DoubleML

January 4, 2026

1 DoubleML

Exploration of household water risk using DoubleML on MICS data.

1.1 EDA and preprocessing

[1]:
```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import optuna
from sklearn.preprocessing import OrdinalEncoder, OneHotEncoder
from sklearn.compose import ColumnTransformer
from doubleml import DoubleMLData, DoubleMLPLR
from xgboost import XGBClassifier

import warnings
warnings.filterwarnings("ignore")
```

[2]:
```python
# Load raw data
mics = pd.read_csv("mics.csv", low_memory=False)
mics.head()
```

[2]:
```
   HH1  HH2  HINT  HH3  HH4  HH5D     HH5M  HH5Y      HH6      HH7  …  \
0    1    5  12.0   12   11     2  6. JUNE  2017  2. Rural  1. EAST  …
1    1   14  15.0   15   11     3  6. JUNE  2017  2. Rural  1. EAST  …
2    1   22  15.0   15   11     4  6. JUNE  2017  2. Rural  1. EAST  …
3    2    3  12.0   12   11     5  6. JUNE  2017  2. Rural  1. EAST  …
4    2   11  12.0   12   11     5  6. JUNE  2017  2. Rural  1. EAST  …

  NoRiskHome_01_2 RiskHome_0_12 RiskSource_0_12 water_treatment3  Any_U5  \
0               1             1               1                0       1
1               1             1               0                0       1
2               0             1               1                0       1
3               0             1               1                0       1
4               1             1               1                0       0

   Region windex_ur windex5_categ  wq27_decile  SomeRiskHome
0       1         2          Poor            7             1
```

```
1       1         2         Poor          1             1
2       1         2       Middle          8             1
3       1         2       Middle          8             1
4       1         1         Poor          8             1

[5 rows x 784 columns]
```

[3]:

```python
# Keep only the columns used downstream
required_cols = [
    "windex_ur", "windex5", "helevel", "country_cat", "urban",
    "WS1_g", "wq27_decile", "WQ15_g", "RiskSource",
    "water_treatment", "VeryHighRiskHome", "SomeRiskHome",
]

mics = mics[required_cols].copy()
mics[required_cols].info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 54340 entries, 0 to 54339
Data columns (total 12 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   windex_ur         54340 non-null  int64
 1   windex5           54340 non-null  object
 2   helevel           54340 non-null  object
 3   country_cat       54340 non-null  object
 4   urban             54340 non-null  object
 5   WS1_g             54340 non-null  object
 6   wq27_decile       54340 non-null  int64
 7   WQ15_g            54340 non-null  object
 8   RiskSource        54340 non-null  object
 9   water_treatment   54340 non-null  int64
 10  VeryHighRiskHome  54340 non-null  int64
 11  SomeRiskHome      54340 non-null  int64
dtypes: int64(5), object(7)
memory usage: 5.0+ MB
```

[4]:

```python
# Map string categories to numeric codes for model consumption
HE_LEVEL = {
    "No education": 0,
    "Primary": 1,
    "Secondary or higher": 2,
}

URBAN = {
    "Rural": 0,
    "Urban": 1,
}
```

```
RISK_SOURCE = {
    "No risk": 0,
    "Moderate to high risk": 1,
    "Very high risk": 2,
}

mics["helevel"] = mics["helevel"].map(HE_LEVEL)
mics["urban"] = mics["urban"].map(URBAN)
mics["RiskSource"] = mics["RiskSource"].map(RISK_SOURCE)
```

[6]:

```
"""Encode categorical variables.
- WQ15_g: one-hot with reference level dropped.
- windex5: ordinal to preserve welfare ordering.
- country_cat, WS1_g, water_treatment: one-hot with reference.
Other columns pass through unchanged.
"""
wq15_categories = [[
    "Treat: Nothing",
    "Treat: Strain/Settle",
    "Treat: Chlorine/Aquatabs/PUR",
    "Treat: Boil",
    "Treat: Other",
]]

windex5_cat = [[
    "Poorest",
    "Poor",
    "Middle",
    "Rich",
    "Richest",
]]

cat_default = ["country_cat", "WS1_g", "water_treatment"]
cat_wq15 = ["WQ15_g"]
ord_windex5 = ["windex5"]

ct = ColumnTransformer(
    [
        (
            "wq15",
            OneHotEncoder(
                categories=wq15_categories,
                drop="first",
                sparse_output=False,
                handle_unknown="ignore",
            ),
```

```
            cat_wq15,
        ),
        (
            "windex5",
            OrdinalEncoder(categories=windex5_cat),
            ord_windex5,
        ),
        (
            "other_cat",
            OneHotEncoder(drop="first", sparse_output=False,␣
↪handle_unknown="ignore"),
            cat_default,
        ),
    ],
    remainder="passthrough",
    verbose_feature_names_out=False,
)

ct.set_output(transform="pandas")
mics = ct.fit_transform(mics)
mics.head()
```

