Economics 600a
Yale University, Fall 2025
Marek Chadim[*]
Homework Solutions[†‡]

1 Overview

This assignment estimates demand and supply in a stylized model of the pay-TV market. A synthetic dataset is constructed to represent the industry, and a series of simple estimation exercises are performed. Using the `pyBLP` package by Conlon and Gortmaker (2020), the model is then estimated and applied to simulate potential merger scenarios.

2 Model

There are T markets, each with four inside goods $j \in \{1, 2, 3, 4\}$ and an outside option. Goods 1 and 2 are satellite television services (e.g., DirecTV and Dish); goods 3 and 4 are wired television services (e.g., Frontier and Comcast in New Haven). The conditional indirect utility of consumer i for good j in market t is given by

$$\begin{aligned} u_{ijt} &= \beta^{(1)} x_{jt} + \beta_i^{(2)} satellite_{jt} + \beta_i^{(3)} wired_{jt} + \alpha p_{jt} + \xi_{jt} + \epsilon_{ijt} \qquad j > 0 \\ u_{i0t} &= \epsilon_{i0t}, \end{aligned}$$

where x_{jt} is a measure of good j's quality, p_{jt} is its price, $satellite_{jt}$ is an indicator equal to 1 for the two satellite services, and $wired_{jt}$ is an indicator equal to 1 for the two wired services. The remaining notation is as usual in the class notes, including the i.i.d. type-1 extreme value ϵ_{ijt}. Each consumer purchases the good giving them the highest conditional indirect utility.

Goods are produced by single-product firms. Firm j's (log) marginal cost in market t is

$$\ln mc_{jt} = \gamma^0 + \mathrm{w}_{jt}\gamma^1 + \omega_{jt}/8,$$

where w_{jt} is an observed cost shifter. Firms compete by simultaneously choosing prices in each market under complete information. Firm j has profit

$$\pi_{jt} = \max_{p_{jt}} M_t(p_{jt} - mc_{jt}) s_{jt}(p_t).$$

3 Generate Fake Data

Generate a data set from the model above. Start with

$$\begin{aligned} \beta^{(1)} &= 1,\ \beta_i^{(k)} \sim \text{iid } N(4,1) \ \text{ for } k = 2, 3 \\ \alpha &= -2 \\ \gamma^{(0)} &= 1/2,\ \gamma^{(1)} = 1/4. \end{aligned}$$

[*]marek.chadim@yale.edu

[†]I acknowledge the use of AI tools for coding and typesetting assistance.

[‡]Replication notebook available at https://github.com/marek-chadim/pyblp.

1. Draw the exogenous product characteristic x_{jt} for $T = 600$ geographically defined markets (e.g., cities). Assume each x_{jt} is equal to the absolute value of an iid standard normal draw, as is each w_{jt}. Simulate demand and cost unobservables as well, specifying

$$\begin{pmatrix} \xi_{jt} \\ \omega_{jt} \end{pmatrix} \sim N\left(\begin{pmatrix} 0 \\ 0 \end{pmatrix}, \begin{pmatrix} 1 & 0.25 \\ 0.25 & 1 \end{pmatrix} \right) \text{ iid across } j, t.$$

```
np.random.seed(1995)

# Model parameters
T, J = 600, 4
alpha, beta1 = -2, 1
beta2, beta3 = 4, 4
sigma_satellite, sigma_wired = 1, 1
gamma0, gamma1 = 0.5, 0.25

# Product data structure
data = [{'market_ids': t, 'firm_ids': j+1, 'product_ids': j}
        for t in range(T) for j in range(J)]
product_data = pd.DataFrame(data)

# Exogenous variables: x_jt and w_jt as absolute values of iid standard normal draws
product_data['x'] = np.abs(np.random.normal(0, 1, len(product_data)))
product_data['w'] = np.abs(np.random.normal(0, 1, len(product_data)))
# Indicators
product_data['satellite'] = (product_data['firm_ids'].isin([1, 2]).astype(int))
product_data['wired'] = (product_data['firm_ids'].isin([3, 4]).astype(int))

# Unobservables: xi_jt and omega_jt with covariance matrix [[1, 0.25], [0.25, 1]]
cov_matrix = np.array([[1, 0.25], [0.25, 1]])
A = np.linalg.cholesky(cov_matrix)
z = np.random.normal(0, 1, (len(product_data), 2))
unobs = z @ A.T
product_data['xi'] = unobs[:, 0]  # demand unobservable
product_data['omega'] = unobs[:, 1]  # cost unobservable
```

2. Solve for the equilibrium prices for each good in each market.

   (a) Start by writing a procedure to approximate the derivatives of market shares with respect to prices (taking prices, shares, x, and demand parameters as inputs). The key steps are:

      i. For each (j, t), write the choice probability s_{jt} as a weighted average (integral) of the (multinomial logit) choice probabilities conditional on the value of each consumer's random coefficients;

      Answer:
      From the utility specification, we write:

$$u_{ijt} = \delta_{jt} + \nu_{ijt}$$

where mean utility and individual deviation are defined as follows:

$$\delta_{jt} = \beta^{(1)} x_{jt} + \alpha p_{jt} + \xi_{jt},$$
$$\nu_{ijt} = \beta_i^{(2)} satellite_{jt} + \beta_i^{(3)} wired_{jt} + \epsilon_{ijt}.$$

Given the random coefficients $\beta_i = (\beta_i^{(2)}, \beta_i^{(3)})$ and assuming that ϵ_{ijt} follows a Type-1 extreme value distribution, the conditional choice probability takes the multinomial logit form (Berry et al., 1995):

$$s_{jt}|\beta_i = \frac{\exp(\delta_{jt} + \beta_i^{(2)} satellite_{jt} + \beta_i^{(3)} wired_{jt})}{1 + \sum_{k=1}^{4} \exp(\delta_{kt} + \beta_i^{(2)} satellite_{kt} + \beta_i^{(3)} wired_{kt})},$$

where the denominator includes the outside option normalized to zero utility.
The unconditional market share is the integral over the distribution of random coefficients:

$$s_{jt} = \int \frac{\exp(\delta_{jt} + \beta_i^{(2)} satellite_{jt} + \beta_i^{(3)} wired_{jt})}{1 + \sum_{k=1}^{4} \exp(\delta_{kt} + \beta_i^{(2)} satellite_{kt} + \beta_i^{(3)} wired_{kt})} f(\beta_i^{(2)}, \beta_i^{(3)})\, d\beta_i^{(2)} d\beta_i^{(3)},$$

where $f(\cdot)$ denotes the joint density of $(\beta_i^{(2)}, \beta_i^{(3)})$, each distributed independently as $\mathcal{N}(4, 1)$.

ii. Anticipating differentiation under the integral sign, derive the analytical expression for the derivative of the *integrand* with respect to each p_{kt};
Answer:
Let the integrand be denoted

$$I_{jt}(\beta_i) \;=\; \frac{\exp(\delta_{jt} + \beta_i^{(2)}\; satellite_{jt} + \beta_i^{(3)}\; wired_{jt})}{1 + \sum_{\ell=1}^{4} \exp(\delta_{\ell t} + \beta_i^{(2)}\; satellite_{\ell t} + \beta_i^{(3)}\; wired_{\ell t})}.$$

Since $\delta_{\ell t} = \beta^{(1)} x_{\ell t} + \alpha p_{\ell t} + \xi_{\ell t}$, we have $\dfrac{\partial \delta_{\ell t}}{\partial p_{kt}} = \alpha \mathbf{1}\{\ell = k\}$. Differentiating the multinomial logit integrand with respect to p_{kt} (anticipating differentiation under the integral sign), and using the usual logit algebra, yields the compact expression

$$\frac{\partial I_{jt}(\beta_i)}{\partial p_{kt}} = \alpha\, I_{jt}(\beta_i)\Big(\mathbf{1}\{j = k\} - I_{kt}(\beta_i)\Big),$$

where

$$I_{kt}(\beta_i) = \frac{\exp(\delta_{kt} + \beta_i^{(2)} satellite_{kt} + \beta_i^{(3)} wired_{kt})}{1 + \sum_{\ell=1}^{4} \exp(\delta_{\ell t} + \beta_i^{(2)} satellite_{\ell t} + \beta_i^{(3)} wired_{\ell t})}.$$

Equivalently, for the own- and cross-price cases:

$$\frac{\partial I_{jt}}{\partial p_{jt}} = \alpha\, I_{jt}(1 - I_{jt}), \qquad \frac{\partial I_{jt}}{\partial p_{kt}} = -\alpha\, I_{jt}\, I_{kt} \quad (k \neq j).$$

(Justification: differentiation under the integral sign is permitted here because the integrand and its derivative are bounded for each β_i and the mixing density $f(\beta_i)$ is integrable; dominated convergence applies.)

