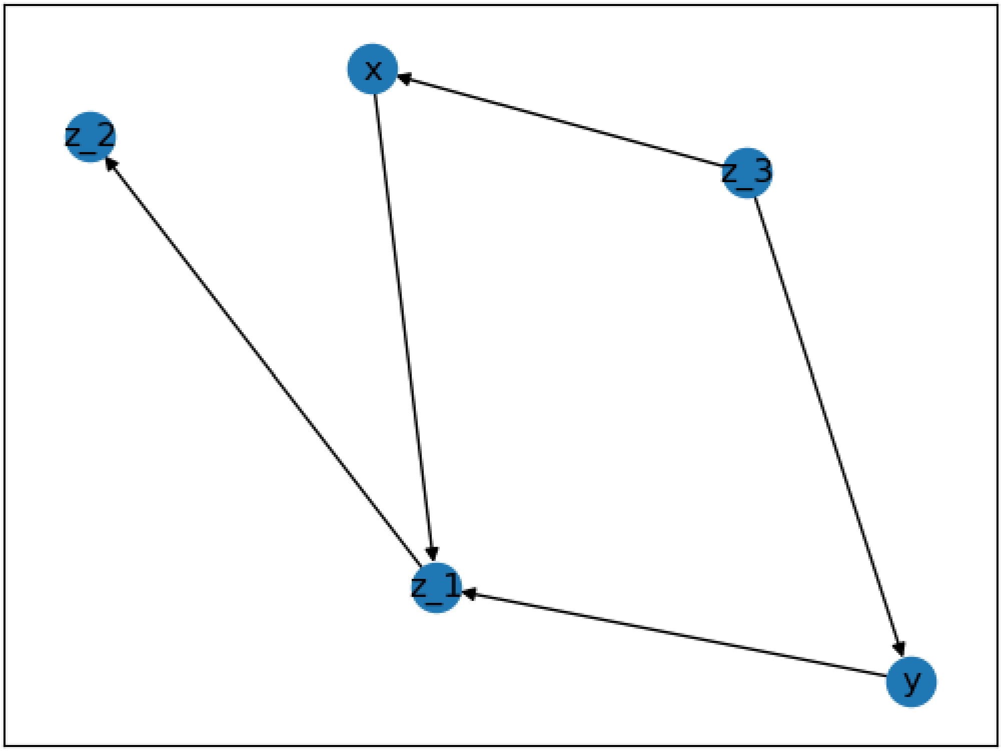
In [1]:

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import networkx as nx
from matplotlib import pyplot as plt
import random
```

In [2]:

```
#Graph in Exercise 2
dag_1 = nx.DiGraph()
dag_1.add_edges_from([("z_3", "x"), ("z_3", "y"), ("x", "z_1"), ("y", "z_1")
plt.tight_layout()
nx.draw_networkx(dag_1, arrows=True)
plt.show()
```



In [4]:

```
def find_collidors(graph):
    colliders = []
    for node in graph.nodes():
        parents = list(graph.predecessors(node))
        if len(parents) > 1:
            colliders.append((parents,node))
    return colliders

find_collidors(dag_1)
```

Out[4]:

```
[(['x', 'y'], 'z_1')]
```

In [5]:

```
def find_direct_paths(graph):
    direct_paths = []
    for node in graph.nodes():
        parents = list(graph.predecessors(node))
        if len(parents) == 1:
            direct_paths.append((parents,node))
    return direct_paths

find_direct_paths(dag_1)
```

Out[5]:

```
[(['z_3'], 'x'), (['z_3'], 'y'), (['z_1'], 'z_2')]
```

In [6]:

```
def find_open_door_paths(graph):
    open_door_paths = []
    for edge in graph.edges():
        for neighbor in graph.neighbors(edge[1]):
            if neighbor != edge[0] and not graph.has_edge(edge[0], neighbor)
                open_door_paths.append((edge[0], edge[1], neighbor))
    return open_door_paths

find_open_door_paths(dag_1)
```

Out[6]:

```
[('z_3', 'x', 'z_1'),
 ('z_3', 'y', 'z_1'),
 ('x', 'z_1', 'z_2'),
 ('y', 'z_1', 'z_2')]
```

In [7]:

```
def find_back_door_paths(graph):
    back_door_paths = []
    for node1 in graph.nodes():
        for node2 in graph.nodes():
            if node1 != node2 and not graph.has_edge(node1, node2):
                common_ancestors = set(graph.predecessors(node1)).intersecti
                if common_ancestors:
                    back_door_paths.append((node1, common_ancestors.pop(), r
    return back_door_paths

find_back_door_paths(dag_1)
```

Out[7]:

```
[('x', 'z_3', 'y'), ('y', 'z_3', 'x')]
```

In [27]:

```
#Exercise 17
random.seed(42)

def generate_binary_array(size, prob_true):
    return [1 if random.random() > prob_true else 0 for i in range(size)]

def generate_dependent_binary_array(size, conditions, prob_true):
    return [1 if (condition and random.random() > 0.5) or
                 (not condition and random.random() > 0.5)
            else 0 for condition in conditions]

# Generate P(z_3)=0.3
z3_val = generate_binary_array(1000, 0.7)

# Generate x based on value of z3
x_probs = [0.5 if z3_val == 1 else 0.5]
x = generate_dependent_binary_array(1000, z3, x_probs)
```

```
# Generate y based on value of z3
y_probs = [0.4 if z3_val == 1 else 0.6]
y = generate_dependent_binary_array(1000, z3, y_probs)

# Generate z1 based on value of x and y
z1_probs = {(1, 1): 0.6, (1, 0): 0.4, (0, 1): 0.7, (0, 0): 0.3}
z1 = generate_dependent_binary_array(1000, list(zip(y, x)), z1_probs[(1, 1)]
```

In [30]:

```
random_df = pd.DataFrame({'z3':z3, 'x':x, 'y':y, 'z1':z1})
random_df.head()
```

Out[30]:

	z3	x	y	z1
0	0	0	0	1
1	0	1	1	1
2	0	1	0	1
3	0	1	0	0
4	1	0	0	0

In [34]:

```
# Equation 42
probabilities = df.groupby(['z3', 'x']).agg(
    py=('y', lambda x: (x == 1).sum() / len(x)),
    pz1=('z1', lambda x: (x == 1).sum() / len(x)),
    pz3=('z3', 'count')
)
result_42 = (probabilities['py'] * probabilities['pz1'] * probabilities['pz3
print(result_42)
```

```
0.4984239434174164
```

In [41]:

```
#Equation 43
filtered_df_condition_1 = (df['z1'] == 1) & (df['x'] == 1) & (df['y'] == 1)
filtered_df_condition_2 = (df['y'] == 1)

grouped_condition_1 = df[filtered_df_condition_1].groupby(['z3', 'x'])
grouped_condition_2 = df[filtered_df_condition_2].groupby(['z3', 'x'])

probabilities_condition_1 = grouped_condition_1.agg(
    py=('y', lambda x: (x == 1).sum() / len(x)),
    pz1=('z1', lambda x: (x == 1).sum() / len(x)),
    pz3=('z3', 'count'))

probabilities_condition_2 = grouped_condition_2.agg(
    py=('y', lambda x: (x == 1).sum() / len(x)),
    pz1=('z1', lambda x: (x == 1).sum() / len(x)),
    pz3=('z3', 'count'))

result_43 = (probabilities_condition_1['py'] * probabilities_condition_1['pz
print(result_43)
```

```
0.5327868852459017
```

In [ ]: