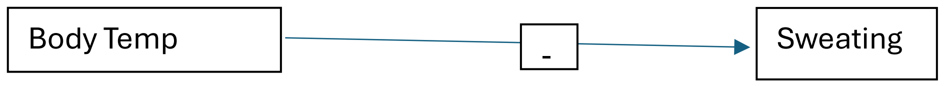
Homework Reflection 5

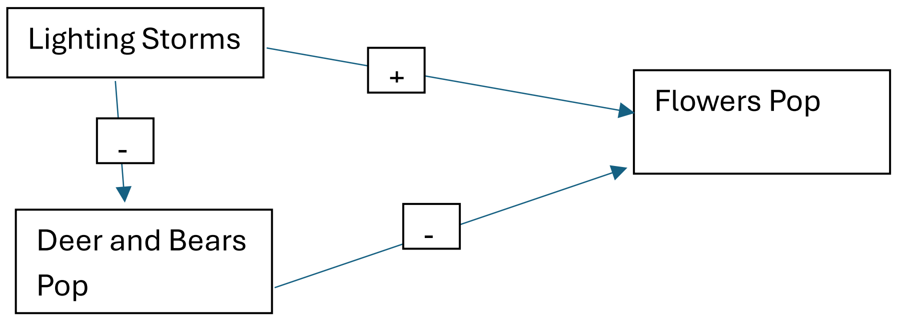
1. Draw a diagram for the following negative feedback loop:
Sweating causes body temperature to decrease. High body temperature causes sweating.
A negative feedback loop means that one thing increases another while the second thing decreases the first.
Remember that we are using directed acyclic graphs where two things cannot directly cause each other.



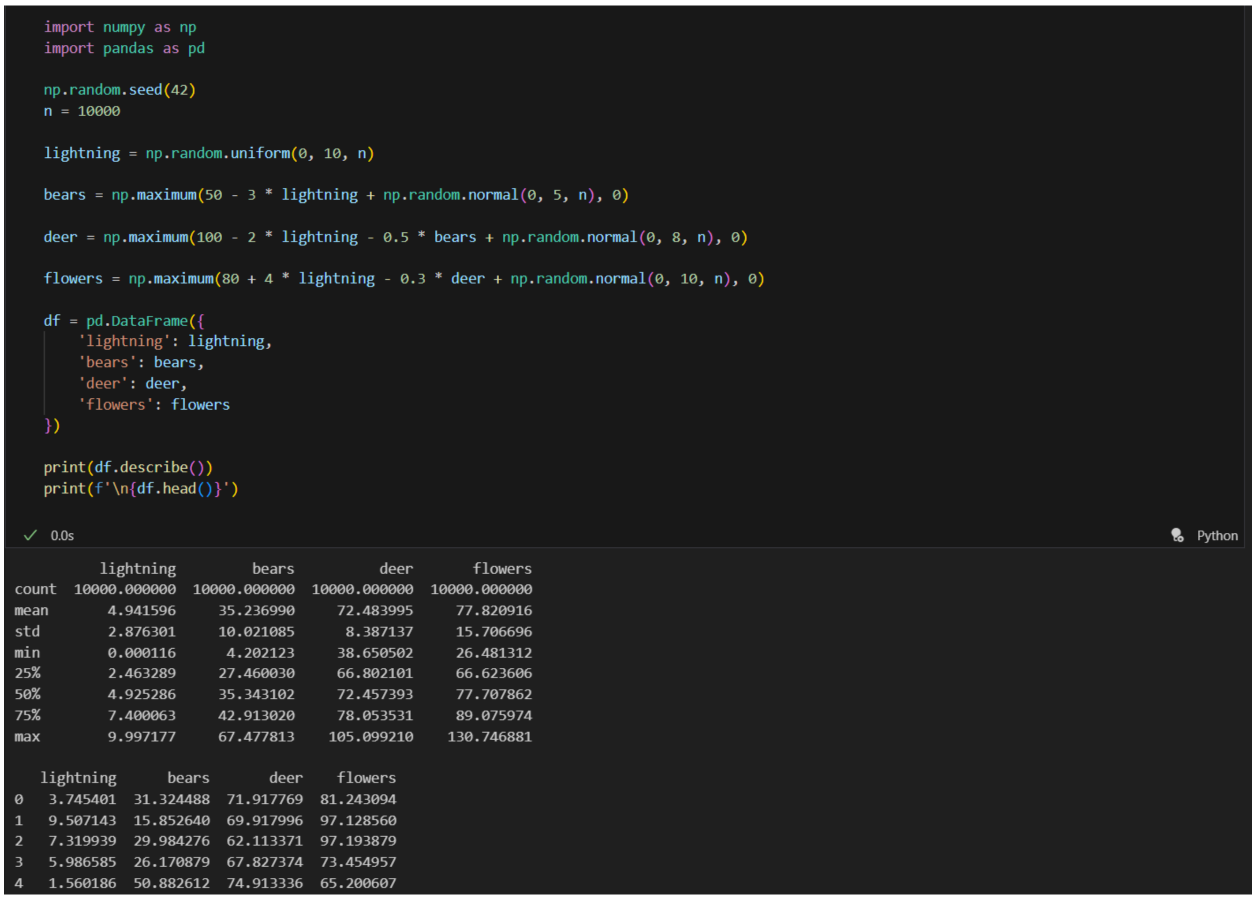
2. Describe an example of a positive feedback loop. This means that one things increases another while the second things also increases the first.