iii. Use the expression you obtained in (2) and simulation draws of the random coefficients to approximate the integral that corresponds to $\partial s_{jt}/\partial p_{kt}$ for each j and k (i.e., replace the integral with the mean over the values at each simulation draw). Recall the advice in the lecture regarding "jittering."

```
def market_shares_and_derivatives(prices, market_data, nu_draws):
    J = len(market_data)
    x = market_data['x'].values
    xi = market_data['xi'].values
    sat = market_data['satellite'].values
    wired = market_data['wired'].values

    # Compute utilities once
    utilities = (
        beta1 * x + xi +
        nu_draws[:, 0:1] * sat +
        nu_draws[:, 1:2] * wired +
        alpha * prices
    )
    utilities = np.column_stack([utilities, np.zeros(nu_draws.shape[0])])
    exp_u = np.exp(utilities - np.max(utilities, axis=1, keepdims=True))
    choice_probs = exp_u / exp_u.sum(axis=1, keepdims=True)
    inside_shares_draws = choice_probs[:, :J]

    # Shares: average over draws
    shares = np.mean(inside_shares_draws, axis=0)

    # Derivatives: compute analytically from choice probabilities
    derivatives = np.zeros((J, J))
    for j in range(J):
        for k in range(J):
            indicator = float(j == k)
            deriv_draws = (
                alpha * inside_shares_draws[:, j] *
                (indicator - inside_shares_draws[:, k])
            )
            derivatives[j, k] = np.mean(deriv_draws)

    return shares, derivatives, inside_shares_draws
```

To ensure reproducibility and prevent convergence problems caused by stochastic jitter across iterations, the simulation draws for the random coefficients are generated once (with a fixed seed) and reused throughout the procedure rather than redrawn at each call.

```
all_nu_draws = [np.random.multivariate_normal([beta2, beta3],
np.diag([sigma_satellite, sigma_wired]), size=10000) for _ in range(T)]
```

iv. Experiment to see how many simulation draws you need to get precise approximations and check this again at the equilibrium shares and prices you obtain below.

Answer:

I test convergence by computing derivatives multiple times with different random draws and measuring the standard deviation across repetitions:

```
def test_convergence(prices, market_data, nu_draws_full,
                     draw_counts, n_reps=100):
    stds = []
    n_available = len(nu_draws_full)
    for n_draws in draw_counts:
        deriv_list = []
        for rep in range(n_reps):
            # Randomly sample n_draws from pre-drawn samples
            indices = np.random.choice(n_available, size=n_draws,
                                       replace=False)
            nu_draws = nu_draws_full[indices]
            _, derivs, _ = market_shares_and_derivatives(
                prices, market_data, nu_draws
            )
            deriv_list.append(derivs)
        stds.append(np.std(deriv_list, axis=0).mean())
    return np.array(stds)
# Test at initial prices (p = MC) for market 0
product_data['mc'] = np.exp(
    gamma0 + gamma1 * product_data['w'] + product_data['omega'] / 8
)
market_0 = product_data[product_data['market_ids'] == 0]
prices_init = market_0['mc'].values
draw_counts = [50, 100, 200, 500, 1000, 2000, 5000]
stds = test_convergence(prices_init, market_0, all_nu_draws[0], draw_counts)
```

Convergence Results at Initial Prices:

Draws	Mean Std Dev	Rel. Change
50	0.0080	–
100	0.0054	33.2%
200	0.0037	30.6%
500	0.0025	32.0%
1000	0.0017	34.8%
2000	0.0011	35.1%
5000	0.0005	51.0%

The standard deviation declines monotonically with the number of simulation draws. With 5,000 draws, the standard deviation is 0.0005, which provides a sufficiently precise approximation for practical purposes.

(b) The FOC for firm j's profit maximization problem in market t is

$$(p_{jt} - mc_{jt})\frac{\partial s_{jt}(p_t)}{\partial p_{jt}} + s_{jt} = 0$$

$$\implies p_{jt} - mc_{jt} = -\left(\frac{\partial s_{jt}(p_t)}{\partial p_{jt}}\right)^{-1} s_{jt} \tag{1}$$

(c) Substituting in your approximation of each $\left(\frac{\partial s_{jt}(p_t)}{\partial p_{jt}}\right)$, solve the system of equations (1) (J equations per market) for the equilibrium prices in each market.

i. To do this you will need to solve a system of $J \times J$ nonlinear equations. Make sure to check the exit flag for each market to make sure you have a solution.

```
def solve_prices_direct(market_data, mc_market, nu_draws):
    J = len(market_data)
    def foc_residual(prices):
        # Compute shares and derivatives at current prices
        shares, derivatives, _ = market_shares_and_derivatives(
            prices, market_data, nu_draws
            )
        # Inversion of derivative matrix
        invD = np.linalg.inv(derivatives)
        # FOC residuals:
        residuals = prices - mc_market + invD @ shares
        return residuals
    # Initial guess: marginal costs
    p0 = mc_market.copy()
    # Solve using root finder (hybr method)
    sol = opt.root(foc_residual, p0, method='hybr', tol=1e-8)
    prices_sol = sol.x
    success = sol.success
    return prices_sol, success
# Solve using direct method
equilibrium_prices_direct = []
success_flags_direct = []
for t in range(T):
    market_data = product_data[product_data['market_ids'] == t]
    mc_market = market_data['mc'].values
    nu_draws = all_nu_draws[t]
    prices_direct, success = solve_prices_direct(
        market_data, mc_market, nu_draws
        )
    equilibrium_prices_direct.append(prices_direct)
    success_flags_direct.append(success)
equilibrium_prices_direct = np.array(equilibrium_prices_direct)
success_count = sum(success_flags_direct)
```

