Case Study: Effect of Sodium Intake on Blood Pressure

Motivation:

- 63% of Americans aged over 60 have high blood pressure (>=140mmHg)
- 85% of Americans aged over 50 consume more than 2.3g sodium/day
- federal recommendation: less than 2.3g sodium/day

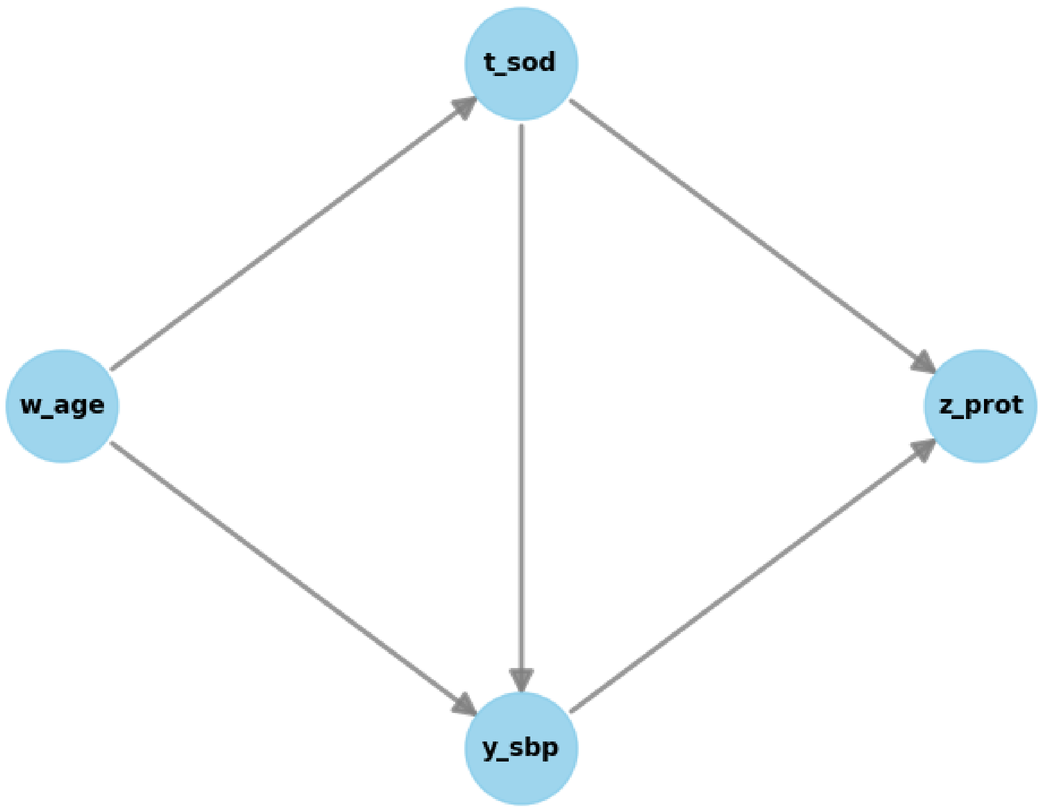
Data:

- (simulated) epidemiological example taken from Luque-Fernandez et al. (2018)
    - we corrected the real-world numbers
- Outcome Y: (systolic) blood pressure
- Treatment T: sodium intake
- Covariates
    - W age
    - Z amount of protein excreted in urine

Variables

var	type	desc
w_age	covariate	Age (years)
z_prot	covariate	24-hour excretion of urinary protein (proteinuria) (mg) (□□ Proteinurie)
t_sod	treatment	24-hour dietary sodium intake (g)
y_sbp	outcome	Systolic blood pressure (mmHg)

Causal Mechanisms



In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

def generate_data(n, seed):
    rng = np.random.default_rng(seed)
    # structural equations
    age = rng.normal(loc=65, scale=5, size=n) # [years]
    sodium = age / 18 + rng.normal(0, 1, n) # [gramm]
    sbp = 1.05 * sodium + 2.15 * age + rng.normal(0, 1, n) # [mmHg] # fixed: 2.0 ->
    prot = 2.00 * sbp + 2.80 * sodium + rng.normal(0, 1, n) # [mg]
    hypertension = np.where(sbp > 140, 1, 0) # 1 where sbp > 140 mmHg
    sodium_upperlimit = np.where(sodium >= 2.3, 1, 0) # 1 where sodium intake >= 2.
    data = pd.DataFrame({
        'y_sbp': sbp,
        't_sod': sodium,
        'w_age': age,
        'z_prot': prot,
        'hypertension': hypertension,
        'sodium_upperlimit': sodium_upperlimit
    })
    return data