An example is social media engagement. When a post gets more likes it becomes more visible to other users. Higher visibility leads to even more likes.

3. Draw a diagram for the following situation:
Lightning storms frighten away deer and bears, decreasing their population, and cause flowers to grow, increasing their population.
Bears eat deer, decreasing their population.
Deer eat flowers, decreasing their population.



Write a dataset that simulates this situation. (Show the code.) Include noise / randomness in all cases.
Identify a backdoor path with one or more confounders for the relationship between deer and flowers.

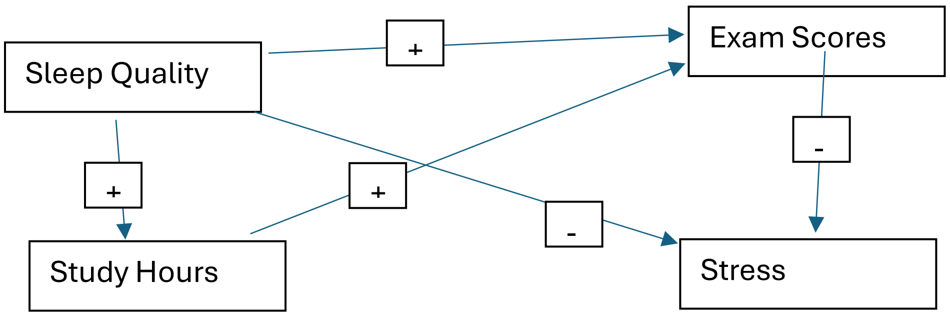


```python
import numpy as np
import pandas as pd

np.random.seed(42)
n = 10000

lightning = np.random.uniform(0, 10, n)

bears = np.maximum(50 - 3 * lightning + np.random.normal(0, 5, n), 0)

deer = np.maximum(100 - 2 * lightning - 0.5 * bears + np.random.normal(0, 8, n), 0)

flowers = np.maximum(80 + 4 * lightning - 0.3 * deer + np.random.normal(0, 10, n), 0)

df = pd.DataFrame({
    'lightning': lightning,
    'bears': bears,
    'deer': deer,
    'flowers': flowers
})

print(df.describe())
print(f'\n{df.head()}')
```

```
          lightning         bears          deer       flowers
count  10000.000000  10000.000000  10000.000000  10000.000000
mean       4.941596     35.236990     72.483995     77.820916
std        2.876301     10.021085      8.387137     15.706696
min        0.000116      4.202123     38.650502     26.481312
25%        2.463289     27.460030     66.802101     66.623606
50%        4.925286     35.343102     72.457393     77.707862
75%        7.400063     42.913020     78.053531     89.075974
max        9.997177     67.477813    105.099210    130.746881

   lightning      bears       deer    flowers
0   3.745401  31.324488  71.917769  81.243094
1   9.507143  15.852640  69.917996  97.128560
2   7.319939  29.984276  62.113371  97.193879
3   5.986585  26.170879  67.827374  73.454957
4   1.560186  50.882612  74.913336  65.200607
```

Lightning is a confounder for the relationship between deer and flowers. It decreases deer population while increasing flower population. If we want to estimate the causal effect of deer on flowers we need to control for lightning.

4. Draw a diagram for a situation of your own invention. The diagram should include at least four nodes, one confounder, and one collider. Be sure that it is acyclic (no loops). Which node would say is most like a treatment (X)? Which is most like an outcome (Y)?



Confounder: Sleep Quality is a confounder for the relationship between Study Hours and Exam Score. Good sleep increases both study hours and exam scores directly.

Collider: Stress is a collider. Both Study Hours and Exam Score cause Stress.