[6]:
```
   WQ15_g_Treat: Strain/Settle  WQ15_g_Treat: Chlorine/Aquatabs/PUR  \
0                          0.0                                  0.0
1                          0.0                                  0.0
2                          0.0                                  0.0
3                          0.0                                  0.0
4                          0.0                                  0.0

   WQ15_g_Treat: Boil  WQ15_g_Treat: Other  windex5  country_cat_Benin  \
0                 0.0                  0.0      1.0                0.0
1                 0.0                  0.0      1.0                0.0
2                 0.0                  0.0      2.0                0.0
3                 0.0                  0.0      2.0                0.0
4                 0.0                  0.0      0.0                0.0

   country_cat_Central African Republic  country_cat_Chad  \
0                                   0.0               0.0
1                                   0.0               0.0
2                                   0.0               0.0
3                                   0.0               0.0
4                                   0.0               0.0

   country_cat_DR Congo  country_cat_Dominican Republic  …  \
0                   0.0                             0.0  …
1                   0.0                             0.0  …
2                   0.0                             0.0  …
```

```
3                    0.0                              0.0  …
4                    0.0                              0.0  …

   WS1_g_Tube/Well/Borehole  WS1_g_Unprotected well/spring  water_treatment_1  \
0                       1.0                            0.0                0.0
1                       1.0                            0.0                0.0
2                       1.0                            0.0                0.0
3                       0.0                            1.0                0.0
4                       1.0                            0.0                0.0

   windex_ur  helevel  urban  wq27_decile  RiskSource  VeryHighRiskHome  \
0          2        0      0            7           1                 0
1          2        0      0            1           0                 0
2          2        0      0            8           2                 1
3          2        0      0            8           2                 1
4          1        0      0            8           1                 0

   SomeRiskHome
0             1
1             1
2             1
3             1
4             1

[5 rows x 42 columns]
```

2 Binary treatment

2.1 Outcome: VeryHighRiskHome

[7]:
```python
# Define outcome, treatment, and controls
binary_y = "VeryHighRiskHome"
binary_d = ["water_treatment_1"]
binary_x = [col for col in mics.columns if col not in [binary_y,␣
 ↪"SomeRiskHome"] + binary_d]

# Build DoubleML data object
binary_data_vhr = DoubleMLData(
    data=mics,
    y_col=binary_y,
    d_cols=binary_d,
    x_cols=binary_x,
)

# Base learners
ml_l_xgb = XGBClassifier(
    use_label_encoder=False,
```

```python
    objective="binary:logistic",
    eval_metric="logloss",
    eta=0.1,
    n_estimators=34,
)

ml_m_xgb = XGBClassifier(
    use_label_encoder=False,
    objective="binary:logistic",
    eval_metric="logloss",
    eta=0.1,
    n_estimators=34,
)

# Double machine learning model
binary_model_vhr = DoubleMLPLR(
    binary_data_vhr,
    ml_l=ml_l_xgb,
    ml_m=ml_m_xgb,
)
```

[8]:

```python
# Hyperparameter search with Optuna (keeps the same space as original)
def ml_l_params(trial):
    return {
        "n_estimators": trial.suggest_int("n_estimators", 50, 200, step=50),
        "max_depth": trial.suggest_int("max_depth", 3, 10),
        "min_child_weight": trial.suggest_int("min_child_weight", 1, 20),
    }


def ml_m_params(trial):
    return {
        "n_estimators": trial.suggest_int("n_estimators", 50, 200, step=50),
        "max_depth": trial.suggest_int("max_depth", 3, 10),
        "min_child_weight": trial.suggest_int("min_child_weight", 1, 20),
    }


param_space = {"ml_l": ml_l_params, "ml_m": ml_m_params}
optuna_settings = {
    "n_trials": 100,
    "show_progress_bar": True,
    "verbosity": optuna.logging.WARNING,
}

binary_model_vhr.tune_ml_models(
    ml_param_space=param_space,
```

```
    optuna_settings=optuna_settings,
)
```

```
  0%|          | 0/100 [00:00<?, ?it/s]

  0%|          | 0/100 [00:00<?, ?it/s]
```

