smd_compare[['covariate','SMD_unweighted','SMD_IPW','SMD_OW']])
```

	covariate	SMD_unweighted	SMD_IPW	SMD_OW
31	temp1	-0.021371	-0.019588	-3.946086e-15
30	wblc1	0.083558	0.008637	-2.874201e-16
27	pafi1	-0.433242	0.001451	-2.640352e-16
3	C(cat1)[T.MOSF]	0.390869	-0.002812	-1.190487e-16
13	C(gastr)[T.Yes]	0.120910	-0.008236	7.382539e-17
14	C(chfhx)[T.1]	0.069492	0.004685	-7.041134e-17
18	C(cardiohx)[T.1]	0.115618	0.009936	-6.957818e-17
16	C(hema)[T.Yes]	-0.061712	0.009282	5.862800e-17
0	C(cat1)[T.CHF]	0.095027	0.012998	-4.683244e-17
1	C(cat1)[T.COPD]	-0.342364	-0.001375	-3.438065e-17

20	C(adld3p)[T.1]	-0.137562	0.000811	-3.393926e-17
19	C(meta)[T.Yes]	-0.028088	-0.010872	3.356318e-17
21	C(adld3p)[T.2]	-0.079467	-0.015933	-2.536092e-17
7	C(cat2)[T.Lung Cancer]	-0.057467	-0.003731	-1.689476e-17
6	C(cat2)[T.Coma]	-0.088566	-0.003613	1.541747e-17
22	C(adld3p)[T.3]	-0.047601	-0.010254	-1.084672e-17
25	C(adld3p)[T.6]	-0.074307	0.016961	-1.028921e-17
26	C(adld3p)[T.7]	-0.056723	-0.015833	-7.672770e-18
5	C(cat2)[T.Colon Cancer]	0.009167	0.000591	-3.133881e-18
29	surv2md1	-0.198498	-0.017546	0.000000e+00
28	wtkilo1	0.255679	0.001967	0.000000e+00
24	C(adld3p)[T.5]	-0.053471	-0.023271	0.000000e+00
32	das2d3pc	0.062583	0.005996	0.000000e+00
33	age	-0.061352	0.010122	0.000000e+00
17	C(chrpulhx)[T.1]	-0.192322	0.007286	0.000000e+00
23	C(adld3p)[T.4]	-0.081817	-0.008314	0.000000e+00
15	C(sex)[T.Male]	0.093127	0.012025	0.000000e+00
12	C(dementhx)[T.1]	-0.163129	-0.035144	0.000000e+00
11	C(ca)[T.Yes]	-0.071840	0.007643	0.000000e+00
10	C(ca)[T.No]	0.104294	-0.022928	0.000000e+00
9	C(cat2)[T.MOSF w/Sepsis]	0.217651	0.000349	0.000000e+00
8	C(cat2)[T.MOSF w/Malignancy]	-0.163797	0.003740	0.000000e+00
4	C(cat1)[T.Other]	-0.175228	0.006074	0.000000e+00
2	C(cat1)[T.Coma]	-0.207278	-0.027898	0.000000e+00
34	urin1	0.005727	0.016422	0.000000e+00

2.4.2 1.3.2 Balance Plot

From the Love plot on the comparisons between original standardized mean differences (ASD) of all covariates and ASD of all covariates after IPW, OW, ATT weighting methods, we observe that the weighted ASD of all covariates are improved (under three different weighting methods) compared to the unweighted. In particular, the resulting weighted ASD for all covariates after OW (Overlap Weighting) are exactly 0.

[ ]:
```
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd


df_plot = smd_compare[['covariate','SMD_unweighted','SMD_IPW','SMD_OW']]


df_melt = (
    df_plot
    .melt(id_vars='covariate',
          value_vars=['SMD_unweighted','SMD_IPW','SMD_OW'],
          var_name='Weighting',
          value_name='SMD')
```

```
)
df_melt['abs_SMD'] = df_melt['SMD'].abs()
df_melt['Weighting'] = df_melt['Weighting'].map({
    'SMD_unweighted':'Unadjusted',
    'SMD_IPW':'IPW',
    'SMD_OW':'Overlap'
})


order = (
    df_plot
    .assign(abs_unw=lambda d: d['SMD_unweighted'].abs())
    .sort_values('abs_unw', ascending=False)['covariate']
)
df_melt['covariate'] = pd.Categorical(df_melt['covariate'],
                                      categories=order,
                                      ordered=True)


plt.figure(figsize=(6, len(order)*0.25))
sns.scatterplot(
    data=df_melt,
    x='abs_SMD',
    y='covariate',
    hue='Weighting',
    s=50,
    alpha=0.8
)


plt.axvline(0, color='grey', linestyle='--', linewidth=1, alpha=0.5)

plt.axvline(0.1, color='grey', linestyle='--', linewidth=1, alpha=0.8)

plt.title('Balance Plot (|SMD|): Unadjusted vs. IPW vs. Overlap')
plt.xlabel('Absolute Standardized Mean Difference')
plt.ylabel('')
plt.legend(title='')
plt.tight_layout()
plt.show()
```

3 2. Outcome Regression

First, we applied outcome regression to estimate ATE. The model's design matrix was confirmed to be full rank. We predicted each unit's potential outcomes under treatment and control, and computed the ATE as the mean difference in predicted probabilities, along with relative risk (RR) and odds ratio (OR). To assess uncertainty, we constructed confidence intervals using both bootstrap

resampling and the sandwich (robust) variance estimator.

3.1 2.1 Logistic Regression

[34]:
```
import pandas as pd
import numpy as np
import statsmodels.formula.api as smf


# fit outcome regression
outcome_formula = (
    'death ~ C(cat1) + C(cat2) + C(ca) + pafi1 + wtkilo1 + surv2md1 + '
    'C(dementhx) + C(gastr) + wblc1 + temp1 + das2d3pc + age + '
    'C(chfhx) + C(sex) + urin1 + C(hema) + C(chrpulhx) + C(cardiohx) + '
    'C(meta) + C(adld3p) + treatment'
)

model_outcome = smf.logit(outcome_formula, data=rhc).fit(disp=False,␣
 ↪maxiter=100)

print(model_outcome.summary())
```

```
                           Logit Regression Results
==============================================================================
Dep. Variable:                  death   No. Observations:                 5735
Model:                          Logit   Df Residuals:                     5698
Method:                           MLE   Df Model:                           36
Date:                Mon, 28 Apr 2025   Pseudo R-squ.:                  0.1745
Time:                        10:58:51   Log-Likelihood:                -3068.0
converged:                      False   LL-Null:                       -3716.7
Covariance Type:            nonrobust   LLR p-value:                3.646e-249
==============================================================================
                                 coef    std err          z      P>|z|
[0.025      0.975]
------------------------------------------------------------------------------
Intercept                      7.7025      0.927      8.308      0.000
5.885       9.520
C(cat1)[T.CHF]                 0.1580      0.153      1.031      0.303
-0.143       0.459
C(cat1)[T.COPD]                0.2850      0.145      1.962      0.050
0.000       0.570
C(cat1)[T.Coma]                0.3723      0.163      2.290      0.022
0.054       0.691
C(cat1)[T.MOSF]                0.0793      0.085      0.928      0.353
-0.088       0.247
C(cat1)[T.Other]               0.4383      0.176      2.493      0.013
```