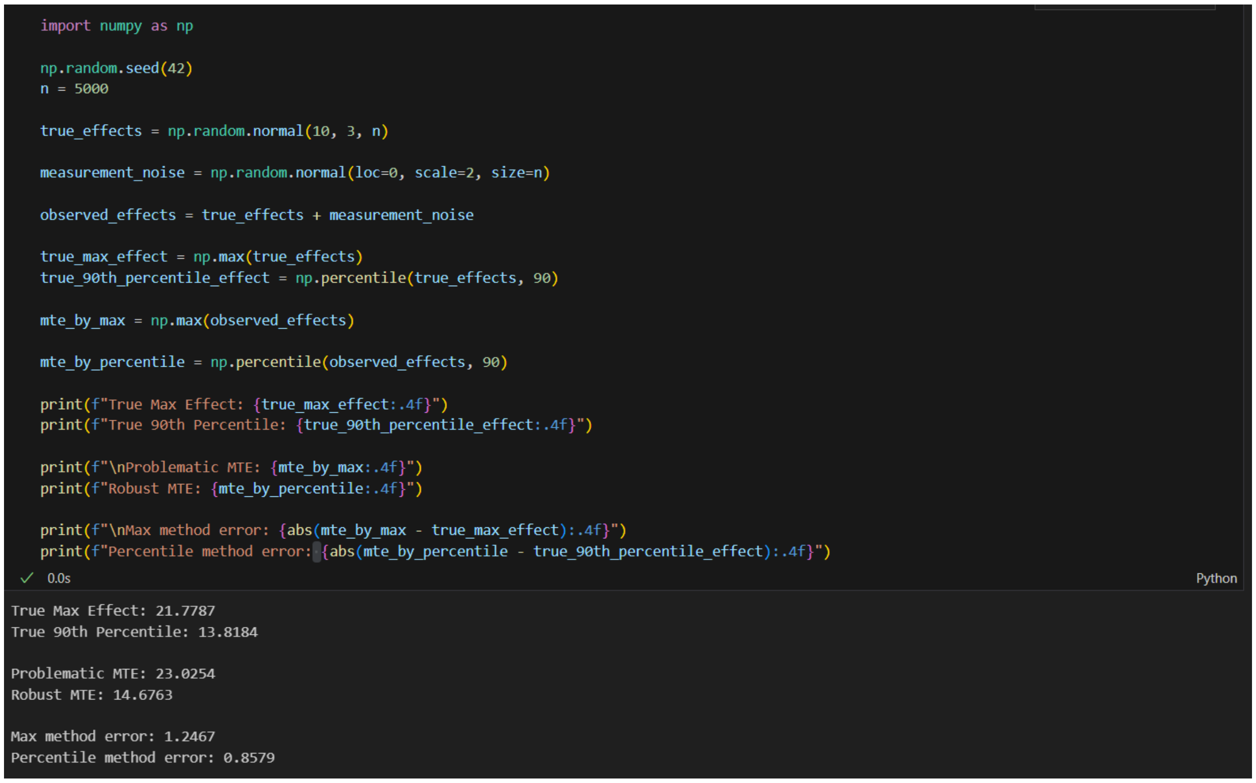
Treatment (X): Study Hours is most like a treatment because it's something a student can directly control or intervene on.
Outcome (Y): Exam Score is most like an outcome because it's what we ultimately care about measuring.

Homework Reflection 6

1. What is a potential problem with computing the Marginal Treatment Effect simply by comparing each untreated item to its counterfactual and taking the maximum difference? (Hint: think of statistics here. Consider that only the most extreme item ends up being used to estimate the MTE. That's not necessarily a bad thing; the MTE is supposed to come from the untreated item that will produce the maximum effect. But there is nevertheless a problem.)
Possible answer: We are likely to find the item with the most extreme difference, which may be high simply due to randomness.
(Please explain / justify this answer, or give a different one if you can think of one.)

The main problem is that we are picking the single most extreme value from our entire dataset. When you look at thousands of observations some will have unusually high treatment effects just by chance. This is especially true if there's measurement error or noise in the counterfactual matching process. For example, if you flip a coin 1000 times and record the longest streak of heads you'll probably get something like 10 heads in a row even though each flip is random. The maximum treatment effect might just be a statistical outlier rather than a true indication of who benefits most from treatment.
The problem gets worse with larger datasets. With more observations we have more chances to find an extreme value that's just noise. This means our MTE estimate becomes unreliable and will vary a lot if we collect new data. We're essentially cherry picking the one observation that had the best luck in terms of random variation.

2. Propose a solution that remedies this problem and write some code that implements your solution. It's very important here that you clearly explain what your solution will do.
Possible answer: maybe we could take the 90th percentile of the treatment effect and use it as a proxy for the Marginal Treatment Effect.
(Either code this answer or choose a different one.)