Results: Successfully solved 600/600 markets.

ii. Do this again using the algorithm of Morrow and Skerlos (2011), discussed in section 3.6 of Conlon and Gortmaker (2020) (and in the `pyBLP` "problem simulation tutorial"). Use the numerical integration approach you used in step (a) to approximate the terms defined in equation (25) of Conlon and Gortmaker (2020). If you get different results using this method, resolve this discrepancy either by correcting your code or explaining why your preferred method is the one to be trusted.

```
def solve_prices_morrow_skerlos(market_data,
mc_market, nu_draws, max_iter=100, tol=1e-6):
    prices = mc_market.copy()
    for iteration in range(max_iter):
        shares, derivatives, inside_shares_draws =
        market_shares_and_derivatives(prices, market_data, nu_draws)
        Lambda = np.diag(alpha * shares)
        Gamma =
        alpha * (inside_shares_draws.T @ inside_shares_draws) / nu_draws.shape[0]
        diff = prices - mc_market
        zeta = np.linalg.solve(Lambda, Gamma.T @ diff - shares)
        prices_new = mc_market + zeta
        foc_residual = Lambda @ (prices - mc_market - zeta)
        if np.max(np.abs(foc_residual)) < tol:
            break
        prices = 0.5 * prices + 0.5 * prices_new
    return prices, iteration + 1

# Solve using Morrow-Skerlos method
equilibrium_prices_ms = []
iterations_ms = []
for t in range(T):
    market_data = product_data[product_data['market_ids'] == t]
    mc_market = market_data['mc'].values
    nu_draws = all_nu_draws[t]
    prices_ms, iters = solve_prices_morrow_skerlos(market_data, mc_market, nu_draws)
    equilibrium_prices_ms.append(prices_ms)
    iterations_ms.append(iters)
equilibrium_prices_ms = np.array(equilibrium_prices_ms)

# Compare direct vs Morrow-Skerlos
    price_diff = np.abs(np.array(equilibrium_prices_direct) - equilibrium_prices_ms)

# Use Morrow-Skerlos prices
product_data['prices'] = equilibrium_prices_ms.flatten()
```

Results: Successfully solved 600/600 markets.

The Morrow-Skerlos method produces virtually identical results to the direct method (differences of order 10^{-6}).

```
# Compare derivative convergence at initial vs equilibrium prices
prices_equilibrium = market_0['prices'].values
initial_stds = stds
n_available = len(all_nu_draws[0])
n_reps = 100
eq_stds = []
for n_draws in draw_counts:
    deriv_list = []
    for _ in range(n_reps):
        indices = np.random.choice(n_available, size=n_draws, replace=False)
        nu_draws = all_nu_draws[0][indices]
        _, derivatives, _ = market_shares_and_derivatives(
            prices_equilibrium, market_0, nu_draws)
        deriv_list.append(derivatives)
    eq_stds.append(np.std(deriv_list, axis=0).mean())
eq_stds = np.array(eq_stds)
ratios = eq_stds / initial_stds
avg_ratio = np.mean(ratios)
```

Results: The derivative approximation via Monte Carlo integration is more stable at equilibrium prices compared to initial prices (marginal costs), as the ratio remains consistently around 0.68 across all draw counts.

Draws	Initial Std Dev	Equilibrium Std Dev	Ratio (Eq/Init)
50	0.0080	0.0053	0.66
100	0.0054	0.0036	0.68
200	0.0037	0.0027	0.71
500	0.0025	0.0017	0.68
1000	0.0017	0.0011	0.69
2000	0.0011	0.0007	0.68
5000	0.0005	0.0004	0.67

3. Calculate "observed" shares for your fake data set using your parameters, your draws of x,w, β_i, ω, ξ, and your equilibrium prices.

```
observed_shares = []
for t in range(T):
    market_data = product_data[product_data['market_ids'] == t]
    prices_market = market_data['prices'].values
    # Use pre-drawn simulation draws for this market
    shares_market, _, _ = market_shares_and_derivatives(
        prices_market, market_data, all_nu_draws[t])
    observed_shares.extend(shares_market)
product_data['shares'] = observed_shares
```

Results: Share range: 0.000 to 0.725, mean: 0.136 (std: 0.122). Market share sums average 0.543 (min: 0.303, max: 0.746), implying outside shares average 0.457. Average satellite product share: 0.135; average wired product share: 0.136.