```
0.094       0.783
C(cat2)[T.Colon Cancer]         22.7946   2.79e+05   8.18e-05      1.000
-5.46e+05    5.46e+05
C(cat2)[T.Coma]                 -1.7417      0.554     -3.144      0.002
-2.827      -0.656
C(cat2)[T.Lung Cancer]          18.7316   1.82e+04      0.001      0.999
-3.56e+04    3.57e+04
C(cat2)[T.MOSF w/Malignancy]    -1.8237      0.471     -3.868      0.000
-2.748      -0.900
C(cat2)[T.MOSF w/Sepsis]        -1.5873      0.478     -3.321      0.001
-2.524      -0.651
C(ca)[T.No]                     -0.9639      0.197     -4.894      0.000
-1.350      -0.578
C(ca)[T.Yes]                    -0.7519      0.203     -3.699      0.000
-1.150      -0.353
C(dementhx)[T.1]                 0.3024      0.116      2.602      0.009
0.075       0.530
C(gastr)[T.Yes]                  0.2404      0.096      2.493      0.013
0.051       0.429
C(chfhx)[T.1]                    0.5367      0.105      5.118      0.000
0.331       0.742
C(sex)[T.Male]                   0.2493      0.065      3.836      0.000
0.122       0.377
C(hema)[T.Yes]                   0.8266      0.166      4.966      0.000
0.500       1.153
C(chrpulhx)[T.1]                 0.1760      0.100      1.761      0.078
-0.020       0.372
C(cardiohx)[T.1]                 0.1777      0.095      1.868      0.062
-0.009       0.364
C(meta)[T.Yes]                   0.2250      0.154      1.463      0.143
-0.076       0.526
C(adld3p)[T.1]                  -0.5940      0.135     -4.388      0.000
-0.859      -0.329
C(adld3p)[T.2]                  -0.4521      0.200     -2.264      0.024
-0.844      -0.061
C(adld3p)[T.3]                  -0.2611      0.343     -0.760      0.447
-0.934       0.412
C(adld3p)[T.4]                  -0.0071      0.335     -0.021      0.983
-0.663       0.649
C(adld3p)[T.5]                  -0.3284      0.291     -1.129      0.259
-0.899       0.242
C(adld3p)[T.6]                  -0.5074      0.308     -1.646      0.100
-1.111       0.097
C(adld3p)[T.7]                  -0.8878      0.448     -1.980      0.048
-1.767      -0.009
pafi1                            0.0010      0.000      3.320      0.001
0.000       0.002
wtkilo1                         -0.0026      0.001     -2.395      0.017
```

```
-0.005      -0.000
surv2md1                        -3.9180      0.297    -13.176      0.000
-4.501      -3.335
wblc1                            0.0076      0.003      2.506      0.012
0.002       0.014
temp1                           -0.0498      0.020     -2.502      0.012
-0.089      -0.011
das2d3pc                        -0.0564      0.006     -8.701      0.000
-0.069      -0.044
age                              0.0115      0.002      5.051      0.000
0.007       0.016
urin1                           -0.0002   2.99e-05     -5.865      0.000
-0.000      -0.000
treatment                        0.2516      0.071      3.538      0.000
0.112       0.391
==============================================================================
================
```

```
C:\New Duke\5. Courses\STA 663\CS 61A\CS61A\venv\lib\site-
packages\statsmodels\base\model.py:607: ConvergenceWarning: Maximum Likelihood
optimization failed to converge. Check mle_retvals
  warnings.warn("Maximum Likelihood optimization failed to "
```

[15]:
```python
# Check the rank of Design matrix
X = model_outcome.model.exog
colnames = model_outcome.model.exog_names

import numpy as np

# Compute the rank of the design matrix
rank = np.linalg.matrix_rank(X)

# Get the dimensions of the matrix
n_rows, n_cols = X.shape

print(f"Design matrix shape: {n_rows} × {n_cols}")
print(f"Matrix rank: {rank}")
print("Full rank:", "Yes" if rank == n_cols else "No")
```

```
Design matrix shape: 5735 × 37
Matrix rank: 37
Full rank: Yes
```

3.2 2.2 ATE Estimand

[35]:
```python
#
rhc_trt = rhc.copy()
rhc_trt['treatment'] = 1
rhc_ctrl = rhc.copy()
```

```
rhc_ctrl['treatment'] = 0

#
prob_trt = model_outcome.predict(rhc_trt)
prob_ctrl = model_outcome.predict(rhc_ctrl)

#
ATE = np.mean(prob_trt - prob_ctrl)
risk_trt = prob_trt.mean()
risk_ctrl = prob_ctrl.mean()
RR  = risk_trt / risk_ctrl
OR  = (risk_trt/(1-risk_trt)) / (risk_ctrl/(1-risk_ctrl))

print(f'ATE (Risk Difference): {ATE:.4f}')
print(f'Relative Risk (RR):       {RR:.4f}')
print(f'Odds Ratio (OR):         {OR:.4f}')
```

```
ATE (Risk Difference): 0.0451
Relative Risk (RR):       1.0714
Odds Ratio (OR):         1.2211
```

3.3 2.3 Confidence Interval

3.3.1 2.3.1 Bootstrap Confidence Interval

[ ]:
```
import pandas as pd
import numpy as np
import patsy
import statsmodels.formula.api as smf
import warnings