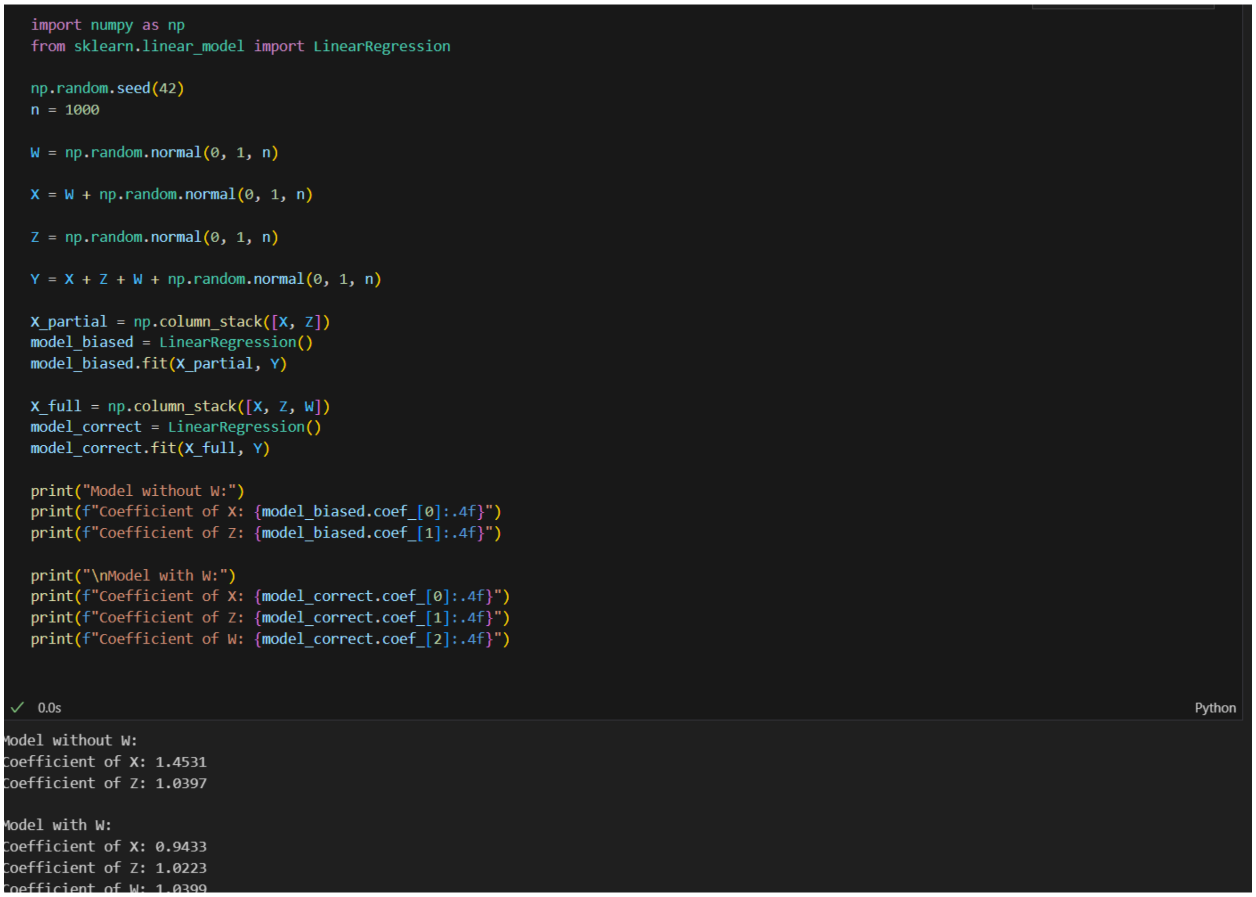
```python
import numpy as np

np.random.seed(42)
n = 5000

true_effects = np.random.normal(10, 3, n)

measurement_noise = np.random.normal(loc=0, scale=2, size=n)

observed_effects = true_effects + measurement_noise

true_max_effect = np.max(true_effects)
true_90th_percentile_effect = np.percentile(true_effects, 90)

mte_by_max = np.max(observed_effects)

mte_by_percentile = np.percentile(observed_effects, 90)

print(f"True Max Effect: {true_max_effect:.4f}")
print(f"True 90th Percentile: {true_90th_percentile_effect:.4f}")

print(f"\nProblematic MTE: {mte_by_max:.4f}")
print(f"Robust MTE: {mte_by_percentile:.4f}")

print(f"\nMax method error: {abs(mte_by_max - true_max_effect):.4f}")
print(f"Percentile method error: {abs(mte_by_percentile - true_90th_percentile_effect):.4f}")
```

✓ 0.0s Python

```
True Max Effect: 21.7787
True 90th Percentile: 13.8184

Problematic MTE: 23.0254
Robust MTE: 14.6763

Max method error: 1.2467
Percentile method error: 0.8579
```

Homework Reflection 7

1. Create a linear regression model involving a confounder that is left out of the model. Show whether the true correlation between $$X$$ and $$Y$$ is overestimated, underestimated, or neither. Explain in words why this is the case for the given coefficients you have chosen.