4. Below you'll be using x and w as instruments in the demand estimation. Check whether these appear to be good instruments in your fake data using some regressions of prices and market shares on the exogenous variables (or some function of them; see the related discussion in the coding tips). If you believe the instruments are not providing enough variation, modify the parameter choices above until you are satisfied. Report your final choice of parameters and the results you rely on to conclude that the instruments seem good enough.

```
# Create instruments: quadratics, interactions, competitor sums, within-nest
product_data['x**2'] = product_data['x'] ** 2
product_data['w**2'] = product_data['w'] ** 2
product_data['x*w'] = product_data['x'] * product_data['w']
product_data['sum_x_competitors'] =
product_data.groupby('market_ids')['x'].transform('sum') - product_data['x']
product_data['sum_w_competitors'] =
product_data.groupby('market_ids')['w'].transform('sum') - product_data['w']
product_data['x_other_in_nest'] =
product_data.groupby(['market_ids','satellite'])['x'].transform('sum')-product_data['x']
product_data['w_other_in_nest'] =
product_data.groupby(['market_ids','satellite'])['w'].transform('sum')-product_data['w']

Z = product_data[['satellite', 'wired', 'x', 'w', 'x**2', 'w**2', 'x*w',
                  'sum_x_competitors', 'sum_w_competitors',
                  'x_other_in_nest', 'w_other_in_nest']]

# Test instrument validity
price_model = sm.OLS(product_data['prices'], Z).fit()
share_model = sm.OLS(product_data['shares'], Z).fit()
xi_model = sm.OLS(product_data['xi'], Z).fit()
omega_model = sm.OLS(product_data['omega'], Z).fit()

# Joint F-tests for excluded instruments
excluded_vars = ['w', 'x**2', 'w**2', 'x*w', 'sum_x_competitors',
                 'sum_w_competitors', 'x_other_in_nest', 'w_other_in_nest']
hypothesis = ', '.join([f'{var}=0' for var in excluded_vars])
price_f_test = price_model.f_test(hypothesis)
share_f_test = share_model.f_test(hypothesis)
xi_f_test = xi_model.f_test(hypothesis)
omega_f_test = omega_model.f_test(hypothesis)
```

Results: The parameters $\alpha = -2$, $\beta^{(1)} = 1$, $\beta_i^{(2)} \sim N(4, 1^2)$, $\beta_i^{(3)} \sim N(4, 1^2)$, $\gamma^{(0)} = 0.5$, $\gamma^{(1)} = 0.25$ were retained as final. The excluded demand instruments (w, x^2, w^2, xw, competitor sums, within-nest variables) are relevant based on the first-stage regressions: for prices, $R^2 = 0.511$, $F = 301.04$ ($p < 0.001$); for shares, $R^2 = 0.364$, $F = 91.06$ ($p < 0.001$). The instruments are also uncorrelated with the structural errors: for ξ, $F = 0.63$ ($p = 0.75$); for ω, $F = 0.58$ ($p = 0.80$).

4 Estimate Some Mis-specified Models

5. Estimate the multinomial logit model of demand by OLS (ignoring the endogeneity of prices).

```
# Compute logit delta: ln(s_jt / s_0t)
product_data['logit_delta'] =
np.log(product_data['shares'] / product_data['outside_share'])

# OLS: beta_hat = (X^T X)^(-1) X^T y
y = product_data['logit_delta'].values
X = product_data[['prices', 'x', 'satellite', 'wired']].values
beta_hat = np.linalg.inv(X.T @ X) @ X.T @ y

# HC0 robust standard errors
residuals = y - X @ beta_hat
V = X.T @ np.diag(residuals**2) @ X
cov_matrix_ols = np.linalg.inv(X.T @ X) @ V @ np.linalg.inv(X.T @ X)
se_ols = np.sqrt(np.diag(cov_matrix_ols))
```

Results:

OLS estimates for $\ln(s_{jt}/s_{0t}) = \alpha p_{jt} + \beta^{(1)} x_{jt} + \beta^{(2)} satellite_{jt} + \beta^{(3)} wired_{jt}$ (no intercept):

Parameter	Estimate	Std. Error	t-stat	p-value
α (prices)	-1.246	0.051	-24.41	0.000
$\beta^{(1)}$ (x)	0.854	0.032	26.36	0.000
$\beta^{(2)}$ (satellite)	1.757	0.165	10.67	0.000
$\beta^{(3)}$ (wired)	1.790	0.164	10.90	0.000