sbp_sod_age_prot = ["y_sbp", "t_sod", "w_age", "z_prot"]
data = generate_data(n=1000, seed=111)
data
```

Out[1]:

	y_sbp	t_sod	w_age	z_prot	hypertension	sodium_upperlimit
0	140.695443	3.204703	63.462811	289.814072	1	1
1	133.235319	3.756593	60.806716	276.262146	0	1
2	144.788758	3.289475	65.628189	299.751282	1	1
3	136.606249	4.238763	61.696614	284.298170	0	1
4	143.396119	3.611865	65.791576	296.846522	1	1
...	...	...	...	...	...	...
995	125.437922	2.944481	56.470776	258.749576	0	1
996	156.403686	3.670764	71.434463	321.752170	1	1
997	131.840275	2.942233	59.413070	271.213898	0	1
998	145.518012	3.400475	65.431605	299.667019	1	1
999	167.433112	4.993656	74.928183	347.609654	1	1

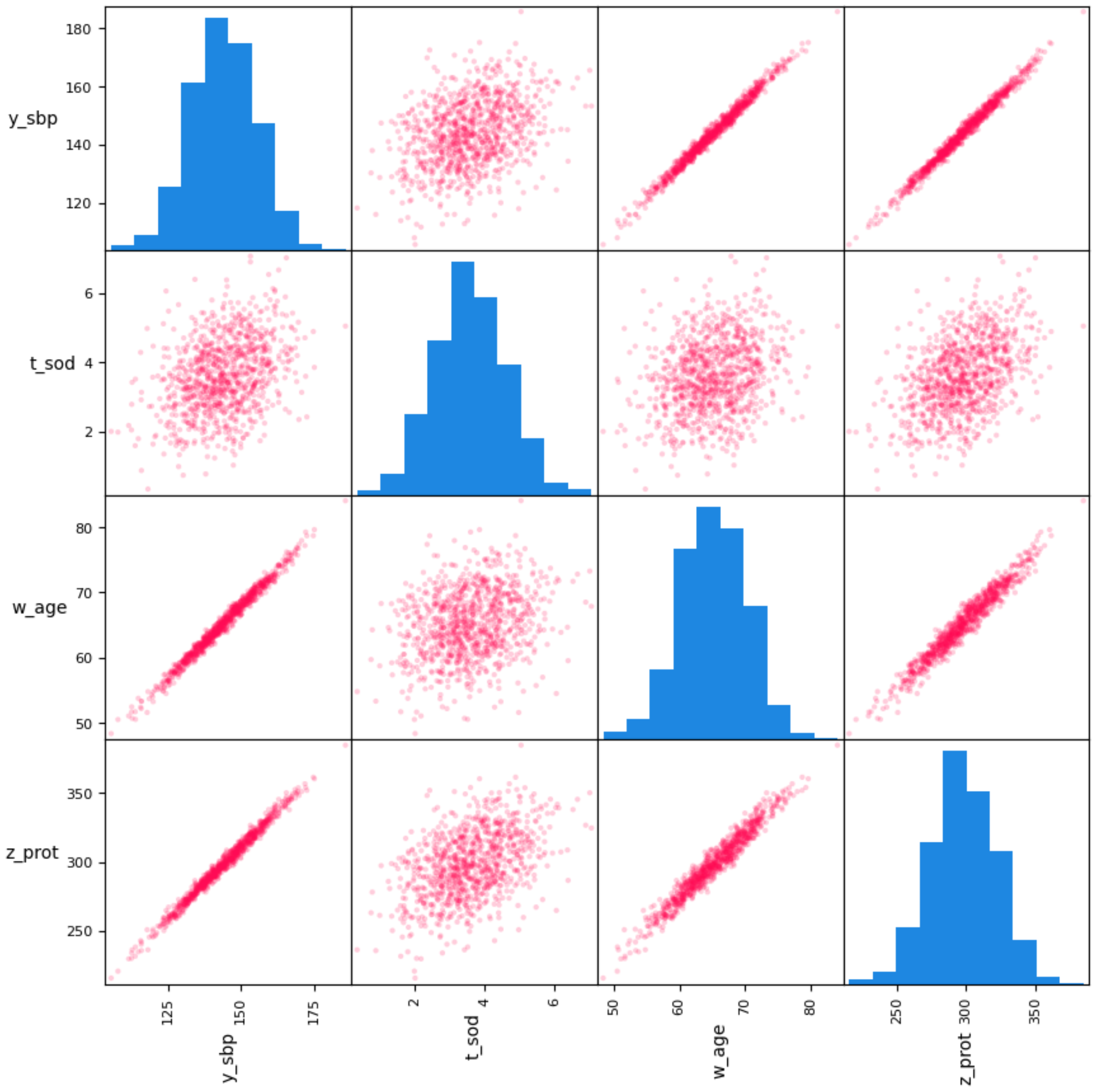
1000 rows × 6 columns

In [2]:
```
data.describe()
```

Out[2]:

	y_sbp	t_sod	w_age	z_prot	hypertension	sodium_upp
count	1000.000000	1000.000000	1000.000000	1000.000000	1000.000000	1000.
mean	143.983373	3.604801	65.191883	297.999718	0.638000	0.
std	11.300196	1.038661	5.083829	23.757908	0.480819	0.
min	105.668916	0.343651	48.384404	215.789250	0.000000	0.
25%	136.552744	2.878595	61.799655	282.579498	0.000000	1.
50%	143.949487	3.561624	65.081524	297.213762	1.000000	1.
75%	151.929276	4.323143	68.760501	314.667970	1.000000	1.
max	185.729127	7.068915	84.078218	384.563068	1.000000	1.

In [3]:
```
axes = pd.plotting.scatter_matrix(data[sbp_sod_age_prot], figsize=(10, 10), c='#ff0
for ax in axes.flatten():
    ax.xaxis.label.set_rotation(90)
    ax.yaxis.label.set_rotation(0)
    ax.yaxis.label.set_ha('right')
```



Linear Regression

Model 0: Systolic Blood Pressure in mmHg = β_0 + $\beta_1 \times$ Sodium in g + ε

Model 1: Systolic Blood Pressure in mmHg = β_0 + $\beta_1 \times$ Sodium in g + $\beta_2 \times$Age + ε

Model 2: Systolic Blood Pressure in mmHg = β_0 + $\beta_1 \times$ Sodium in g + $\beta_2 \times$Age + $\beta_3 \times$ Proteinuria in mg + ε

In [4]:
```
# https://www.statsmodels.org/devel/examples/notebooks/generated/ols.html
import statsmodels.api as sm
from statsmodels.formula.api import ols
from scipy.stats import norm
```

In [5]:
```
# Fit the linear regression model
fit0 = ols("y_sbp ~ t_sod", data).fit()
```