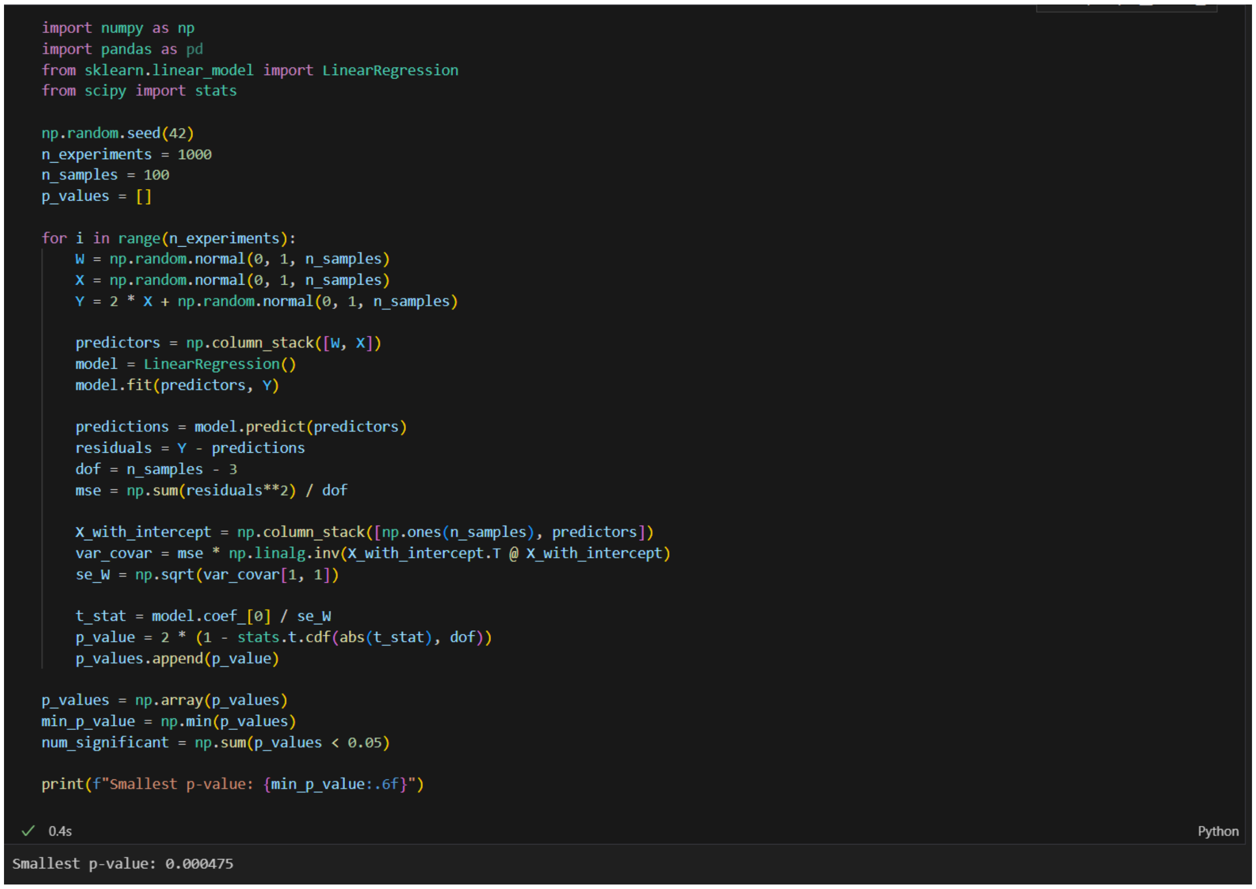
```python
import numpy as np
from sklearn.linear_model import LinearRegression

np.random.seed(42)
n = 1000

W = np.random.normal(0, 1, n)

X = W + np.random.normal(0, 1, n)

Z = np.random.normal(0, 1, n)

Y = X + Z + W + np.random.normal(0, 1, n)

X_partial = np.column_stack([X, Z])
model_biased = LinearRegression()
model_biased.fit(X_partial, Y)

X_full = np.column_stack([X, Z, W])
model_correct = LinearRegression()
model_correct.fit(X_full, Y)

print("Model without W:")
print(f"Coefficient of X: {model_biased.coef_[0]:.4f}")
print(f"Coefficient of Z: {model_biased.coef_[1]:.4f}")

print("\nModel with W:")
print(f"Coefficient of X: {model_correct.coef_[0]:.4f}")
print(f"Coefficient of Z: {model_correct.coef_[1]:.4f}")
print(f"Coefficient of W: {model_correct.coef_[2]:.4f}")
```

✓ 0.0s Python

```
Model without W:
Coefficient of X: 1.4531
Coefficient of Z: 1.0397

Model with W:
Coefficient of X: 0.9433
Coefficient of Z: 1.0223
Coefficient of W: 1.0399
```

The coefficient of X is overestimated when we leave out W. This happens because X and W are correlated and W also affects Y. When we omit W from the model the regression tries to explain all the variation in Y using only X and Z. Since W influences Y and W is correlated with X the model incorrectly attributes some of W's effect on Y to X instead.