6. Re-estimate the multinomial logit model of demand by two-stage least squares, instrumenting for prices with the exogenous demand shifters x and excluded cost shifters w. Discuss the predictive value of the instruments in the first stage, and how the second-stage results differ from those obtained by OLS.

```
# First stage: regress prices on instruments
Z = product_data[['satellite', 'wired', 'x', 'w', 'x**2', 'w**2',
                  'x*w', 'sum_x_competitors', 'sum_w_competitors']].values
sigma_hat = np.linalg.inv(Z.T @ Z) @ Z.T @ product_data['prices'].values
prices_hat = Z @ sigma_hat
resid_fs = product_data['prices'].values - prices_hat
SST = np.sum((product_data['prices'].values - product_data['prices'].mean())**2)
SSR = np.sum(resid_fs**2)
R2_first = 1 - SSR/SST
# Second stage
X_hat = np.column_stack([prices_hat, product_data['x'].values,
         product_data['satellite'].values, product_data['wired'].values])
```

```
beta_hat_iv = np.linalg.inv(X_hat.T @ X_hat) @ X_hat.T @ y
# HC0 robust SE for 2SLS
X = np.column_stack([product_data['prices'].values, product_data['x'].values,
     product_data['satellite'].values, product_data['wired'].values])
resid_iv = y - X @ beta_hat_iv
P_Z = Z @ np.linalg.inv(Z.T @ Z) @ Z.T
Omega = np.diag(resid_iv**2)
XPZ = X.T @ P_Z
cov_iv = np.linalg.inv(XPZ @ X) @ XPZ @ Omega @ P_Z @ X @ np.linalg.inv(XPZ @ X)
se_iv = np.sqrt(np.diag(cov_iv))
```

Results: The first-stage $R^2 = 0.507$ confirms strong predictive power of the instruments. Second-stage 2SLS estimates are:

Parameter	Estimate	Std. Error	t-stat	p-value
α (prices)	-1.938	0.064	-30.51	0.000
$\beta^{(1)}$ (x)	0.923	0.035	26.15	0.000
$\beta^{(2)}$ (satellite)	3.993	0.208	19.21	0.000
$\beta^{(3)}$ (wired)	4.034	0.209	19.34	0.000

2SLS recovers parameters close to the truth ($\alpha = -2$, $\beta^{(1)} = 1$, $\beta^{(2)}, \beta^{(3)} \sim \mathcal{N}(4,1)$). OLS severely underestimates because equilibrium pricing implies each p_{jt} depends on all demand shocks $\xi_t = (\xi_{1t}, \ldots, \xi_{Jt})$ and all cost shocks $\omega_t = (\omega_{1t}, \ldots, \omega_{Jt})$, both through first-order conditions and strategic complementarity (Berry and Haile, 2021). The covariance $\text{Cov}(\xi_{jt}, \omega_{jt}) = 0.25$ creates positive correlation between ξ_{jt} and p_{jt}, biasing $\hat{\alpha}_{\text{OLS}}$ toward zero. Instruments shift equilibrium prices while satisfying $E[\xi_{jt}|x_t, w_t] = 0$, breaking the endogeneity between prices and demand unobservables.

7. Now estimate a nested logit model by two-stage least squares, treating "satellite" and "wired" as the two nests for the inside goods. Allow a different nesting parameter for each nest.

```
# Within-group shares for nesting
product_data["group_share"] = product_data.groupby(
    ["market_ids", "satellite"])["shares"].transform("sum")
product_data["ln_within_share"] = np.log(
    product_data["shares"] / product_data["group_share"])

# Nest-specific within-group share variables
product_data["ln_within_share_sat"] = (
    product_data["ln_within_share"] * product_data["satellite"])
product_data["ln_within_share_wired"] = (
    product_data["ln_within_share"] * product_data["wired"])
```

```
# First stage: instrument for prices, ln_within_share_sat, ln_within_share_wired
Z = product_data[['x', 'satellite', 'wired', 'w', 'x**2', 'w**2', 'x*w',
                  'sum_x_competitors', 'sum_w_competitors',
                  'x_other_in_nest', 'w_other_in_nest']].values
endog_vars = ['prices', 'ln_within_share_sat', 'ln_within_share_wired']
endog_hat = np.zeros((len(product_data), 3))
for i, var in enumerate(endog_vars):
    sigma = np.linalg.inv(Z.T @ Z) @ Z.T @ product_data[var].values
    endog_hat[:, i] = Z @ sigma

# Second stage
X_hat = np.column_stack([endog_hat[:, 0], product_data['x'].values,
         product_data['satellite'].values, product_data['wired'].values,
         endog_hat[:, 1], endog_hat[:, 2]])
beta_hat_nl = np.linalg.inv(X_hat.T @ X_hat) @ X_hat.T @ y

# HC0 robust SE for 2SLS
X = np.column_stack([product_data['prices'].values, product_data['x'].values,
     product_data['satellite'].values, product_data['wired'].values,
     product_data['ln_within_share_sat'].values,
     product_data['ln_within_share_wired'].values])
resid = y - X @ beta_hat_nl
P_Z = Z @ np.linalg.inv(Z.T @ Z) @ Z.T
XPZ = X.T @ P_Z
cov_nl =
np.linalg.inv(XPZ @ X) @ XPZ @ np.diag(resid**2) @ P_Z @ X @ np.linalg.inv(XPZ @ X)
se_nl = np.sqrt(np.diag(cov_nl))
```

