1.1

$P(\text{OR gate} \mid x=1, z=1)$

AND gate			OR gate		
X	Y	Z	X	Y	Z
1	1	1	1	0	1
			1	1	1

$P(\text{OR gate}) = \frac{2}{3}$

1.2

$$P(y=1 \mid x=1, z=1) = \frac{P(y=1, x=1, z=1)}{P(x=1, z=1)}$$

$$= \frac{2}{3} \text{ (from truth tables)}$$

1.3 when $x = 1, z=1$ **Abduction**

$P(\text{OR gate}) = 2/3$ (choosing OR gate)
$P(\text{AND gate}) = 1/3$. (choosing AND gate)

Intervention : $x=0$
Prediction:

$$z=0 = \left(P(OR) * P(z=0 \mid x=0)\right) + P(AND) * P(z=0 \mid x=0)$$

$$= \left(\frac{2}{3} * \frac{1}{2} + \frac{1}{3} * 1\right)$$

$$= (2/3)$$

$$z=1 = P(OR) * P(z=1 \mid x=0) + P(AND) * P(z=1 \mid x=0)$$

$$= \left(\frac{2}{3} * \frac{1}{2} + \frac{1}{3} * 0\right) = 1/3$$

2.2.a

$$PNS = P(Y=1|X=1) - P(Y=1|X=0)$$
$$= 0.9198813 - 0.1992071$$
$$= 0.720.$$

$$PN = \frac{PNS}{P(Y=1|X=1)} = \frac{0.720}{0.919} = \underline{\underline{0.783}}$$

$$PS = \frac{PNS}{P(Y=0|X=0)}$$

$$P(Y=0|X=0) = \frac{P(Y=0, X=0)}{P(X=0)}$$

$$= \frac{0.4}{0.4995}.$$

$$PS = \frac{0.720}{0.4} * 0.4995 = \underline{\underline{0.8991}}$$

Scanned with CamScanner

In [1]:
```
import pyro
import pyro.distributions as dist
from pyro.infer import Importance, EmpiricalMarginal
import matplotlib.pyplot as plt
import torch
import numpy as np
import pandas as pd
import random
```

1. 4

In [2]:
```
def model():
    Nx = pyro.sample('Nx', dist.Bernoulli(torch.tensor(0.5)))
    Ny = pyro.sample('Ny', dist.Bernoulli(torch.tensor(0.5)))
    Nz = pyro.sample('Nz', dist.Bernoulli(torch.tensor(0.5)))
    X = pyro.sample('X', dist.Delta(Nx))
    Y = pyro.sample('Y', dist.Delta(Ny))
    ZVal = (Nz * (min((X+Y), torch.tensor(1.0)))) + ((torch.tensor(1.
0) - Nz) * min((X*Y), torch.tensor(1.0)))
    Z = pyro.sample('Z', dist.Delta(ZVal))
    return X, Y, Z
```

1.5

In [3]:
```
conditoned_model = pyro.condition(model, data = {"X": torch.tensor(1.
), "Z": torch.tensor(1.)})
```

In [4]:
```
posterior = Importance(conditoned_model, num_samples=1000).run()
```

Abduction Step (1.6.1)

In [7]:
```
for _ in range(1):
    trace = posterior()
    print(f"Nx is {trace.nodes['Nx']['value']}")
    print(f"Ny is {trace.nodes['Ny']['value']}")
    print(f"Nz is {trace.nodes['Nz']['value']}")
```

```
Nx is 1.0
Ny is 0.0
Nz is 1.0
```

Action Step (1.6.2)

In [8]:
```python
intervention_model = pyro.do(model, data = {"X": torch.tensor(0.)})
```

Prediction Step (1.6.3)

In [9]:
```python
prediction_model = pyro.condition(intervention_model, data={"Nx": tor
ch.tensor(0.), "Ny": torch.tensor(1.), "Nz": torch.tensor(0.)})
```

In [10]:
```python
trace_prediction = pyro.poutine.trace(prediction_model)
for _ in range(1):
    trace = trace_prediction.get_trace()
    print(f"Z is {trace.nodes['Z']['value']}")
```

```
Z is 0.0
```

1.6.d

In [11]:
```python
def counterfactual(posterior):
    # Abduction Step
    #Nx = torch.tensor(0.)
    #Ny = torch.tensor(0.)
    #Nz = torch.tensor(0.)
    # Get abduction samples
    for _ in range(1):
        trace = posterior()
        Nx = trace.nodes['Nx']['value']
        Ny = trace.nodes['Ny']['value']
        Nz = trace.nodes['Nz']['value']

    # Intervention Step
    intervention_model = pyro.do(model, data = {"X": torch.tensor(0.
)})

    # Condition Step
    prediction_model = pyro.condition(intervention_model, data={"Nx":
Nx, "Ny": Ny, "Nz": Nz})
    trace_prediction = pyro.poutine.trace(prediction_model)
    for _ in range(1):
        trace = trace_prediction.get_trace()
        z = trace.nodes['Z']['value']
        return z
```

In [12]:
```python
conditoned_model = pyro.condition(model, data = {"X": torch.tensor(1.
), "Z": torch.tensor(1.)})
posterior = Importance(conditoned_model, num_samples=1000).run()
counterfactual_samples = [counterfactual(posterior) for _ in range(10
00)]
```

In [13]:
```python
PZ1_X1Z1 = sum(counterfactual_samples)/len(counterfactual_samples)
```

In [14]:
```
pZ0_x1Z1 = 1 - PZ1_X1Z1
```

In [15]:
```
pZ0_x1Z1
```

Out[15]:
```
tensor(0.6710)
```

In [16]:
```
PZ1_X1Z1
```

Out[16]:
```
tensor(0.3290)
```

2.1

In [17]:
```
def scm():
    Nx = pyro.sample('Nx', dist.Bernoulli(torch.tensor(0.5)))
    Nq = pyro.sample('Nq', dist.Bernoulli(torch.tensor(0.9)))
    Ny = pyro.sample('Ny', dist.Bernoulli(torch.tensor(0.2)))
    X = pyro.sample('X', dist.Delta(Nx))
    Q = pyro.sample('Q', dist.Delta(Nq))
    YVal = (X and Q) or Ny
    Y = pyro.sample('Y', dist.Delta(YVal))
    return X, Q, Y
```

In [18]:
```
def counterfactualscm(posterior, intervention_model):

    Nx = torch.tensor(0.)
    Ny = torch.tensor(0.)
    Nq = torch.tensor(0.)
    for _ in range(1):
        trace = posterior()
        Nx = trace.nodes['Nx']['value']
        Ny = trace.nodes['Ny']['value']
        Nq = trace.nodes['Nq']['value']

    # Prediction Step
    prediction_model = pyro.condition(intervention_model, data={"Nx":
Nx, "Ny": Ny, "Nq": Nq})
    trace_prediction = pyro.poutine.trace(prediction_model)
    for _ in range(1):
        trace = trace_prediction.get_trace()
        y = trace.nodes['Y']['value']
        return y
```

In [31]:
```
condition_pn = pyro.condition(scm, data = {"X": torch.tensor(1.), "Y"
: torch.tensor(1.)})
# Intervention Step
intervention_model = pyro.do(scm, data = {"X": torch.tensor(0.)})
posterior = Importance(condition_pn, num_samples=1000).run()
pn = [counterfactualscm(posterior, intervention_model) for _ in range
(1000)]
```

In [32]:
```
pn = 1 - (sum(pn)/len(pn))
```

In [33]:
```
pn
```

Out[33]:
```
tensor(0.7920)
```

In [43]:
```
condition_ps = pyro.condition(scm, data = {"X": torch.tensor(0.), "Y"
: torch.tensor(0.)})
# Intervention Step
intervention_model_ps = pyro.do(scm, data = {"X": torch.tensor(1.)})
posterior_ps = Importance(condition_ps, num_samples=1000).run()
ps = [counterfactualscm(posterior_ps, intervention_model_ps) for _ in
range(1000)]
```