#
warnings.filterwarnings("ignore")

outcome_formula = (
    'death ~ C(cat1) + C(cat2) + C(ca) + pafi1 + wtkilo1 + surv2md1 + '
    'C(dementhx) + C(gastr) + wblc1 + temp1 + das2d3pc + age + '
    'C(chfhx) + C(sex) + urin1 + C(hema) + C(chrpulhx) + C(cardiohx) + '
    'C(meta) + C(adld3p) + treatment'
)
rhs_formula = outcome_formula.split('~', 1)[1]

def compute_effect(df):
    #
    model = smf.logit(outcome_formula, data=df).fit(disp=False, method='bfgs',␣
 ↪maxiter=100)
    df_trt  = df.copy(); df_trt['treatment'] = 1
    df_ctrl = df.copy(); df_ctrl['treatment'] = 0
```

```
    p_trt   = model.predict(df_trt)
    p_ctrl  = model.predict(df_ctrl)
    ate = np.mean(p_trt - p_ctrl)
    rr  = np.mean(p_trt) / np.mean(p_ctrl)
    or_ = (np.mean(p_trt)/(1-np.mean(p_trt))) / (np.mean(p_ctrl)/(1-np.
 ↪mean(p_ctrl)))
    return np.array([ate, rr, or_])

def bootstrap_ate_with_rank_check(data, func, num_rep=1000, seed=0):
    np.random.seed(seed)
    boot_res = []
    for _ in range(num_rep):
        sample = data.sample(n=len(data), replace=True)
        X_samp = patsy.dmatrix(rhs_formula, sample, return_type='dataframe').
 ↪values
        if np.linalg.matrix_rank(X_samp) < X_samp.shape[1]:
            continue
        boot_res.append(func(sample))

    boot_res = np.vstack(boot_res)
    est_mean = boot_res.mean(axis=0)
    lower_ci = np.percentile(boot_res, 2.5, axis=0)
    upper_ci = np.percentile(boot_res, 97.5, axis=0)

    return pd.DataFrame({
        'estimate': est_mean,
        '2.5%':     lower_ci,
        '97.5%':    upper_ci
    }, index=['ATE','RR','OR'])

ci_table = bootstrap_ate_with_rank_check(rhc, compute_effect, num_rep=1000,␣
 ↪seed=0)
print(ci_table)
```

3.3.2 2.3.2 Sandwich estimator

The sandwich variance estimator for the parameter $\hat{\beta}$ is:

$$\widehat{\mathrm{Var}}(\hat{\beta}) = (X^\top W X)^{-1} \left(\sum_{i=1}^{n} u_i u_i^\top \right) (X^\top W X)^{-1}$$

$$\widehat{ATE} = \frac{1}{n} \sum_{i=1}^{n} \left(\hat{Y}_i(1) - \hat{Y}_i(0) \right)$$

Using the delta method, the variance of $\widehat{ATE}$ can be approximated by:

$$\widehat{\mathrm{Var}}(\widehat{ATE}) = \nabla_\beta \widehat{ATE}^\top \, \widehat{\mathrm{Var}}(\hat{\beta}) \, \nabla_\beta \widehat{ATE}$$

95% confidence interval for ATE is:

$$\widehat{ATE} \pm 1.96 \times \sqrt{\widehat{\mathrm{Var}}(\widehat{ATE})}$$

[16]:
```
import pandas as pd
import numpy as np
import scipy.special as sc
import statsmodels.formula.api as smf


outcome_formula = (
    'death ~ C(cat1) + C(cat2) + C(ca) + pafi1 + wtkilo1 + surv2md1 + '
    'C(dementhx) + C(gastr) + wblc1 + temp1 + das2d3pc + age + '
    'C(chfhx) + C(sex) + urin1 + C(hema) + C(chrpulhx) + C(cardiohx) + '
    'C(meta) + C(adld3p) + treatment'
)


res     = smf.logit(outcome_formula, data=rhc).fit(disp=False)
res_rob = smf.logit(outcome_formula, data=rhc).fit(disp=False, cov_type="HC3")

b      = res.params.values
V      = res.cov_params().values
V_rob  = res_rob.cov_params().values


exog    = res.model.exog                      # (n×p)
X_trt   = exog.copy();   X_trt[:, -1] = 1     #
X_ctrl  = exog.copy();   X_ctrl[:, -1] = 0


_trt    = X_trt.dot(b)
_ctrl   = X_ctrl.dot(b)
p_trt   = sc.expit( _trt)    #
p_ctrl  = sc.expit( _ctrl)


ate = np.mean(p_trt - p_ctrl)


diff = X_trt * (p_trt*(1-p_trt))[:, None] \
     - X_ctrl * (p_ctrl*(1-p_ctrl))[:, None]
g = diff.mean(axis=0)       # (p,)
```