Results: Nested logit 2SLS estimates (prices, within-group shares instrumented):

Parameter	Estimate	Std. Error	t-stat	p-value
α (prices)	-1.605	0.097	-16.51	0.000
$\beta^{(1)}$ (x)	0.802	0.041	19.53	0.000
$\beta^{(2)}$ (satellite)	3.106	0.603	5.15	0.000
$\beta^{(3)}$ (wired)	3.348	0.484	6.92	0.000
ρ_{sat} (ln_within_sat)	0.115	0.449	0.26	0.797
ρ_{wired} (ln_within_wired)	0.312	0.465	0.67	0.502

The nesting parameters ($\rho_{\text{sat}} = 0.115$, $\rho_{\text{wired}} = 0.312$) are not significantly different from zero. This model imposes correlation through $\text{Cov}(\epsilon_{ijt}, \epsilon_{ikt} \mid i, t)$ for products j, k in the same nest, effectively modeling substitution patterns through the error structure. The true DGP instead generates substitution via random coefficients $\beta_i^{(2)}, \beta_i^{(3)} \sim \mathcal{N}(4, 1)$ on satellite/wired indicators, creating heterogeneous valuations where preference heterogeneity drives within-category substitution, not correlated shocks.

8. Using the nested logit results, provide a table comparing the estimated own-price elasticities to the true own-price elasticities. Provide two additional tables showing the true matrix of diversion ratios and the diversion ratios implied by your estimates.

```
def compute_nested_logit_elasticities(market_df, alpha, rho_sat, rho_wired):
    J = len(market_df)
    prices, shares = market_df['prices'].values, market_df['shares'].values
    sat, wired = market_df['satellite'].values, market_df['wired'].values
    # Within-nest shares
    s_group = market_df.groupby('satellite')['shares'].transform('sum').values
    s_within = shares / s_group
    rho = np.where(sat == 1, rho_sat, rho_wired)
    # Jacobian elements
    elast = np.zeros((J, J))
    for j in range(J):
        lambda_jj = alpha * shares[j] / (1 - rho[j])
        for k in range(J):
            same_nest = (sat[j] == sat[k]) and (wired[j] == wired[k])
            gamma_jk = alpha * shares[j] * shares[k]
            if same_nest:
                gamma_jk += (rho[j]/(1-rho[j])) * alpha * s_within[j] * shares[k]
            jac_jk = lambda_jj - gamma_jk if j == k else -gamma_jk
            elast[j,k] = jac_jk * prices[k] / shares[j]
    return elast

def compute_rc_elasticities_observed_shares(market_df, nu_draws, alpha, beta_x,
                                            beta_sat, beta_wired, sigma_sat, sigma_wired):
    J = len(market_df)
    prices = market_df['prices'].values
    x, observed_shares = market_df['x'].values, market_df['shares'].values
    sat, wired = market_df['satellite'].values, market_df['wired'].values
    rc_deviation = sigma_sat*nu_draws[:,0:1]*sat + sigma_wired*nu_draws[:,1:2]*wired
    delta = np.log(observed_shares)  # Initial guess
    for iteration in range(1000):
        utilities = delta[np.newaxis, :] + rc_deviation
        exp_utils = np.exp(utilities)
        choice_probs = exp_utils / (1 + exp_utils.sum(axis=1, keepdims=True))
        predicted_shares = choice_probs.mean(axis=0)
        delta_new = delta + np.log(observed_shares) - np.log(predicted_shares)
        if np.max(np.abs(delta_new - delta)) < 1e-14:
            delta = delta_new
            break
        delta = delta_new

    # Compute elasticities from backed-out choice probabilities
    utilities = delta[np.newaxis, :] + rc_deviation
    exp_utils = np.exp(utilities)
    choice_probs = exp_utils / (1 + exp_utils.sum(axis=1, keepdims=True))
    elast = np.zeros((J, J))
    for j in range(J):
        for k in range(J):
            if j == k:
                deriv = alpha * np.mean(choice_probs[:,j] * (1 - choice_probs[:,j]))
            else:
                deriv = -alpha * np.mean(choice_probs[:,j] * choice_probs[:,k])
            elast[j,k] = (prices[k] / observed_shares[j]) * deriv
    return elast

# Compute for all markets and average
alpha_nl, rho_sat_nl, rho_wired_nl = beta_hat_iv_nested[[0, 4, 5]]
nl_elast = [compute_nested_logit_elasticities(
    product_data[product_data['market_ids']==t],
    alpha_nl, rho_sat_nl, rho_wired_nl) for t in range(T)]
avg_nl = np.mean([np.diag(e) for e in nl_elast], axis=0)
true_elast = [compute_rc_elasticities_observed_shares(
    product_data[product_data['market_ids']==t], all_nu_draws[t],
    -2.0, 1.0, 4.0, 4.0, 1.0, 1.0) for t in range(T)]
avg_true = np.mean([np.diag(e) for e in true_elast], axis=0)

def compute_diversion_ratios(elasticity_matrices, product_data, T, J):
    diversion_matrices = []
    for t in range(T):
        elast_matrix = elasticity_matrices[t]
        market_data_t = product_data[product_data['market_ids'] == t]
        shares = market_data_t['shares'].values
        prices = market_data_t['prices'].values
        # Convert elasticities to Jacobian: ds_j/dp_k = (s_j/p_k) * e_jk
        jacobian = np.zeros((J, J))
        for j in range(J):
            for k in range(J):
                jacobian[j, k] = (shares[j] / prices[k]) * elast_matrix[j, k]
        # Replace diagonal with outside option derivative
        jacobian_diag = np.diag(jacobian).copy()
        np.fill_diagonal(jacobian, -jacobian.sum(axis=1))
        # Compute diversion: D_jk = -Jacobian[j,k] / Jacobian[j,j]
        diversion = -jacobian / jacobian_diag[:, None]
        diversion_matrices.append(diversion)
    return diversion_matrices

true_div = np.mean(compute_diversion_ratios(true_elast, product_data, T, J), axis=0)
nl_div = np.mean(compute_diversion_ratios(nl_elast, product_data, T, J), axis=0)
```