```
fit1 = ols("y_sbp ~ t_sod + w_age", data).fit()
fit2 = ols("y_sbp ~ t_sod + w_age + z_prot", data).fit()
```

In [6]:
```
fit0.params
```

Out[6]:
```
Intercept    130.584035
t_sod          3.717082
dtype: float64
```

In [7]:
```
fit1.params
```

Out[7]:
```
Intercept   -0.009662
t_sod        1.058473
w_age        2.150229
dtype: float64
```

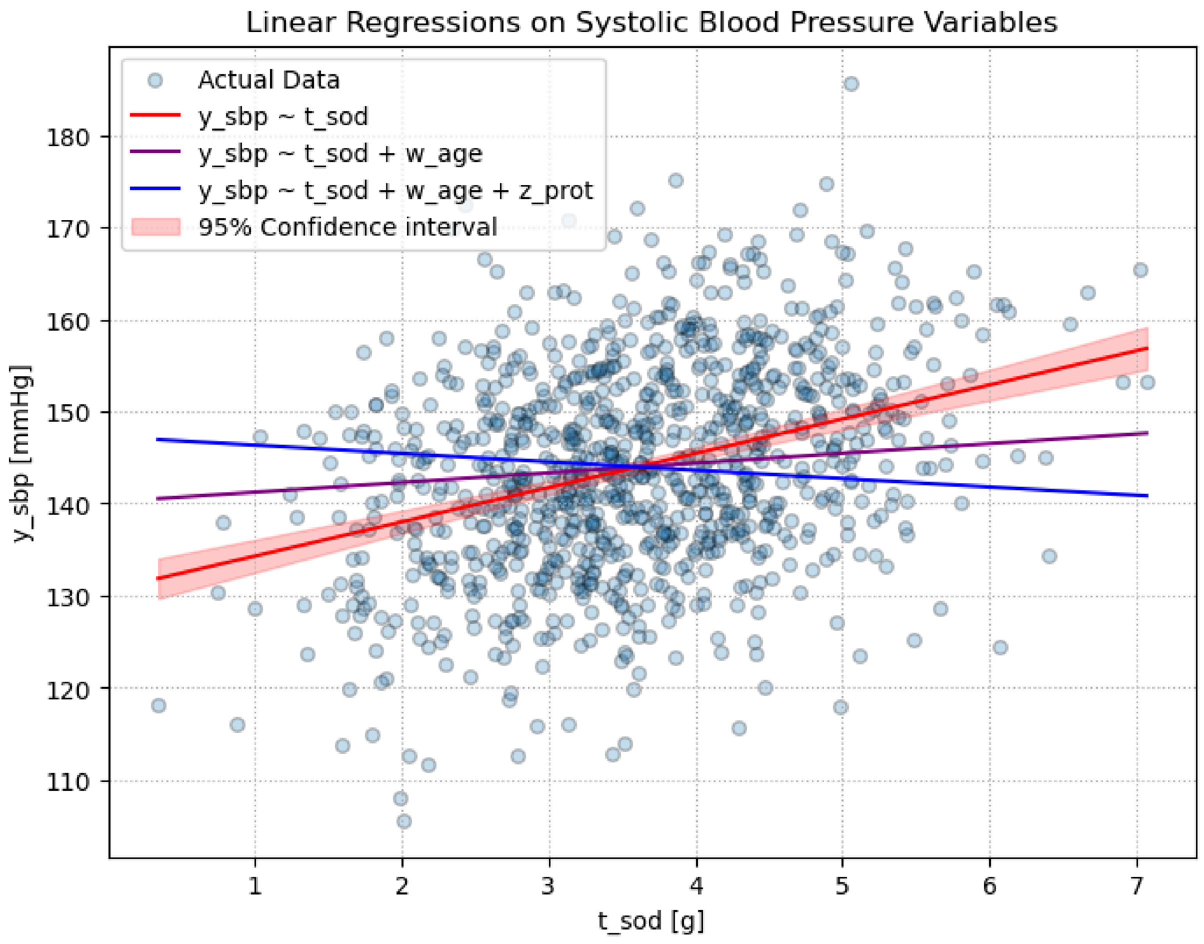
In [8]:
```
fit2.params
```

Out[8]:
```
Intercept   -0.022544
t_sod       -0.910760
w_age        0.422265
z_prot       0.401882
dtype: float64
```

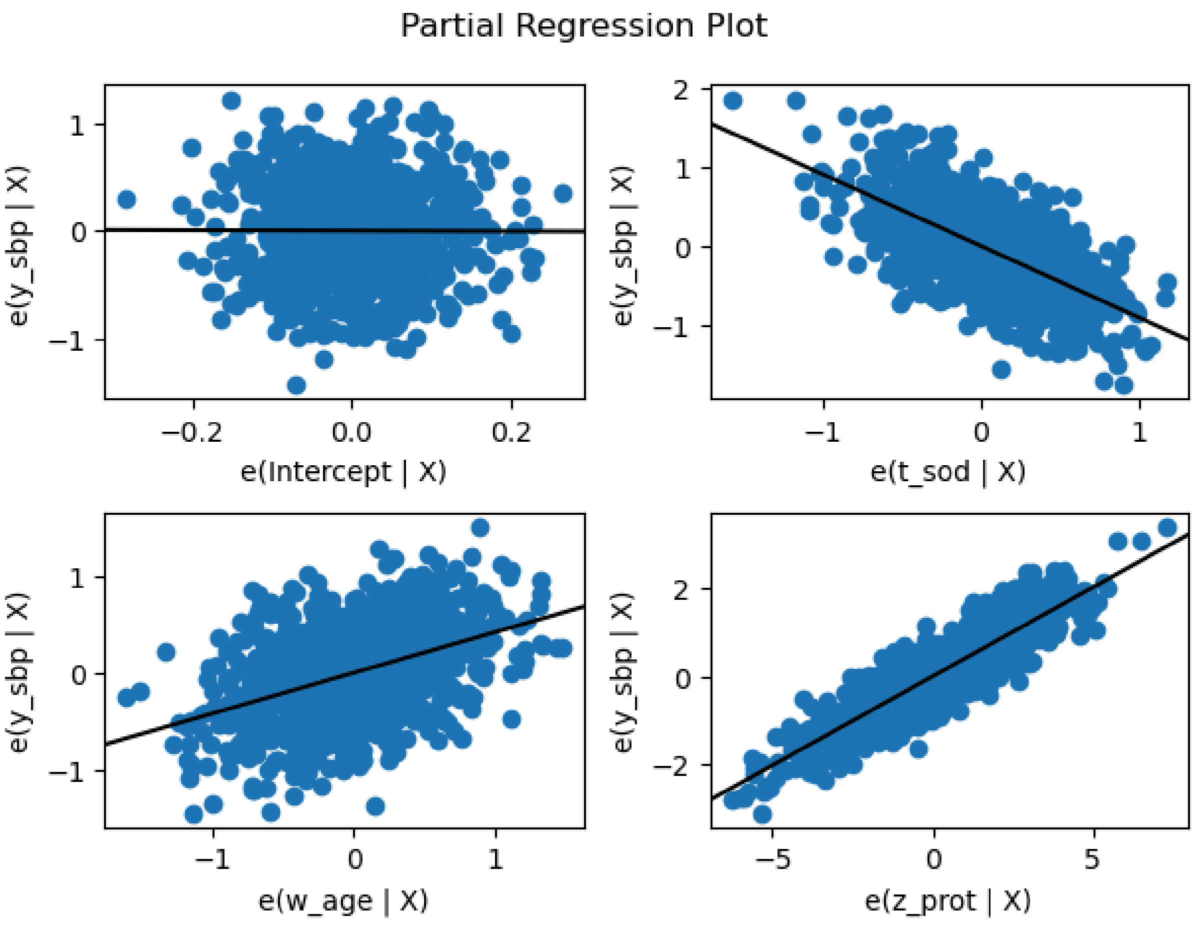
In [9]:
```
xvalues = data.t_sod
x_line = np.linspace(xvalues.min(), xvalues.max(), 100)
y = data.y_sbp