2. Perform a linear regression analysis in which one of the coefficients is zero, e.g.
W = [noise]
X = [noise]
Y = 2 * X + [noise]
And compute the p-value of a coefficient - in this case, the coefficient of W.
(This is the likelihood that the estimated coefficient would be as high or low as it is, given that the actual coefficient is zero.)
If the p-value is less than 0.05, this ordinarily means that we judge the coefficient to be nonzero (incorrectly, in this case.)
Run the analysis 1000 times and report the best (smallest) p-value.
If the p-value is less than 0.05, does this mean the coefficient actually is nonzero? What is the problem with repeating the analysis?



```
import numpy as np
import pandas as pd
from sklearn.linear_model import LinearRegression
from scipy import stats

np.random.seed(42)
n_experiments = 1000
n_samples = 100
p_values = []

for i in range(n_experiments):
    W = np.random.normal(0, 1, n_samples)
    X = np.random.normal(0, 1, n_samples)
    Y = 2 * X + np.random.normal(0, 1, n_samples)

    predictors = np.column_stack([W, X])
    model = LinearRegression()
    model.fit(predictors, Y)

    predictions = model.predict(predictors)
    residuals = Y - predictions
    dof = n_samples - 3
    mse = np.sum(residuals**2) / dof

    X_with_intercept = np.column_stack([np.ones(n_samples), predictors])
    var_covar = mse * np.linalg.inv(X_with_intercept.T @ X_with_intercept)
    se_W = np.sqrt(var_covar[1, 1])

    t_stat = model.coef_[0] / se_W
    p_value = 2 * (1 - stats.t.cdf(abs(t_stat), dof))
    p_values.append(p_value)

p_values = np.array(p_values)
min_p_value = np.min(p_values)
num_significant = np.sum(p_values < 0.05)

print(f"Smallest p-value: {min_p_value:.6f}")
```