[8]:
```
<doubleml.plm.plr.DoubleMLPLR at 0x2201570d160>
```

[9]:
```
# Fit and summarize
binary_model_vhr.fit()
binary_model_vhr.summary
```

[9]:
```
                       coef   std err         t     P>|t|     2.5 %    97.5 %
water_treatment_1 -0.248442  0.148278 -1.675508  0.093835 -0.539062  0.042179
```

[10]:
```
# Group-wise treatment effects (GATE)
groups = pd.DataFrame({
    "Education level": mics["helevel"].map({0: "No education", 1: "Primary", 2:␣
  ↪"Secondary or higher"}),
})
binary_model_vhr.gate(groups=groups).summary
```

[10]:
```
                                 coef   std err         t         P>|t|  \
Group_No education           0.035164  0.014908  2.358706  1.833877e-02
Group_Primary               -0.464279  0.204036 -2.275477  2.287730e-02
Group_Secondary or higher -0.100706  0.015403 -6.538222  6.225440e-11

                             [0.025    0.975]
Group_No education           0.005945  0.064384
Group_Primary               -0.864181 -0.064376
Group_Secondary or higher -0.130895 -0.070518
```

[11]:
```
# Group-wise treatment effects by area
groups = pd.DataFrame({
    "Area": mics["urban"].map({0: "Rural", 1: "Urban"}),
})
binary_model_vhr.gate(groups=groups).summary
```

[11]:
```
                 coef   std err        t     P>|t|    [0.025    0.975]
Group_Rural -0.074243  0.070411 -1.05442  0.291691 -0.212245  0.063760
Group_Urban -0.422782  0.228160 -1.85301  0.063881 -0.869967  0.024403
```

[12]:
```
# Group-wise treatment effects by wealth index
groups = pd.DataFrame({
    "Wealth Index": mics["windex5"].map({
        0: "Poorest",
        1: "Poor",
        2: "Middle",
```

```
        3: "Rich",
        4: "Richest",
    }),
})
binary_model_vhr.gate(groups=groups).summary
```

[12]:

```
                    coef     std err          t          P>|t|      [0.025  \
Group_Middle  -0.748184    0.011650 -64.222019  0.000000e+00  -0.771018
Group_Poor     0.020821    0.014088   1.477948  1.394218e-01  -0.006790
Group_Poorest  3.038094   11.077811   0.274250  7.838922e-01 -18.674016
Group_Rich    -0.166478    0.010998 -15.137308  9.189958e-52  -0.188033
Group_Richest -0.104436    0.011130  -9.383241  6.397560e-21  -0.126250

                  0.975]
Group_Middle   -0.725351
Group_Poor      0.048432
Group_Poorest  24.750204
Group_Rich     -0.144922
Group_Richest  -0.082621
```

3 Binary treatment

3.1 Outcome: SomeRiskHome

[13]:

```
# Define outcome, treatment, and controls for the alternative outcome
binary_y = "SomeRiskHome"
binary_d = ["water_treatment_1"]
binary_x = [col for col in mics.columns if col not in [binary_y,␣
 ↪"VeryHighRiskHome"] + binary_d]

binary_data_some = DoubleMLData(
    data=mics,
    y_col=binary_y,
    d_cols=binary_d,
    x_cols=binary_x,
)

ml_l_xgb = XGBClassifier(
    use_label_encoder=False,
    objective="binary:logistic",
    eval_metric="logloss",
    eta=0.1,
    n_estimators=34,
)

ml_m_xgb = XGBClassifier(
    use_label_encoder=False,
```

```
    objective="binary:logistic",
    eval_metric="logloss",
    eta=0.1,
    n_estimators=34,
)

binary_model_some = DoubleMLPLR(
    binary_data_some,
    ml_l=ml_l_xgb,
    ml_m=ml_m_xgb,
)
```