# Create a line with the regression coefficients
coeff = fit0.params
y_line0 = coeff.Intercept + coeff.t_sod * x_line

coeff = fit1.params
y_line1 = coeff.Intercept + coeff.t_sod * x_line + coeff.w_age * data.w_age.mean()

coeff = fit2.params
y_line2 = coeff.Intercept + coeff.t_sod * x_line + coeff.w_age * data.w_age.mean()
```

In [10]:
```
fig, ax = plt.subplots(figsize=(8, 6))
# Create a scatter plot of the actual data
ax.scatter(xvalues, y, label='Actual Data', alpha=0.25, s=30, edgecolors='k')
# Plot the regression lines
ax.plot(x_line, y_line0, color='red', label='y_sbp ~ t_sod')
ax.plot(x_line, y_line1, color='purple', label='y_sbp ~ t_sod + w_age')
ax.plot(x_line, y_line2, color='blue', label='y_sbp ~ t_sod + w_age + z_prot')

# Get the confidence interval
conf_int = fit0.get_prediction(pd.DataFrame({'t_sod': x_line, 'const': 1})).conf_in
ax.fill_between(x_line, conf_int[:, 0], conf_int[:, 1], color='red', alpha=0.2, lab

# Add labels and title
ax.set_xlabel('t_sod [g]')
ax.set_ylabel('y_sbp [mmHg]')
ax.set_title('Linear Regressions on Systolic Blood Pressure Variables')
ax.legend()
ax.grid(1, ls=':')
```

```
# Show the plot
plt.show()
```



```
In [11]: fig = sm.graphics.plot_partregress_grid(fit2)
         fig.tight_layout(pad=1.0)
```



DoWhy - Graphs

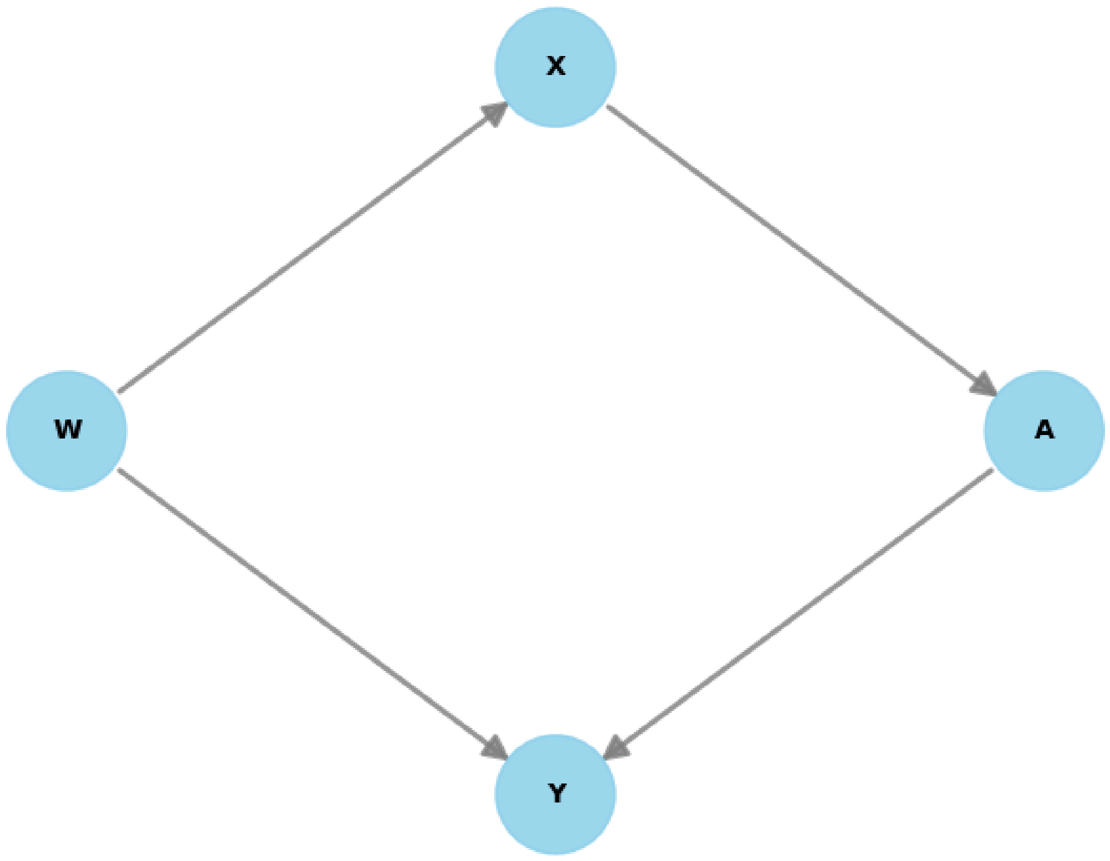
```
In [12]: import dowhy
         from dowhy import CausalModel

         import numpy as np
         import pandas as pd
         import matplotlib.pyplot as plt
```

