Module 1

This notebook contains the code used for the coding examples in Module 1 of the 2025 course “Causal Inference with Linear Regression: A Modern Approach” by CausAI.

Imports

```python
import numpy as np
import pandas as pd
```

Customer Churn Example (Videos 1.13, 1.14 & 1.15)

Causal Graph: $DiscountSent \leftarrow HistoricalShoppingFrequency \rightarrow Churn, DiscountSent \rightarrow Churn$

$DiscountSent$ is a binary variable, $HistoricalShoppingFrequency$ is denoted as variable $ShoppingFrequency$, and has 3 categories $low, medium, high$

Generate Data

```python
# Set random seed for reproducibility
np.random.seed(3)

# Parameters for the DGP
n_customers = 50000
```

```python
def simulate_data(n, random_treatment=False):

    # Define shopping frequency probabilities
    shopping_frequency_probs = {"low": 0.3, "medium": 0.5, "high": 0.2}

    # Define probabilities for discount sent (based on shopping frequency)
    discount_given_probs = {"low": 0.8, "medium": 0.5, "high": 0.2}

    # Define base churn probabilities (based on shopping frequency)
    base_churn_probs = {"low": 0.5, "medium": 0.3, "high": 0.1}

    # Define the treatment effect of discount on churn (negative effect, reduces churn)
    treatment_effect = -0.1

    # Generate synthetic data
    customer_ids = np.arange(1, n + 1)

    shopping_frequencies = np.random.choice(
        ["low", "medium", "high"],
        size=n_customers,
        p=list(shopping_frequency_probs.values()),
    )

    if random_treatment == True:
        discount_sent = np.random.binomial(1, 0.5, size=n_customers)
    else:
        discount_sent = np.array(
            [
                np.random.binomial(1, discount_given_probs[freq])
                for freq in shopping_frequencies
            ]
        )

    # Generate churn probabilities and churn outcome
    churn_probs = np.array(
        [
            base_churn_probs[freq] + (treatment_effect if discount == 1 else 0)
            for freq, discount in zip(shopping_frequencies, discount_sent)
        ]
    )

    churn_outcomes = np.random.binomial(1, churn_probs)

    # Create DataFrame
    data = pd.DataFrame(
        {
            "CustomerID": customer_ids,
            "ShoppingFrequency": shopping_frequencies,
            "DiscountSent": discount_sent,
            "Churn": churn_outcomes,
        }
    )

    return data
```

```
data_observational = simulate_data(
    n_customers
)  # observational dataset, where discount given depends on shopping frequency
data_random_experiment = simulate_data(
    n_customers, random_treatment=True
)  # experimental dataset, where whether a discount is given is randomized
```

```
data_observational.head()
```

	CustomerID	ShoppingFrequency	DiscountSent	Churn
0	1	medium	0	1
1	2	medium	1	0
2	3	low	1	1
3	4	medium	1	0
4	5	high	0	0

```
data_random_experiment.head()
```

	CustomerID	ShoppingFrequency	DiscountSent	Churn
0	1	high	0	0
1	2	medium	1	0
2	3	medium	0	0
3	4	low	1	0
4	5	medium	1	1

Compute $E[churn|discount = 1] - E[churn|discount = 0]$ in observational data

```
# Group by DiscountSent and calculate mean churn for each group
grouped_on_treatment = (
    data_observational[["DiscountSent", "Churn"]]
    .groupby(["DiscountSent"])
    .mean()
)
grouped_on_treatment
```

	Churn
DiscountSent	
0	0.257992
1	0.277119

```python
# Extract mean churn rates for treated and untreated groups
churn_mean_treated = grouped_on_treatment.loc[
    1, "Churn"
]  # approximation for E[Churn | Discount = 1]
churn_mean_untreated = grouped_on_treatment.loc[
    0, "Churn"
]  # approximation for E[Churn | Discount = 0]

# Calculate the difference
difference = (
    churn_mean_treated - churn_mean_untreated
)  # estimate for E[Churn | Discount = 1] - E[Churn | Discount = 0]

print(f"Mean churn rate (Treated): {churn_mean_treated:.4f}")
print(f"Mean churn rate (Untreated): {churn_mean_untreated:.4f}")
print(f"Difference in churn rates: {difference:.4f}")
```

```
Mean churn rate (Treated): 0.2771
Mean churn rate (Untreated): 0.2580
Difference in churn rates: 0.0191
```

This quantity is not equal to the ATE of -0.1, but instead a biased representation of it. We don't have ignorability due to the confounder $shopping frequency$.

Compute $E[churn|discount = 1] - E[churn|discount = 0]$ in experimental data

```python
# Group by DiscountSent and calculate mean churn
grouped_on_treatment_random = (
    data_random_experiment[["DiscountSent", "Churn"]]
    .groupby(["DiscountSent"])
    .mean()
)

# Extract mean churn rates for treated and untreated groups
churn_mean_treated_random = grouped_on_treatment_random.loc[1, "Churn"]
churn_mean_untreated_random = grouped_on_treatment_random.loc[0, "Churn"]

# Calculate the difference
difference_random = churn_mean_treated_random - churn_mean_untreated_random

print(f"Mean churn rate (Treated): {churn_mean_treated_random:.4f}")
print(f"Mean churn rate (Untreated): {churn_mean_untreated_random:.4f}")
print(f"Difference in churn rates: {difference_random:.4f}")
```

```
Mean churn rate (Treated): 0.2223
Mean churn rate (Untreated): 0.3230
Difference in churn rates: -0.1007
```

Here the difference is very close to the true ATE of -0.1. We have ignorability and so the ATE is simply equal to the associational difference $E[churn|discount = 1] - E[churn|discount = 0]$. Any differences between ATE and calculated quantity are simply due to statistical noise (feel free to check this by increasing the sample size).

Compute $E[churn|discount = 1, shoppingfrequency = z]$ and $E[churn|discount = 0, shoppingfrequency = z]$, $z \in \{\{low\}, \{medium\}, \{high\}\}$, take their difference and weight by probability of that value of $shoppingfrequency$ occuring

```
# Calculate average churn rates stratified by shopping frequency
stratified_avg_churn = (
    data_observational.groupby(["ShoppingFrequency", "DiscountSent"])["Churn"]
    .mean()
    .reset_index()
)
stratified_avg_churn
```

	ShoppingFrequency	DiscountSent	Churn
0	high	0	0.096690
1	high	1	0.000000
2	low	0	0.502242
3	low	1	0.400635
4	medium	0	0.301148
5	medium	1	0.202279

```
# Calculate proportions of shopping frequencies
shopping_freq_proportions = (
    data_observational["ShoppingFrequency"]
    .value_counts(normalize=True)
    .to_dict()
)

# Compute overall ATE using the adjustment formula
```

```python
ate = 0
for freq, prop in shopping_freq_proportions.items():
    treated = stratified_avg_churn[
        (stratified_avg_churn["ShoppingFrequency"] == freq)
        & (stratified_avg_churn["DiscountSent"] == 1)
    ]["Churn"].values[0]
    untreated = stratified_avg_churn[
        (stratified_avg_churn["ShoppingFrequency"] == freq)
        & (stratified_avg_churn["DiscountSent"] == 0)
    ]["Churn"].values[0]
    ate += prop * (treated - untreated)


print(f"Overall Average Treatment Effect (ATE): {ate:.4f}")
```

```
Overall Average Treatment Effect (ATE): -0.0993
```

We have conditional ignorability given $shopping frequency$, and so we can apply the adjustment formula to obtain the true ATE using our observational data.

This notebook was converted with convert.ploomber.io