[14]:
```
# Hyperparameter search for the alternative outcome

def ml_l_params(trial):
    return {
        "n_estimators": trial.suggest_int("n_estimators", 50, 200, step=50),
        "max_depth": trial.suggest_int("max_depth", 3, 10),
        "min_child_weight": trial.suggest_int("min_child_weight", 1, 20),
    }


def ml_m_params(trial):
    return {
        "n_estimators": trial.suggest_int("n_estimators", 50, 200, step=50),
        "max_depth": trial.suggest_int("max_depth", 3, 10),
        "min_child_weight": trial.suggest_int("min_child_weight", 1, 20),
    }


param_space = {"ml_l": ml_l_params, "ml_m": ml_m_params}
optuna_settings = {
    "n_trials": 100,
    "show_progress_bar": True,
    "verbosity": optuna.logging.WARNING,
}

binary_model_some.tune_ml_models(
    ml_param_space=param_space,
    optuna_settings=optuna_settings,
)
```

```
  0%|          | 0/100 [00:00<?, ?it/s]

  0%|          | 0/100 [00:00<?, ?it/s]
```

[14]:
```
<doubleml.plm.plr.DoubleMLPLR at 0x220155342d0>
```

[15]:
```python
# Fit and summarize the alternative outcome
binary_model_some.fit()
binary_model_some.summary
```

[15]:
```
                       coef   std err         t     P>|t|     2.5 %    97.5 %
water_treatment_1 -0.139592  0.105387 -1.324558  0.185318 -0.346148  0.066964
```

[16]:
```python
# Group-wise treatment effects (GATE) for the alternative outcome
groups = pd.DataFrame({
    "Education level": mics["helevel"].map({0: "No education", 1: "Primary", 2:␣
 ↪"Secondary or higher"}),
})
binary_model_some.gate(groups=groups).summary
```

[16]:
```
                               coef   std err          t         P>|t|  \
Group_No education         0.000242  0.009657   0.025018  9.800409e-01
Group_Primary             -0.093097  0.159811  -0.582546  5.601987e-01
Group_Secondary or higher -0.372114  0.020548 -18.109089  2.701827e-73

                             [0.025    0.975]
Group_No education        -0.018686  0.019169
Group_Primary             -0.406320  0.220126
Group_Secondary or higher -0.412388 -0.331840
```

[17]:
```python
# Group-wise treatment effects by area for the alternative outcome
groups = pd.DataFrame({
    "Area": mics["urban"].map({0: "Rural", 1: "Urban"}),
})
binary_model_some.gate(groups=groups).summary
```

[17]:
```
                 coef   std err          t         P>|t|    [0.025    0.975]
Group_Rural  0.059870  0.050783   1.178948  2.384190e-01 -0.039662  0.159402
Group_Urban -0.339227  0.020104 -16.874011  6.988543e-64 -0.378629 -0.299825
```

[18]:
```python
# Group-wise treatment effects by wealth index for the alternative outcome
groups = pd.DataFrame({
    "Wealth Index": mics["windex5"].map({
        0: "Poorest",
        1: "Poor",
        2: "Middle",
        3: "Rich",
        4: "Richest",
    }),
})
binary_model_some.gate(groups=groups).summary
```

```
[18]:                       coef   std err          t           P>|t|     [0.025  \
Group_Middle  -0.321220  0.015149 -21.203906  8.789055e-100  -0.350912
Group_Poor    -0.011082  0.016848  -0.657728   5.107129e-01  -0.044104
Group_Poorest  1.553460  9.366168   0.165859   8.682682e-01 -16.803893
Group_Rich     0.131654  0.014652   8.985360   2.578892e-19   0.102937
Group_Richest -0.360039  0.012002 -29.997618  1.054111e-197  -0.383563

                  0.975]
Group_Middle   -0.291529
Group_Poor      0.021941
Group_Poorest  19.910812
Group_Rich      0.160372
Group_Richest  -0.336515
```

```python
[19]: # Group-wise treatment effects by education + area + wealth for the alternative
      ↪outcome
groups = pd.DataFrame({
    "Edu_Area": (
        mics["helevel"].map({0: "No education", 1: "Primary", 2: "Secondary or
 ↪higher"})
        + " | "
        + mics["urban"].map({0: "Rural", 1: "Urban"})
        + " | "
        + mics["windex5"].map({
            0: "Poorest",
            1: "Poor",
            2: "Middle",
            3: "Rich",
            4: "Richest",
        })
    )
})
binary_model_some.gate(groups=groups).summary
```

```
[19]:                                              coef    std err          t  \
Group_No education | Rural | Middle          0.059931  27.652980   0.002167
Group_No education | Rural | Poor           -0.000273   0.004666  -0.058472
Group_No education | Rural | Poorest        10.122289  13.580495   0.745355
Group_No education | Rural | Rich          -24.772336  38.151129  -0.649321
Group_No education | Rural | Richest       -11.648724  71.623680  -0.162638
Group_No education | Urban | Middle        -43.942685  39.582950  -1.110142
Group_No education | Urban | Poor           15.165182  52.451670   0.289127
Group_No education | Urban | Poorest        32.844291  35.022613   0.937802
Group_No education | Urban | Rich           12.375643  38.459307   0.321785
Group_No education | Urban | Richest       -23.390634  52.405117  -0.446343
Group_Primary | Rural | Middle              -9.164356  20.635857  -0.444099
Group_Primary | Rural | Poor               -27.377695  19.662157  -1.392405
```

```
Group_Primary | Rural | Poorest            -14.224291  16.377901  -0.868505
Group_Primary | Rural | Rich                 0.136777   0.006168  22.173909
Group_Primary | Rural | Richest             85.000374  50.681169   1.677159
Group_Primary | Urban | Middle              -0.312054   0.004663 -66.919940
Group_Primary | Urban | Poor                -6.598315  26.224116  -0.251612
Group_Primary | Urban | Poorest             29.590374  51.643811   0.572970
Group_Primary | Urban | Rich                23.338462  29.215151   0.798848
Group_Primary | Urban | Richest            -28.392376  37.885314  -0.749430
Group_Secondary or higher | Rural | Middle    8.460466  23.703972   0.356922
Group_Secondary or higher | Rural | Poor    -10.472962  22.843970  -0.458456
Group_Secondary or higher | Rural | Poorest  13.304440  21.124707   0.629805
Group_Secondary or higher | Rural | Rich    -17.749751  26.453341  -0.670983
Group_Secondary or higher | Rural | Richest -38.621456  34.670126  -1.113969
Group_Secondary or higher | Urban | Middle  -38.918664  27.840716  -1.397905
Group_Secondary or higher | Urban | Poor     10.196893  30.832675   0.330717
Group_Secondary or higher | Urban | Poorest -38.480445  43.296654  -0.888763
Group_Secondary or higher | Urban | Rich    -17.829522  24.771564  -0.719758
Group_Secondary or higher | Urban | Richest  -0.355439   0.011138 -31.913112