```
def delta_ci(V_mat):
    var_ate  = g.dot(V_mat).dot(g)
    se_ate   = np.sqrt(var_ate)
    return ate, ate - 1.96*se_ate, ate + 1.96*se_ate

ate_reg,    l_reg, u_reg = delta_ci(V)
ate_rob,    l_rob, u_rob = delta_ci(V_rob)

print(f"ATE = {ate_reg:.4f}  (95% CI: [{l_reg:.4f}, {u_reg:.4f}])")
print(f"Sandwich : ATE = {ate_rob:.4f}  (95% CI: [{l_rob:.4f}, {u_rob:.4f}])")
```

```
C:\New Duke\5. Courses\STA 663\CS 61A\CS61A\venv\lib\site-
packages\statsmodels\base\model.py:607: ConvergenceWarning: Maximum Likelihood
optimization failed to converge. Check mle_retvals
  warnings.warn("Maximum Likelihood optimization failed to "
```

```
ATE = 0.0451  (95% CI: [0.0203, 0.0699])
Sandwich : ATE = 0.0451  (95% CI: [0.0204, 0.0698])
```

```
C:\New Duke\5. Courses\STA 663\CS 61A\CS61A\venv\lib\site-
packages\statsmodels\base\model.py:607: ConvergenceWarning: Maximum Likelihood
optimization failed to converge. Check mle_retvals
  warnings.warn("Maximum Likelihood optimization failed to "
```

4 3. Weighting

In this section, we demonstrate several weighting methods including inverse probability weighting, overlap weighting. Our first estimand is the causal risk difference among the combined population (ATE). We use IPW weighting method first and compare the standard errors estimated by robust sandwich estimator (default choice) or bootstrap.

4.1 3.1 IPW

$$\mathbb{E}\left[\frac{ZY}{e(X)} - \frac{(1-Z)Y}{1-e(X)}\right] = \tau^{\text{ATE}}.$$

$$\begin{cases} w_1(X_i) = \frac{1}{e(X_i)}, & \text{for } Z_i = 1 \\ w_0(X_i) = \frac{1}{1-e(X_i)}, & \text{for } Z_i = 0 \end{cases}$$

[17]:
```
# 1) weights
rhc['ipw_weight'] = np.where(rhc['treatment']==1,
                             1/rhc['pscore'],
                             1/(1-rhc['pscore']))

# 2) sandwich SE
ipw_mod = smf.wls('death ~ treatment', data=rhc, weights=rhc['ipw_weight']) \
            .fit(cov_type='HC0')
```

```
print("ATE (IPW)          =", ipw_mod.params['treatment'])
print("Sandwich SE (HC0)  =", ipw_mod.bse['treatment'])

# 3) bootstrap SE
def ipw_ate(df):
    w = df['treatment']/df['pscore'] + (1-df['treatment'])/(1-df['pscore'])
    return smf.wls('death ~ treatment', data=df, weights=w) \
              .fit(disp=False).params['treatment']

B = 500
ates = np.array([ipw_ate(rhc.sample(len(rhc), replace=True)) for _ in range(B)])
print(f"Bootstrap SE (B={B}) =", ates.std(ddof=1))
```

```
ATE (IPW)          = 0.04250964999314673
Sandwich SE (HC0)  = 0.015243504262467322
Bootstrap SE (B=500) = 0.015655216523874523
```

[18]:
```
ate_ipw  = ipw_mod.params['treatment']
se_ipw   = ipw_mod.bse['treatment']


ci_lower = ate_ipw - 1.96 * se_ipw
ci_upper = ate_ipw + 1.96 * se_ipw

print(f"ATE = {ate_ipw:.4f}")
print(f"95% CI (HC0 normal) = [{ci_lower:.4f}, {ci_upper:.4f}]")
```

```
ATE = 0.0425
95% CI (HC0 normal) = [0.0126, 0.0724]
```

[19]:
```
lower_pct = np.percentile(ates, 2.5)
upper_pct = np.percentile(ates, 97.5)

print(f"95% CI (bootstrap percentile) = [{lower_pct:.4f}, {upper_pct:.4f}]")
```

```
95% CI (bootstrap percentile) = [0.0110, 0.0724]
```

4.2 3.2 IPW trimming

Next, we consider to estimate using IPW trimming method. We use the rule of thumb threshold $\delta = 0.1$, i.e., excluding those whose estimated propensity scores are less than 0.1 or bigger than 0.9.

[20]:
```
import numpy as np
import pandas as pd
import statsmodels.formula.api as smf