```
In [13]: dowhy.__version__
```

```
Out[13]: '0.11.1'
```

```
In [14]: # some data, ignore it
         w=[i for i in range(10)]
         np.random.shuffle(w)
         df = pd.DataFrame(data = {'W': w, 'X': range(0,10), 'Y': range(0,100,10), 'A': rang
```

```
In [15]: # https://www.pywhy.org/dowhy/main/example_notebooks/load_graph_example.html
         # With DOT string
         model=CausalModel(
                 data = df,
                 treatment='X',
                 outcome='Y',
                 graph="digraph {W -> X;X -> A;A -> Y;W -> Y;}",
        #graph="digraph {W -> X;X -> A;Y -> A;W -> Y;}" # no directed path
        #graph="digraph {W -> X;X -> A;Y -> A;W -> Y; X->Y}"
)
model.view_model()
```



In [16]:
```
identified_estimand = model.identify_effect()
print(identified_estimand)
```

```
Estimand type: EstimandType.NONPARAMETRIC_ATE

### Estimand : 1
Estimand name: backdoor
Estimand expression:
 d
────(E[Y|W])
d[X]
Estimand assumption 1, Unconfoundedness: If U→{X} and U→Y then P(Y|X,W,U) = P(Y|X,W)

### Estimand : 2
Estimand name: iv
No such variable(s) found!

### Estimand : 3
Estimand name: frontdoor
Estimand expression:
 ⎡ d       d       ⎤
E⎢────(Y)⋅────([A])⎥
 ⎣d[A]    d[X]     ⎦
Estimand assumption 1, Full-mediation: A intercepts (blocks) all directed paths from
X to Y.
Estimand assumption 2, First-stage-unconfoundedness: If U→{X} and U→{A} then P(A|X,
U) = P(A|X)
Estimand assumption 3, Second-stage-unconfoundedness: If U→{A} and U→Y then P(Y|A,
X, U) = P(Y|A, X)
```

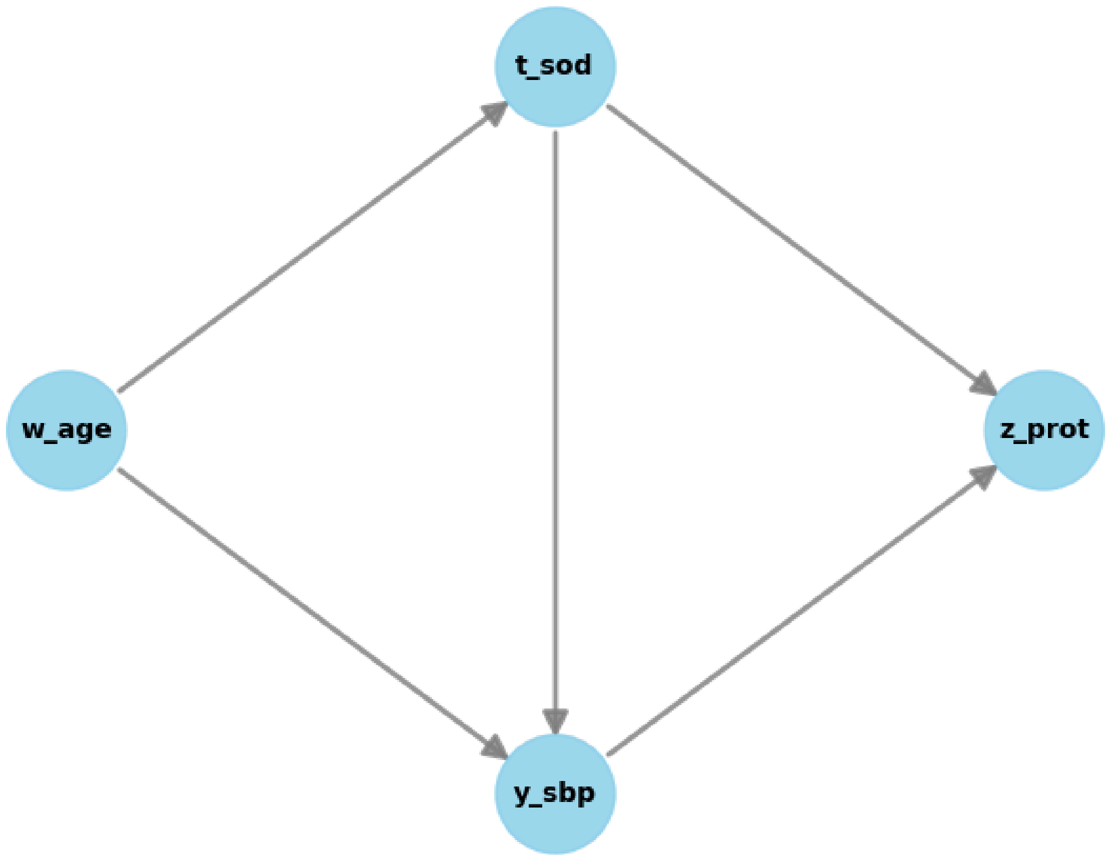
DoWhy Case Study

In [17]:

```
# Define causal model
model = CausalModel(
    data = data,
    graph = """
    digraph {
        t_sod -> y_sbp;
        w_age -> t_sod;
        w_age -> y_sbp;
        y_sbp -> z_prot;
        t_sod -> z_prot;
    }""",
    treatment= "t_sod",
    outcome= "y_sbp"
)
```

In [18]:

```
model.view_model()
```



```
In [19]: model.summary()
```

```
Out[19]: "Model to find the causal effect of treatment ['t_sod'] on outcome ['y_sbp']"
```

Identification

```
Identification of causal effect is the process of determining
whether the effect can be estimated using the available variables'
data.
```

Formally: convert target causal effect expression (e.g. $E[Y|do(A)]$) to a form that can be estimated using observed data distribution, i.e., without the do-operator.

DoWhy provides `identify_effect()` method with optional parameters for estimand types and identification methods. DoWhy complains if there are issues which prevent effect estimation (e.g. cycles in the graph). DoWhy supports the following identification algorithms:

- Backdoor
- Frontdoor
- Instrumental variable
- ID algorithm

```
In [20]: # Identify the causal effect
         identified_estimand = model.identify_effect(method_name="id-algorithm")
         print(identified_estimand)
```

```
Sum over {w_age}:
        Predictor: P(y_sbp|w_age,t_sod)
        Predictor: P(w_age)
```

```
In [21]: # Identify the causal effect
         identified_estimand = model.identify_effect()
         print(identified_estimand)
```

```
Estimand type: EstimandType.NONPARAMETRIC_ATE

### Estimand : 1
Estimand name: backdoor
Estimand expression:
   d
────────(E[y_sbp|w_age])
d[t_sod]
Estimand assumption 1, Unconfoundedness: If U→{t_sod} and U→y_sbp then P(y_sbp|t_so
d,w_age,U) = P(y_sbp|t_sod,w_age)

### Estimand : 2
Estimand name: iv
No such variable(s) found!

### Estimand : 3
Estimand name: frontdoor
No such variable(s) found!
```

estimand expression:

- `w_age` as confounder (recap: conditioning on common causes required to avoid unadjusted confounding)
- `t_sod` as treatment
- `y_sbp` as target variable
- `z_prot` is not involved (recap: not conditioning on common effects to avoid collider bias)

Estimation

Estimation is the "process of quantifying the target effect using the available data". DoWhy has a number of methods (causal inference) for estimation (regression, matching, stratification, and weighting estimators). Methods like inverse probability weighting are not restricted to linear relationships.

- Effect estimation with **backdoor** amounts to estimating a conditional probability distribution. Given an action A, an outcome Y and set of backdoor variables W, the causal effect is identified as $\sum_w E[Y|A, W = w]P(W = w)$.

    - Regression-based methods (DoWhy supports generalized linear models, e.g. to fit logistic regression models)
    - Distance-based matching (applicable only for binary treatments)
    - Propensity-based methods (applicable only for binary treatments)
    - Do-sampler / Pearlian inference / Pearlian interventions (demo)
- Estimating average causal effect with natural experiments (instrumental variables)
- Estimating conditional average causal effect (with EconML package)
    - another example: Conditional Average Treatment Effects (CATE)
- Estimating average causal effect using GCM (intervention)
    - GCM: "graphical causal models" extension of DoWhy
    - estimate such differences in a target node: $\mathbb{E}[Y|\mathrm{do}(T := A)] - \mathbb{E}[Y|\mathrm{do}(T := B)]$

In [22]:
```
# Estimate the causal effect and compare it with Average Treatment Effect
estimate = model.estimate_effect(identified_estimand,
                                 method_name="backdoor.linear_regression",
                                 test_significance=True,
                                 #target_units="att"
                                )
print(estimate)
print("Causal Estimate is " + str(estimate.value))
```

```
*** Causal Estimate ***

## Identified estimand
Estimand type: EstimandType.NONPARAMETRIC_ATE

### Estimand : 1
Estimand name: backdoor
Estimand expression:
   d
────────(E[y_sbp|w_age])
d[t_sod]
Estimand assumption 1, Unconfoundedness: If U→{t_sod} and U→y_sbp then P(y_sbp|t_so
d,w_age,U) = P(y_sbp|t_sod,w_age)

## Realized estimand
b: y_sbp~t_sod+w_age
Target units: ate

## Estimate
Mean value: 1.0584728347864996
p-value: [5.41635817e-176]

Causal Estimate is 1.0584728347864996
```

In [23]:
```
# compare with our linear regression fit from above
fit1.params
```

```
Out[23]:  Intercept   -0.009662
          t_sod        1.058473
          w_age        2.150229
          dtype: float64
```

```
In [24]:  # compare with fit including collider bias
          fit2.params
```

```
Out[24]:  Intercept   -0.022544
          t_sod       -0.910760
          w_age        0.422265
          z_prot       0.401882
          dtype: float64
```

The source performs a Monte-Carlo simulation to estimate the relative collider bias.

$$\frac{|\mu_{SOD,true}| - |\mu_{SOD,bias}|}{|\mu_{SOD,true}|}$$

```
In [25]:  # just for comparison the binarisation method as naive way to compute ACE:
          # average causal effect (t_sod):
          print(f'ACE = {data.query("sodium_upperlimit==1").y_sbp.mean()-data.query("sodium_u
```