                                                   P>|t|      [0.025  \
Group_No education | Rural | Middle          9.982708e-01  -54.138914
Group_No education | Rural | Poor            9.533724e-01   -0.009418
Group_No education | Rural | Poorest         4.560572e-01  -16.494993
Group_No education | Rural | Rich            5.161308e-01  -99.547175
Group_No education | Rural | Richest         8.708036e-01 -152.028558
Group_No education | Urban | Middle          2.669380e-01 -121.523842
Group_No education | Urban | Poor            7.724844e-01  -87.638203
Group_No education | Urban | Poorest         3.483460e-01  -35.798769
Group_No education | Urban | Rich            7.476153e-01  -63.003214
Group_No education | Urban | Richest         6.553498e-01 -126.102775
Group_Primary | Rural | Middle               6.569713e-01  -49.609893
Group_Primary | Rural | Poor                 1.637996e-01  -65.914814
Group_Primary | Rural | Poorest              3.851179e-01  -46.324387
Group_Primary | Rural | Rich                6.134460e-109    0.124688
Group_Primary | Rural | Richest              9.351141e-02  -14.332891
Group_Primary | Urban | Middle               0.000000e+00   -0.321193
Group_Primary | Urban | Poor                 8.013406e-01  -57.996638
Group_Primary | Urban | Poorest              5.666647e-01  -71.629635
Group_Primary | Urban | Rich                 4.243786e-01  -33.922183
Group_Primary | Urban | Richest              4.535983e-01 -102.646226
Group_Secondary or higher | Rural | Middle    7.211503e-01  -37.998464
Group_Secondary or higher | Rural | Poor      6.466246e-01  -55.246321
Group_Secondary or higher | Rural | Poorest   5.288224e-01  -28.099224
Group_Secondary or higher | Rural | Rich      5.022312e-01  -69.597347
Group_Secondary or higher | Rural | Richest   2.652924e-01 -106.573654
Group_Secondary or higher | Urban | Middle    1.621417e-01  -93.485464
Group_Secondary or higher | Urban | Poor      7.408582e-01  -50.234039
```

```
Group_Secondary or higher | Urban | Poorest   3.741307e-01 -123.340328
Group_Secondary or higher | Urban | Rich      4.716742e-01  -66.380895
Group_Secondary or higher | Urban | Richest  1.756497e-223   -0.377268

                                                 0.975]
Group_No education | Rural | Middle          54.258775
Group_No education | Rural | Poor             0.008872
Group_No education | Rural | Poorest         36.739570
Group_No education | Rural | Rich            50.002504
Group_No education | Rural | Richest        128.731110
Group_No education | Urban | Middle          33.638471
Group_No education | Urban | Poor           117.968567
Group_No education | Urban | Poorest        101.487351
Group_No education | Urban | Rich            87.754500
Group_No education | Urban | Richest         79.321507
Group_Primary | Rural | Middle               31.281181
Group_Primary | Rural | Poor                 11.159424
Group_Primary | Rural | Poorest              17.875806
Group_Primary | Rural | Rich                  0.148867
Group_Primary | Rural | Richest             184.333639
Group_Primary | Urban | Middle               -0.302914
Group_Primary | Urban | Poor                 44.800009
Group_Primary | Urban | Poorest             130.810383
Group_Primary | Urban | Rich                 80.599107
Group_Primary | Urban | Richest              45.861475
Group_Secondary or higher | Rural | Middle    54.919397
Group_Secondary or higher | Rural | Poor      34.300396
Group_Secondary or higher | Rural | Poorest   54.708105
Group_Secondary or higher | Rural | Rich      34.097846
Group_Secondary or higher | Rural | Richest   29.330742
Group_Secondary or higher | Urban | Middle    15.648135
Group_Secondary or higher | Urban | Poor      70.627826
Group_Secondary or higher | Urban | Poorest   46.379438
Group_Secondary or higher | Urban | Rich      30.721851
Group_Secondary or higher | Urban | Richest   -0.333609
```