# 1. Trim
```

```python
eps = 0.1
mask = (rhc['pscore'] >= eps) & (rhc['pscore'] <= 1-eps)
rhc_trim = rhc.loc[mask].copy()
print(f"before trimming N={len(rhc)}, after trimming N={len(rhc_trim)}")

# 2.
rhc_trim['ipw_trim'] = (
    rhc_trim['treatment'] / rhc_trim['pscore']
  + (1 - rhc_trim['treatment']) / (1 - rhc_trim['pscore'])
)


ipw_trim_mod = smf.wls(
    'death ~ treatment',
    data = rhc_trim,
    weights = rhc_trim['ipw_trim']
).fit(cov_type='HC0')

ate_trim    = ipw_trim_mod.params['treatment']
se_trim     = ipw_trim_mod.bse['treatment']
ci_trim_lo  = ate_trim - 1.96 * se_trim
ci_trim_hi  = ate_trim + 1.96 * se_trim

print(f" Trimmed IPW ATE               = {ate_trim:.4f}")
print(f" Sandwich CI (normal)          = [{ci_trim_lo:.4f}, {ci_trim_hi:.4f}]")


def ipw_trim_ate(df):
    w = df['treatment']/df['pscore'] + (1-df['treatment'])/(1-df['pscore'])
    fit = smf.wls('death ~ treatment', data=df, weights=w).fit(disp=False)
    return fit.params['treatment']

B = 500   #
np.random.seed(0)
ates_trim = np.empty(B)

for b in range(B):
    samp = rhc_trim.sample(n=len(rhc_trim), replace=True)

    ates_trim[b] = ipw_trim_ate(samp)

low_pct  = np.percentile(ates_trim, 2.5)
high_pct = np.percentile(ates_trim, 97.5)
print(f" Bootstrap CI (B={B})  = [{low_pct:.4f}, {high_pct:.4f}]")
```

```
before trimming N=5735, after trimming N=5240
 Trimmed IPW ATE               = 0.0414
 Sandwich CI (normal)          = [0.0125, 0.0703]
```

```
Bootstrap    CI (B=500)  = [0.0146, 0.0733]
```

4.3 3.3 Overlap Weight

$$\mathbb{E}\left[ZY \cdot (1 - e(X)) + (1 - Z)Y \cdot e(X)\right] = \tau_{\text{OW}}^{\text{ATE}}.$$

$$\begin{cases} w_1(X_i) = 1 - e(X_i), & \text{for } Z_i = 1 \\ w_0(X_i) = e(X_i), & \text{for } Z_i = 0 \end{cases}$$

[21]:
```python
import numpy as np
import statsmodels.formula.api as smf


rhc['ow_weight']  = np.where(rhc['treatment']==1,
                             1-rhc['pscore'],
                             rhc['pscore'])


ow_mod = smf.wls(
    'death ~ treatment',
    data=rhc,
    weights=rhc['ow_weight']
).fit(cov_type='HC0')

ate_ow    = ow_mod.params['treatment']
se_ow     = ow_mod.bse['treatment']
ci_ow_lo  = ate_ow - 1.96 * se_ow
ci_ow_hi  = ate_ow + 1.96 * se_ow

print("=== Overlap-Weighted ATE ===")
print(f"ATE (OW)                = {ate_ow:.4f}")
print(f"Sandwich CI (normal)    = [{ci_ow_lo:.4f}, {ci_ow_hi:.4f}]")


def ow_ate(df):
    fit = smf.wls('death ~ treatment', data=df, weights=df['ow_weight']) \
              .fit(disp=False)
    return fit.params['treatment']

B = 500
np.random.seed(0)
ates_ow = np.empty(B)

for b in range(B):
    samp = rhc.sample(n=len(rhc), replace=True)

    ates_ow[b] = ow_ate(samp)
```

```
low_pct  = np.percentile(ates_ow, 2.5)
high_pct = np.percentile(ates_ow, 97.5)
print(f"Bootstrap CI (percentile, B={B}) = [{low_pct:.4f}, {high_pct:.4f}]")
```