✓ 0.4s   Python

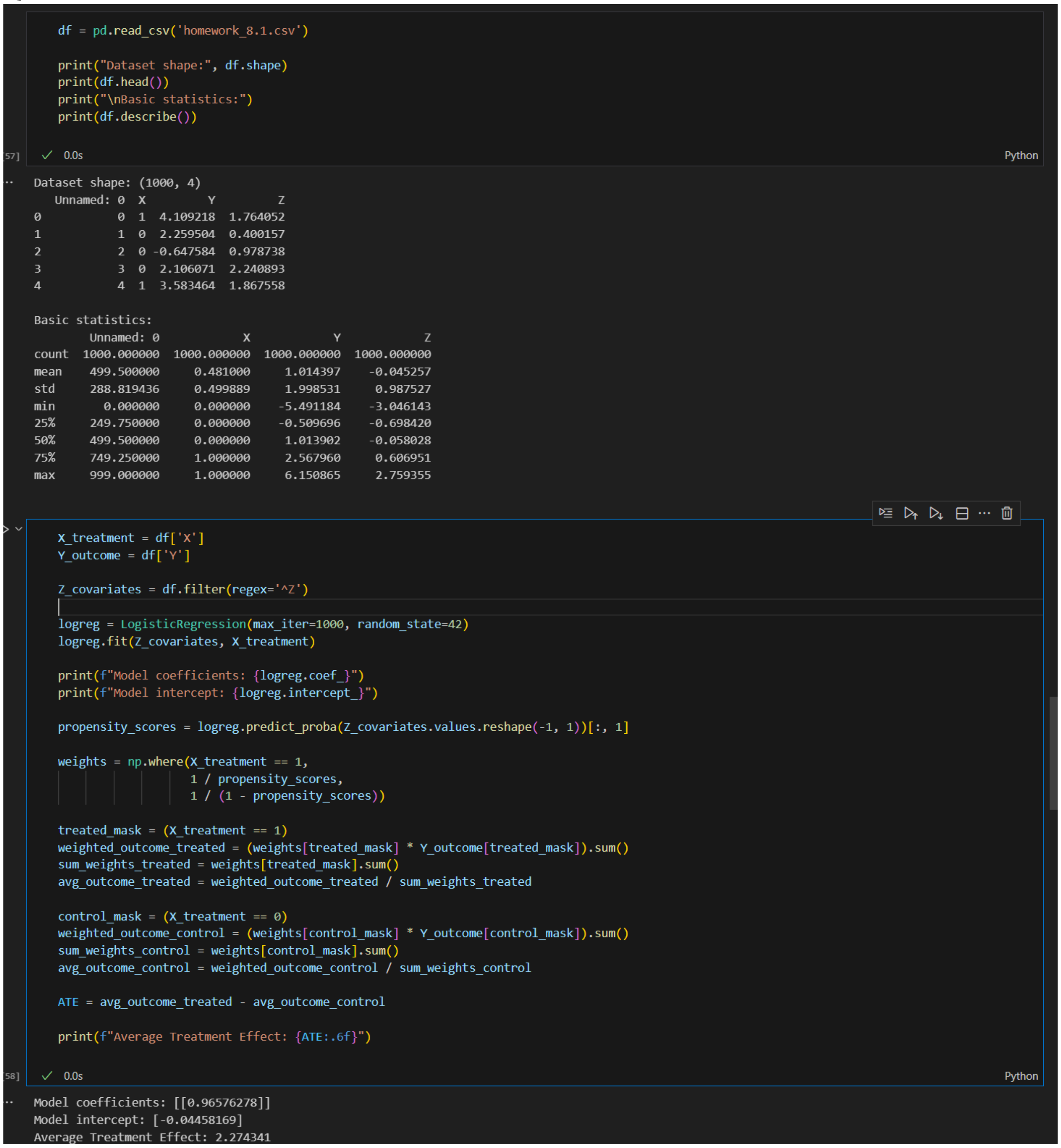
```
Smallest p-value: 0.000475
```

Even though W truly has zero effect on Y we still get experiments with p-values below 0.05. This does not mean W actually has an effect. When we run 1000 experiments and cherry pick the one with the smallest p-value we're guaranteed to find something that looks significant just by random chance. A p-value of 0.05 means there's a 5% chance of getting a result this extreme if the null hypothesis is true. If we run 1000 tests we expect around 50 to show p-values less than 0.05. Reporting only the most extreme one is misleading because we're ignoring all the times where nothing significant happened.

Homework Reflection 8

Include the code you used to solve the two coding quiz problems and write about the obstacles / challenges / insights you encountered while solving them.

Question 1:



```python
df = pd.read_csv('homework_8.1.csv')

print("Dataset shape:", df.shape)
print(df.head())
print("\nBasic statistics:")
print(df.describe())
```

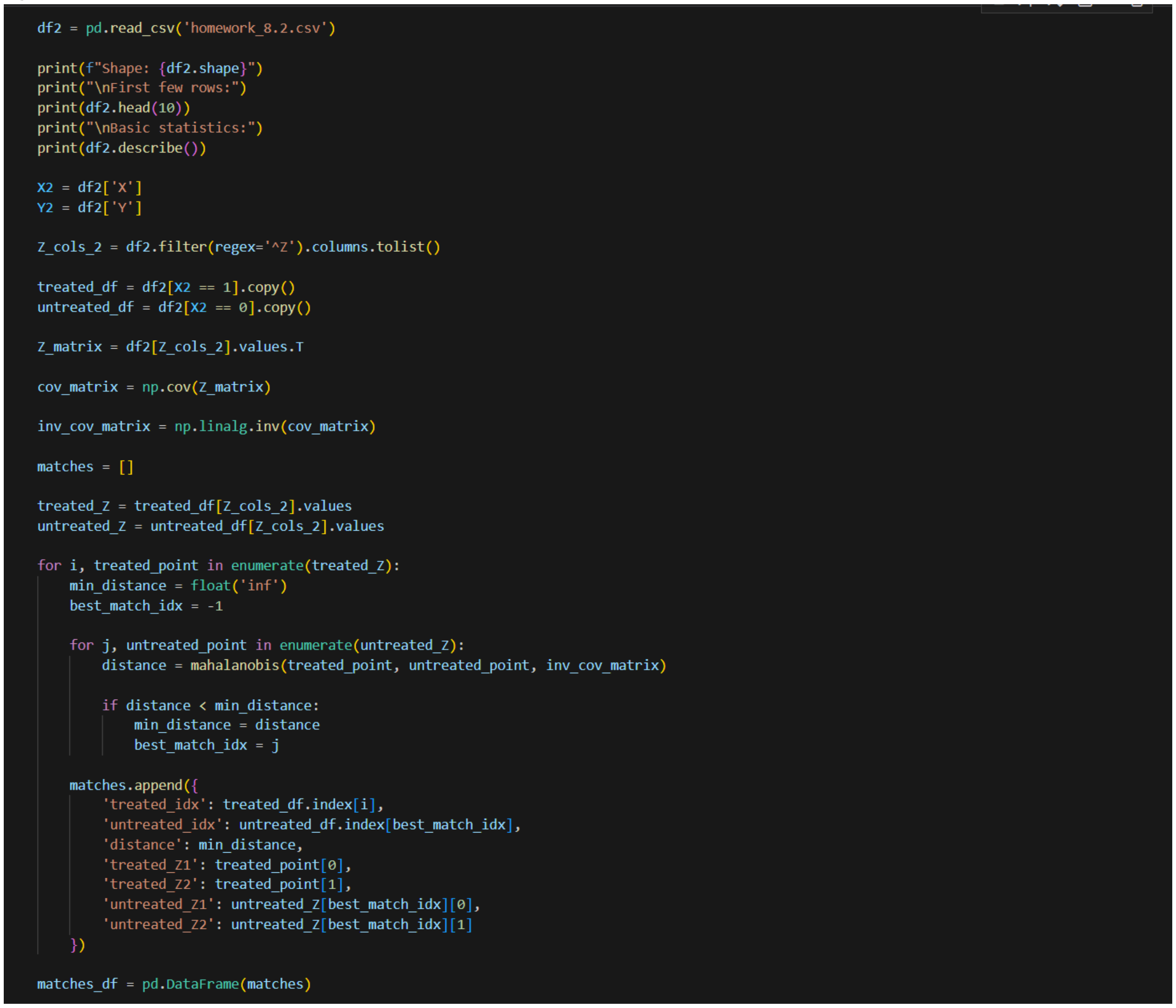
```
Dataset shape: (1000, 4)
   Unnamed: 0  X         Y         Z
0           0  1  4.109218  1.764052
1           1  0  2.259504  0.400157
2           2  0 -0.647584  0.978738
3           3  0  2.106071  2.240893
4           4  1  3.583464  1.867558

Basic statistics:
        Unnamed: 0            X            Y            Z
count  1000.000000  1000.000000  1000.000000  1000.000000
mean    499.500000     0.481000     1.014397    -0.045257
std     288.819436     0.499889     1.998531     0.987527
min       0.000000     0.000000    -5.491184    -3.046143
25%     249.750000     0.000000    -0.509696    -0.698420
50%     499.500000     0.000000     1.013902    -0.058028
75%     749.250000     1.000000     2.567960     0.606951
max     999.000000     1.000000     6.150865     2.759355
```

```python
X_treatment = df['X']
Y_outcome = df['Y']

Z_covariates = df.filter(regex='^Z')

logreg = LogisticRegression(max_iter=1000, random_state=42)
logreg.fit(Z_covariates, X_treatment)

print(f"Model coefficients: {logreg.coef_}")
print(f"Model intercept: {logreg.intercept_}")

propensity_scores = logreg.predict_proba(Z_covariates.values.reshape(-1, 1))[:, 1]

weights = np.where(X_treatment == 1,
                   1 / propensity_scores,
                   1 / (1 - propensity_scores))

treated_mask = (X_treatment == 1)
weighted_outcome_treated = (weights[treated_mask] * Y_outcome[treated_mask]).sum()
sum_weights_treated = weights[treated_mask].sum()
avg_outcome_treated = weighted_outcome_treated / sum_weights_treated

control_mask = (X_treatment == 0)
weighted_outcome_control = (weights[control_mask] * Y_outcome[control_mask]).sum()
sum_weights_control = weights[control_mask].sum()
avg_outcome_control = weighted_outcome_control / sum_weights_control

ATE = avg_outcome_treated - avg_outcome_control

print(f"Average Treatment Effect: {ATE:.6f}")
```

```
Model coefficients: [[0.96576278]]
Model intercept: [-0.04458169]
Average Treatment Effect: 2.274341
```