4 Multinomial treatment

4.1 Outcome: VeryHighRiskHome

[20]:
```python
# Define multinomial treatment columns (one column per category)
multi_y = "VeryHighRiskHome"
multi_d = [col for col in mics.columns if col.startswith("WS1_g_")]
# Drop the binary treatment from controls to avoid duplication
multi_x = [col for col in mics.columns if col not in [multi_y, "SomeRiskHome"]␣
↪+ multi_d + ["water_treatment_1"]]
```

```python
multi_data_vhr = DoubleMLData(
    data=mics,
    y_col=multi_y,
    d_cols=multi_d,
    x_cols=multi_x,
)

ml_l_xgb = XGBClassifier(
    use_label_encoder=False,
    objective="binary:logistic",
    eval_metric="logloss",
    eta=0.1,
    n_estimators=34,
    n_jobs=-1,
)

ml_m_xgb = XGBClassifier(
    use_label_encoder=False,
    objective="multi:softprob",
    eval_metric="mlogloss",
    num_class=len(multi_d),
    eta=0.1,
    n_estimators=34,
    n_jobs=-1,
)

multi_model_vhr = DoubleMLPLR(
    multi_data_vhr,
    ml_l=ml_l_xgb,
    ml_m=ml_m_xgb,
)
```

[21]:

```python
# Optional hyperparameter search (commented to save time)
# def ml_l_params(trial):
#     return {
#         "n_estimators": trial.suggest_int("n_estimators", 50, 200, step=50),
#         "max_depth": trial.suggest_int("max_depth", 3, 10),
#     }
#
#
# def ml_m_params(trial):
#     return {
#         "n_estimators": trial.suggest_int("n_estimators", 50, 200, step=50),
#         "max_depth": trial.suggest_int("max_depth", 3, 10),
#     }
#
# param_space = {"ml_l": ml_l_params, "ml_m": ml_m_params}
```

```python
# optuna_settings = {
#     "n_trials": 100,
#     "show_progress_bar": True,
#     "verbosity": optuna.logging.WARNING,
# }
#
# multi_model_vhr.tune_ml_models(
#     ml_param_space=param_space,
#     optuna_settings=optuna_settings,
# )
```

[22]:
```python
# Fit and summarize
multi_model_vhr.fit()
multi_model_vhr.summary
```

[22]:
```
                                   coef   std err         t         P>|t|  \
WS1_g_Packaged/Bottled water  -0.070180  0.015489 -4.530985  5.870924e-06
WS1_g_Piped water             -0.005642  0.013700 -0.411848  6.804507e-01
WS1_g_Protected well/spring   -0.006038  0.015199 -0.397259  6.911763e-01
WS1_g_Surface/Rain water       0.026649  0.015521  1.717033  8.597313e-02
WS1_g_Tube/Well/Borehole       0.097758  0.013590  7.193605  6.310260e-13
WS1_g_Unprotected well/spring  0.036852  0.017154  2.148323  3.168814e-02

                                  2.5 %    97.5 %
WS1_g_Packaged/Bottled water  -0.100538 -0.039823
WS1_g_Piped water             -0.032493  0.021209
WS1_g_Protected well/spring   -0.035827  0.023751
WS1_g_Surface/Rain water      -0.003770  0.057069
WS1_g_Tube/Well/Borehole       0.071123  0.124392
WS1_g_Unprotected well/spring  0.003231  0.070473
```

[23]:
```python
# Group-wise treatment effects (GATE) for multinomial treatment
groups = pd.DataFrame({
    "Education level": mics["helevel"].map({0: "No education", 1: "Primary", 2:␣
↪"Secondary or higher"}),
})
multi_model_vhr.gate(groups=groups).summary
```

```
---------------------------------------------------------------------------
NotImplementedError                       Traceback (most recent call last)
Cell In[23], line 5
      1 # Group-wise treatment effects (GATE) for multinomial treatment
      2 groups = pd.DataFrame({
      3     "Education level": mics["helevel"].map({0: "No education", 1:␣
↪"Primary", 2: "Secondary or higher"}),
      4 })
----> 5 multi_model_vhr.gate(groups=groups).summary
```

```
File c:\Users\jadrk\Dropbox\Colab Notebooks\MICS\.
↪venv\Lib\site-packages\doubleml\plm\plr.py:521, in DoubleMLPLR.gate(self,␣
↪groups, **kwargs)
    518 if any(groups.sum(0) <= 5):
    519     warnings.warn("At least one group effect is estimated with less tha␣
↪6 observations.")
--> 521 model = self.cate(groups, is_gate=True, **kwargs)
    522 return model

File c:\Users\jadrk\Dropbox\Colab Notebooks\MICS\.
↪venv\Lib\site-packages\doubleml\plm\plr.py:470, in DoubleMLPLR.cate(self,␣
↪basis, is_gate, **kwargs)
    448 """
    449 Calculate conditional average treatment effects (CATE) for a given basi.
    450
   (…)    467     Best linear Predictor model.
    468 """
    469 if self._dml_data.n_treat > 1:
--> 470     raise NotImplementedError(
    471         "Only implemented for single treatment. " + f"Number of␣
↪treatments is {str(self._dml_data.n_treat)}."
    472     )
    473 if self.n_rep != 1:
    474     raise NotImplementedError("Only implemented for one repetition. " +␣
↪f"Number of repetitions is {str(self.n_rep)}.")

NotImplementedError: Only implemented for single treatment. Number of treatment␣
↪is 6.
```

[ ]:
```
# Group-wise treatment effects by area
groups = pd.DataFrame({
    "Area": mics["urban"].map({0: "Rural", 1: "Urban"}),
})
multi_model_vhr.gate(groups=groups).summary
```