```
ACE = 8.480982053797447
```

Refutation / Validation

```
Let us now look at ways of refuting the estimate obtained.
Refutation methods provide tests that every correct estimator
should pass. So if an estimator fails the refutation test (p-value
is <0.05), then it means that there is some problem with the
estimator.
```

Source

Here are the refutation methods from **refute()** documentation as table:

Add Random Common Cause:	Does the estimation method change its estimate after we add an independent random variable as a common cause to the dataset? (Hint: It should not)
Placebo Treatment:	What happens to the estimated causal effect when we replace the true treatment variable with an independent random variable? (Hint: the effect should go to zero)
Dummy Outcome:	What happens to the estimated causal effect when we replace the true outcome variable with an independent random variable? (Hint: The effect should go to zero)
Simulated Outcome:	What happens to the estimated causal effect when we replace the dataset with a simulated dataset based on a known data-generating

	process closest to the given dataset? (Hint: It should match the effect parameter from the data-generating process)
Add Unobserved Common Causes:	How sensitive is the effect estimate when we add an additional common cause (confounder) to the dataset that is correlated with the treatment and the outcome? (Hint: It should not be too sensitive)
Data Subsets Validation:	Does the estimated effect change significantly when we replace the given dataset with a randomly selected subset? (Hint: It should not)
Bootstrap Validation:	Does the estimated effect change significantly when we replace the given dataset with bootstrapped samples from the same dataset? (Hint: It should not)

We only test the first two, but you should generally check for every method. Keep these methods as part of your pipeline. It raises an alert, when an update on the graph or change in the algorithm introduced any issue.

- **Add Random Common Cause**: Does the estimation method change its estimate after we add an independent random variable as a common cause to the dataset? (Hint: It should not)

In [26]:
```
refute_results=model.refute_estimate(
    identified_estimand,
    estimate,
    method_name="random_common_cause")
print(refute_results)
```

```
Refute: Add a random common cause
Estimated effect:1.0584728347864996
New effect:1.0585352051031272
p value:0.9199999999999999
```

- **Placebo Treatment**: What happens to the estimated causal effect when we replace the true treatment variable with an independent random variable? (Hint: the effect should go to zero)

In [27]:
```
refute_results = model.refute_estimate(
    identified_estimand,
    estimate,
    method_name='placebo_treatment_refuter',
    placebo_type='permute',
    num_simulations=20)
print(refute_results)
```

```
Refute: Use a Placebo Treatment
Estimated effect:1.0584728347864996
New effect:-0.0003454525891186222
p value:0.4971000892900671
```

DoWhy GCM - Answering What-If Questions

Documentation

Intervention

DoWhy Graphical Causal Models (GCM) extension offers methods to answer what-if questions in our graphical causal model.

Recap:

```
[...] when performing interventions, we look into the future, for
counterfactuals we look into an alternative past. To reflect this
in the computation, when performing interventions, we generate all
noise using our causal models. For counterfactuals, we use the
noise from actual observed data.
```

Here is an example for interventions: We want to compute the average causal effect of changing the age from 65 to 70.

```
ACE = E[y_sbp | do(w_age := 70), t_sod] - E[y_sbp | do(w_age :=
65), t_sod]
```

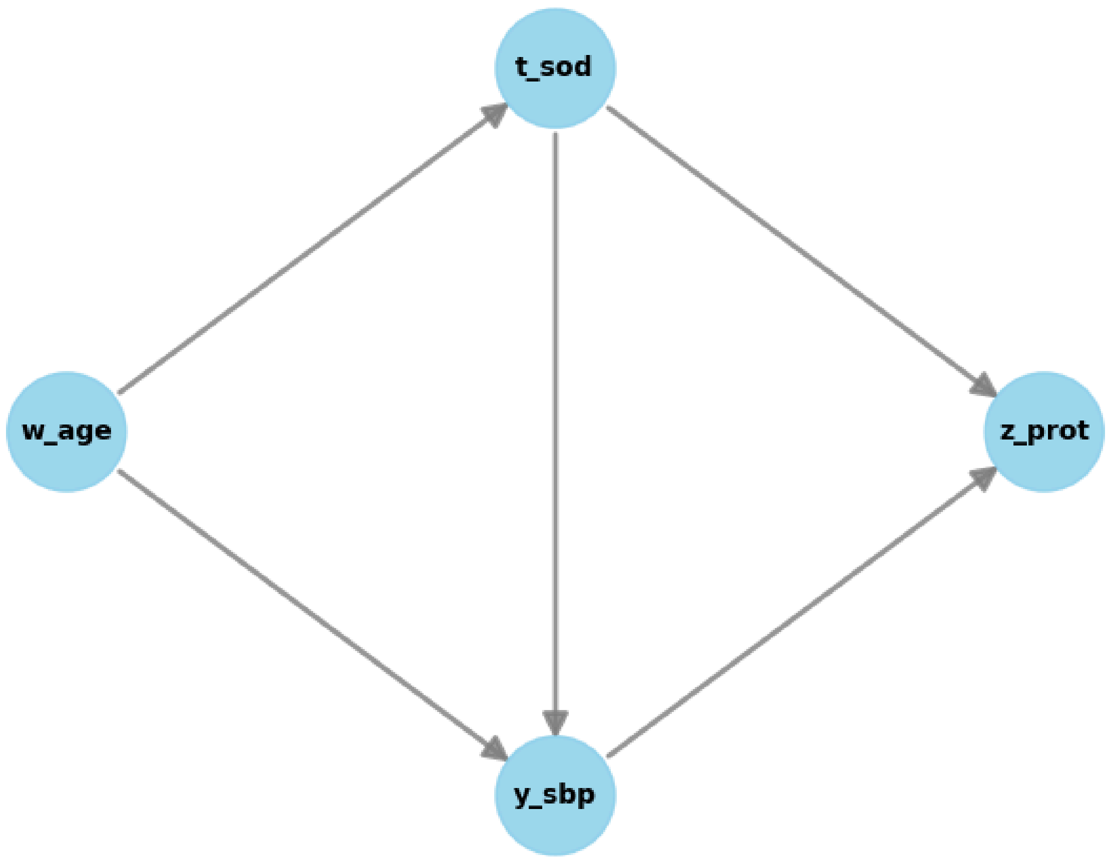
In [28]:
```
data
```

Out[28]:

	y_sbp	t_sod	w_age	z_prot	hypertension	sodium_upperlimit
0	140.695443	3.204703	63.462811	289.814072	1	1
1	133.235319	3.756593	60.806716	276.262146	0	1
2	144.788758	3.289475	65.628189	299.751282	1	1
3	136.606249	4.238763	61.696614	284.298170	0	1
4	143.396119	3.611865	65.791576	296.846522	1	1
...	...	...	...	...	...	...
995	125.437922	2.944481	56.470776	258.749576	0	1
996	156.403686	3.670764	71.434463	321.752170	1	1
997	131.840275	2.942233	59.413070	271.213898	0	1
998	145.518012	3.400475	65.431605	299.667019	1	1
999	167.433112	4.993656	74.928183	347.609654	1	1

1000 rows × 6 columns

In [29]:
```
model.view_model()
```



In [30]:
```
from dowhy import gcm
from scipy.stats import norm
```

In [31]:
```
causal_model = gcm.ProbabilisticCausalModel(model) # model._graph._graph
gcm.auto.assign_causal_mechanisms(causal_model, data)
gcm.fit(causal_model, data)
```