In [44]:
```
ps = sum(ps)/len(ps)
```

In [45]:
```
ps
```

Out[45]:
```
tensor(0.9000)
```

Question 3

3.1) Total Effect (TE) $= E[Y \mid do(X=1)] - E[Y \mid do(X=0)]$

$= I(14 > 10) - I(0 > 10)$

$= 1 - 0$

$= \underline{\underline{1}}$

3.2) CDE $= E[Y \mid do(X=1, T=0)] - E[Y \mid do(X=0, T=0)]$

$= I(8 > 10) - I(0 > 10)$

$= \underline{\underline{0}}$

3.3) NIE $= E[Y \mid do(X=0, T=T_{X=1})] - E[Y \mid do(X=0, T=T_{X=0})]$

$= E[Y \mid do(X=0, T=3)] - E[Y \mid do(X=0, T=0)]$

$= I(6 > 10) - I(0 > 10)$

$= \underline{\underline{0}}$

3.4) NIEr $= E[Y \mid do(X=1, T=T_{X=0})] - E[Y \mid do(X=1, T=T_{X=1})]$

$= E[Y \mid do(X=1, T=0)] - E[Y \mid do(X=1, T=3)]$

$= I(8 > 10) - I(14 > 10)$

$= \underline{\underline{-1}}$

3.5) NDE $=$ TE $+$ NIEr

$= 1 + (-1)$

$= \underline{\underline{0}}$

This implies that the thrash related to the new UI is driving the conversion amongst users.

When the time spent familiarizing with the new UI will go down, so will the engagement and user conversion.

3.6) Even if the exogenous variables are not degenerate their sample mean is still 0 and will result in the same conclusions for NDE and NIE.

3.7) Since user features (U) is not a mediator or on a causal path between X & Y it is not going to effect the approach for computing NDE and NIE.

In [2]:
```
import numpy as np
import pandas as pd

import torch
import pyro
import pyro.distributions as dist

pyro.set_rng_seed(101)
pyro.enable_validation(True)
```

In [3]:
```
df = pd.read_csv('./driver.csv', index_col=0)
```

In [4]:
```
df.head()
```

Out[4]:

	x	y	z
1	0	1.490090	0
2	1	5.170279	1
3	1	7.434170	1
4	0	1.531446	0
5	0	0.935765	0

In [5]:
```
df.describe()
```

Out[5]:

	x	y	z
count	1000.000000	1000.000000	1000.000000
mean	0.498000	2.531169	0.491000
std	0.500246	2.535948	0.500169
min	0.000000	-3.353001	0.000000
25%	0.000000	0.244929	0.000000
50%	0.000000	2.321081	0.000000
75%	1.000000	4.864423	1.000000
max	1.000000	7.895047	1.000000

In [39]:
```
# Calculating do operation using valid adjustment set formula
def estimate_y(x):
    est_sum = 0
    for z in df['z'].unique():
        filter_df = df.query('x == @x & z == @z')
        mean_y = np.mean(filter_df['y'])

        prob_z = df.query('z == @z').size/df.size

        est_sum += (mean_y*prob_z)

    return est_sum
```

In [37]:
```
# E[Y_X=x | X=cx]
def estimate_counterfactual(x, cx):
    est_sum = 0
    for z in df['z'].unique():
        filter_df = df.query('x == @x & z == @z')
        mean_y = np.mean(filter_df['y'])

        filter_df = df.query('x == @cx')
        prob_z = filter_df.query('z == @z').size/filter_df.size

        est_sum += (mean_y*prob_z)

    return est_sum
```

In [38]:
```
print('E(Y_X=0 | X=1) = %.3f' % (estimate_counterfactual(0, 1)))
print('ETT = %.3f' % (estimate_counterfactual(1, 1) - estimate_counter
factual(0, 1)))
print('E(Y_X=1 - Y_X=0) = %.3f - %.3f = %.3f' % (estimate_y(1), estima
te_y(0), estimate_y(1) - estimate_y(0)))
```

```
E(Y_X=0 | X=1) = 3.091
ETT = 1.346
E(Y_X=1 - Y_X=0) = 3.075 - 1.861 = 1.214
```

Analysis

It is clear from the results that the ETT is slight higher than the total effect. This indicates that the training program must have had an impact on the motivation of the drivers results in higher revenues. Having done the ETT analysis, we can safely say that the training program in general is going to result in higher revenues despite the original motivation of the drivers.

```
In [ ]:
```