[ ]:
```
# Group-wise treatment effects by wealth index
groups = pd.DataFrame({
    "Wealth Index": mics["windex5"].map({
        0: "Poorest",
        1: "Poor",
        2: "Middle",
        3: "Rich",
        4: "Richest",
    }),
})
multi_model_vhr.gate(groups=groups).summary
```

```
[ ]: # Group-wise treatment effects by education + area + wealth
groups = pd.DataFrame({
    "Edu_Area": (
        mics["helevel"].map({0: "No education", 1: "Primary", 2: "Secondary or␣
 ↪higher"})
        + " | "
        + mics["urban"].map({0: "Rural", 1: "Urban"})
        + " | "
        + mics["windex5"].map({
            0: "Poorest",
            1: "Poor",
            2: "Middle",
            3: "Rich",
            4: "Richest",
        })
    )
})
multi_model_vhr.gate(groups=groups).summary
```

5 Multinomial treatment

5.1 Outcome: SomeRiskHome

```
[ ]: multi_y = "SomeRiskHome"
multi_d = [col for col in mics.columns if col.startswith("WS1_g_")]
multi_x = [col for col in mics.columns if col not in [multi_y,␣
 ↪"VeryHighRiskHome"] + multi_d + ["water_treatment_1"]]

multi_data_some = DoubleMLData(
    data=mics,
    y_col=multi_y,
    d_cols=multi_d,
    x_cols=multi_x,
)

ml_l_xgb = XGBClassifier(
    use_label_encoder=False,
    objective="binary:logistic",
    eval_metric="logloss",
    eta=0.1,
    n_estimators=34,
    n_jobs=-1,
)

ml_m_xgb = XGBClassifier(
    use_label_encoder=False,
    objective="multi:softprob",
```

```
    eval_metric="mlogloss",
    num_class=len(multi_d),
    eta=0.1,
    n_estimators=34,
    n_jobs=-1,
)

multi_model_some = DoubleMLPLR(
    multi_data_some,
    ml_l=ml_l_xgb,
    ml_m=ml_m_xgb,
)
```

[ ]:
```
# Optional hyperparameter search (commented to save time)
# def ml_l_params(trial):
#     return {
#         "n_estimators": trial.suggest_int("n_estimators", 50, 200, step=50),
#         "max_depth": trial.suggest_int("max_depth", 3, 10),
#     }
#
#
# def ml_m_params(trial):
#     return {
#         "n_estimators": trial.suggest_int("n_estimators", 50, 200, step=50),
#         "max_depth": trial.suggest_int("max_depth", 3, 10),
#     }
#
# param_space = {"ml_l": ml_l_params, "ml_m": ml_m_params}
# optuna_settings = {
#     "n_trials": 100,
#     "show_progress_bar": True,
#     "verbosity": optuna.logging.WARNING,
# }
#
# multi_model_some.tune_ml_models(
#     ml_param_space=param_space,
#     optuna_settings=optuna_settings,
# )
```

[ ]:
```
# Fit and summarize
multi_model_some.fit()
multi_model_some.summary
```

[ ]:
```
# Group-wise treatment effects (GATE) for multinomial treatment
groups = pd.DataFrame({
    "Education level": mics["helevel"].map({0: "No education", 1: "Primary", 2:␣
 ↪"Secondary or higher"}),
```

```
})
multi_model_some.gate(groups=groups).summary
```

[ ]:
```
# Group-wise treatment effects by area
groups = pd.DataFrame({
    "Area": mics["urban"].map({0: "Rural", 1: "Urban"}),
})
multi_model_some.gate(groups=groups).summary
```

[ ]:
```
# Group-wise treatment effects by wealth index
groups = pd.DataFrame({
    "Wealth Index": mics["windex5"].map({
        0: "Poorest",
        1: "Poor",
        2: "Middle",
        3: "Rich",
        4: "Richest",
    }),
})
multi_model_some.gate(groups=groups).summary
```

[ ]:
```
# Group-wise treatment effects by education + area + wealth
groups = pd.DataFrame({
    "Edu_Area": (
        mics["helevel"].map({0: "No education", 1: "Primary", 2: "Secondary or␣
↪higher"})
        + " | "
        + mics["urban"].map({0: "Rural", 1: "Urban"})
        + " | "
        + mics["windex5"].map({
            0: "Poorest",
            1: "Poor",
            2: "Middle",
            3: "Rich",
            4: "Richest",
        })
    )
})
multi_model_some.gate(groups=groups).summary
```