```
Fitting causal models:   0%|
| 0/4 [00:00<?, ?it/s]
Fitting causal mechanism of node t_sod:   0%|
| 0/4 [00:00<?, ?it/s]
Fitting causal mechanism of node y_sbp:   0%|
| 0/4 [00:00<?, ?it/s]
Fitting causal mechanism of node w_age:   0%|
| 0/4 [00:00<?, ?it/s]
Fitting causal mechanism of node z_prot:   0%|
| 0/4 [00:00<?, ?it/s]
Fitting causal mechanism of node z_prot: 100%|██████████████████████████████████████████████████████████████
██████| 4/4 [00:00<00:00, 950.71it/s]
```

In [32]:
```
# intervention
gcm.average_causal_effect(causal_model,
                          target_node='y_sbp',
                          interventions_alternative={'w_age': lambda x: 70},
                          interventions_reference={'w_age': lambda x: 65},
                          num_samples_to_draw=1000,
                          # observed_data=... # factual data. If not provided new d
                         )
```

Out[32]: 10.9841131148151

Computing Counterfactuals

```
I observed a certain outcome z for a variable Z where variable X
was set to a value x. What would have happened to the value of Z,
had I intervened on X to assign it a different value x'?
```

Recap:

```
[...] when performing interventions, we look into the future, for
counterfactuals we look into an alternative past. To reflect this
in the computation, when performing interventions, we generate all
noise using our causal models. For counterfactuals, we use the
noise from actual observed data.
```

Example: My (observed) values are:

- Age: 65
- Sodium intake: 3.9g/day
- Systolic blood pressure: 150 mmHg
- Proteinurie: 300mg

What would my values be if I only consumed 1.5 g of sodium per day?

In [33]:
```
observed = dict(w_age=[65], t_sod=[3.9], y_sbp=[150], z_prot=[300])
```

In [34]:
```
# This will not work if causal_model is not type of InvertibleStructuralCausalModel
# (causal_model is defined as ProbabilisticCausalModel in code above, so this must
gcm.counterfactual_samples(causal_model,
                           interventions={'t_sod': lambda x: 1.5},
                           observed_data=pd.DataFrame(data=observed)
)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Cell In[34], line 3
      1 # This will not work if causal_model is not type of InvertibleStructuralCaus
alModel
      2 # (causal_model is defined as ProbabilisticCausalModel in code above, so thi
s must fail)
----> 3 gcm.counterfactual_samples(causal_model,
      4                            interventions={'t_sod': lambda x: 1.5},
      5                            observed_data=pd.DataFrame(data=observed)
      6 )

File ~/.local/lib/python3.11/site-packages/dowhy/gcm/whatif.py:138, in counterfactua
l_samples(causal_model, interventions, observed_data, noise_data)
    136 if noise_data is None and observed_data is not None:
    137     if not isinstance(causal_model, InvertibleStructuralCausalModel):
--> 138         raise ValueError(
    139             "Since no noise_data is given, this has to be estimated from the
given "
    140             "observed_data. This can only be done with InvertibleStructuralC
ausalModel."
    141         )
    142     # Abduction: For invertible SCMs, we recover exact noise values from dat
a.
    143     noise_data = compute_noise_from_data(causal_model, observed_data)

ValueError: Since no noise_data is given, this has to be estimated from the given ob
served_data. This can only be done with InvertibleStructuralCausalModel.
```

In [35]:
```python
# so redefine our model as InvertibleStructuralCausalModel
causal_model = gcm.InvertibleStructuralCausalModel(graph=model)
# DoWhy does causal mechanism and data fitting
training_data = data[sbp_sod_age_prot]
gcm.auto.assign_causal_mechanisms(causal_model, training_data)
gcm.fit(causal_model, training_data)
```

```
Fitting causal models:   0%|
| 0/4 [00:00<?, ?it/s]
Fitting causal mechanism of node t_sod:   0%|
| 0/4 [00:00<?, ?it/s]
Fitting causal mechanism of node y_sbp:   0%|
| 0/4 [00:00<?, ?it/s]
Fitting causal mechanism of node w_age:   0%|
| 0/4 [00:00<?, ?it/s]
Fitting causal mechanism of node z_prot:   0%|
| 0/4 [00:00<?, ?it/s]
Fitting causal mechanism of node z_prot: 100%|███████████████████
██████████████████████████████████████████
██████| 4/4 [00:00<00:00, 1013.48it/s]
```

In [36]:
```python
gcm.counterfactual_samples(causal_model,
                           interventions={'t_sod': lambda x: 1.5},
                           observed_data=pd.DataFrame(data=observed)
)
```

Out[36]:

	w_age	t_sod	y_sbp	z_prot
0	65	1.5	147.459665	288.238812

In [37]:
```
observed
```

Out[37]:
```
{'w_age': [65], 't_sod': [3.9], 'y_sbp': [150], 'z_prot': [300]}
```

In [ ]:
```
```