```
=== Overlap-Weighted ATE ===
ATE (OW)                = 0.0440
Sandwich CI (normal)    = [0.0162, 0.0718]
Bootstrap CI (percentile, B=500) = [0.0163, 0.0708]
```

5 4. Double Robust

In this section, we introduce the augmented weighting method and follow the same procedure as the previous section. For IPW, augmented weighting is also known as the doubly-robust estimation. It is only singly-robust for other weighting schemes.

5.1 4.1 ATE

$$\mathbb{E}\left[\left(\frac{Z}{e(X)} - \frac{1-Z}{1-e(X)}\right)(Y - m(Z,X)) + (m(1,X) - m(0,X))\right] = \tau_{\text{DR}}^{\text{ATE}}.$$

outcome model for treated:

$$m(1,X) = \mathbb{E}[Y \mid Z = 1, X]$$

outcome model for control:

$$m(0,X) = \mathbb{E}[Y \mid Z = 0, X]$$

propensity score:

$$e(X) = \mathbb{P}(Z = 1 \mid X)$$

[22]:
```
dr_term = (
    rhc['treatment'] * (rhc['death'] - prob_trt) / rhc['pscore']
  - (1 - rhc['treatment']) * (rhc['death'] - prob_ctrl) / (1 - rhc['pscore'])
  + (prob_trt - prob_ctrl)
)

#
ATE_dr = dr_term.mean()
print(f'DR ATE: {ATE_dr:.4f}')
```

```
DR ATE: 0.0403
```

5.2 4.2 Sandwich-based Confidence Interval

[23]:
```
n = len(dr_term)

var_sandwich = dr_term.var(ddof=1) / n
```

```
se_sandwich  = np.sqrt(var_sandwich)

ci_lower = ATE_dr - 1.96 * se_sandwich
ci_upper = ATE_dr + 1.96 * se_sandwich

print(f'DR ATE (sandwich SE): {ATE_dr:.4f} ± {1.96*se_sandwich:.4f}')
print(f'95% CI: [{ci_lower:.4f}, {ci_upper:.4f}]')
```

```
DR ATE (sandwich SE): 0.0403 ± 0.0270
95% CI: [0.0133, 0.0672]
```

5.3 4.3 Confidence Interval via Bootstrap

[ ]:
```
import pandas as pd
import numpy as np
import patsy
import statsmodels.formula.api as smf
import warnings

warnings.filterwarnings("ignore")



rhs_formula = outcome_formula.split('~', 1)[1]

def compute_dr(sample):
    #
    m_ps = smf.logit(ps_formula, data=sample).fit(disp=False, method='bfgs',␣
 ↪maxiter=100)
    samp = sample.copy()
    samp['pscore'] = m_ps.predict(samp)

    #
    m_out = smf.logit(outcome_formula, data=samp).fit(disp=False,␣
 ↪method='bfgs', maxiter=100)

    #
    samp_trt  = samp.assign(treatment=1)
    samp_ctrl = samp.assign(treatment=0)
    m1 = m_out.predict(samp_trt)
    m0 = m_out.predict(samp_ctrl)

    #
    dr = (
        samp['treatment'] * (samp['death'] - m1) / samp['pscore']
      - (1 - samp['treatment']) * (samp['death'] - m0) / (1 - samp['pscore'])
      + (m1 - m0)
```

```
    )
    return dr.mean()

def bootstrap_dr_with_rank_check(data, num_rep=1000, seed=0):
    np.random.seed(seed)
    ates = []
    for _ in range(num_rep):  #
        samp = data.sample(n=len(data), replace=True)
        X_samp = patsy.dmatrix(rhs_formula, samp, return_type='dataframe').
 ↪values
        if np.linalg.matrix_rank(X_samp) < X_samp.shape[1]:
            continue
        ates.append(compute_dr(samp))

    ates = np.array(ates)
    est_mean   = ates.mean()
    lower_ci   = np.percentile(ates, 2.5)
    upper_ci   = np.percentile(ates, 97.5)

    return pd.Series({
        'DR ATE':         est_mean,
        '2.5% CI bound':  lower_ci,
        '97.5% CI bound': upper_ci
    })

ci_series = bootstrap_dr_with_rank_check(rhc, num_rep=1000, seed=0)
# print(ci_series)
```