The first challenge was figuring out which columns were actually covariates. I used df.filter(regex='^Z') to get only columns starting with Z which avoided accidentally including the index column. This was important because including the wrong columns would mess up the propensity score model. The second challenge was understanding why we need inverse probability weights. The key insight is that we're trying to create a pseudo population where treatment assignment is random. Units that are unlikely to receive their actual treatment get higher weights to compensate.

Questions 2:



```python
df2 = pd.read_csv('homework_8.2.csv')

print(f"Shape: {df2.shape}")
print("\nFirst few rows:")
print(df2.head(10))
print("\nBasic statistics:")
print(df2.describe())

X2 = df2['X']
Y2 = df2['Y']

Z_cols_2 = df2.filter(regex='^Z').columns.tolist()

treated_df = df2[X2 == 1].copy()
untreated_df = df2[X2 == 0].copy()

Z_matrix = df2[Z_cols_2].values.T

cov_matrix = np.cov(Z_matrix)

inv_cov_matrix = np.linalg.inv(cov_matrix)

matches = []

treated_Z = treated_df[Z_cols_2].values
untreated_Z = untreated_df[Z_cols_2].values

for i, treated_point in enumerate(treated_Z):
    min_distance = float('inf')
    best_match_idx = -1

    for j, untreated_point in enumerate(untreated_Z):
        distance = mahalanobis(treated_point, untreated_point, inv_cov_matrix)

        if distance < min_distance:
            min_distance = distance
            best_match_idx = j

    matches.append({
        'treated_idx': treated_df.index[i],
        'untreated_idx': untreated_df.index[best_match_idx],
        'distance': min_distance,
        'treated_Z1': treated_point[0],
        'treated_Z2': treated_point[1],
        'untreated_Z1': untreated_Z[best_match_idx][0],
        'untreated_Z2': untreated_Z[best_match_idx][1]
    })

matches_df = pd.DataFrame(matches)
```

Mahalanobis distance accounts for the covariance structure of the data. If two variables are highly correlated then distance along their shared direction matters less than distance perpendicular to it. Another insight was about matching with replacement. Some untreated individuals got matched to multiple treated individuals. This makes sense because if an untreated person has common covariate values they might be the best match for several treated people. Without replacement we would be forced to use worse matches later on. With replacement we can reuse good matches which generally reduces the average matching distance.