Results:

Elasticities are computed as $\varepsilon_{jk} = \frac{p_k}{s_j}\frac{\partial s_j}{\partial p_k}$.

True elasticities are computed using analytical derivatives of the random coefficients logit model (Berry et al., 1995) with observed shares. The method backs out mean utilities δ via contraction mapping (Berry, 1994) from observed shares. Given δ, individual choice probabilities are computed as $s_{ij} = \frac{\exp(\delta_j+\nu_{ij})}{1+\sum_k \exp(\delta_k+\nu_{ik})}$, where $\nu_{ij} = \sigma_{\text{sat}}\cdot\tilde{\beta}_i^{(2)}\cdot\text{satellite}_j+\sigma_{\text{wired}}\cdot\tilde{\beta}_i^{(3)}\cdot\text{wired}_j$ and $\tilde{\beta}_i^{(2)}, \tilde{\beta}_i^{(3)} \sim \mathcal{N}(0,1)$ are standardized draws. Derivatives follow: $\frac{\partial s_j}{\partial p_k} = \alpha\mathbb{E}[s_{ij}(1-s_{ij})]$ (own-price) and $\frac{\partial s_j}{\partial p_k} = -\alpha\mathbb{E}[s_{ij}s_{ik}]$ (cross-price), with expectations taken over 10,000 simulation draws per market.

Nested logit elasticities are computed using the Jacobian formulation (Conlon and Gortmaker, 2020; Conlon and Mortimer, 2021): Jacobian $= \text{diag}(\Lambda) - \Gamma$, where $\Lambda_{jj} = \frac{\alpha s_j}{1-\rho_j}$ and $\Gamma_{jk} = \alpha s_j s_k + \mathbb{I}_{\text{same nest}} \cdot \frac{\rho_j}{1-\rho_j}\alpha s_{j|g} s_k$, with $s_{j|g}$ denoting the within-nest share and ρ_j the nest-specific parameter.

True versus Estimated Own-Price Elasticities:

Product	True	Estimated (NL)
Satellite 1	−5.478	−4.998
Satellite 2	−5.253	−4.855
Wired 1	−5.337	−5.904
Wired 2	−5.463	−6.001

Elasticities are converted to derivatives $\frac{\partial s_j}{\partial p_k} = \frac{s_j}{p_k}\varepsilon_{jk}$. The diagonal is replaced with the outside option derivative $\frac{\partial s_0}{\partial p_j} = -\sum_k \frac{\partial s_k}{\partial p_j}$ (from the adding-up constraint), and diversion ratios are computed as $D_{jk} = -\frac{\partial s_k/\partial p_j}{\partial s_j/\partial p_j}$. Diversion to the outside good is reported on the diagonal instead of $D_{jj} = -1$.

True Diversion Ratios (Random Coefficients Logit):

	Satellite 1	Satellite 2	Wired 1	Wired 2
Satellite 1	0.525	0.245	0.118	0.111
Satellite 2	0.235	0.532	0.120	0.113
Wired 1	0.107	0.113	0.549	0.231
Wired 2	0.106	0.111	0.243	0.540

Estimated Diversion Ratios (Nested Logit):

	Satellite 1	Satellite 2	Wired 1	Wired 2
Satellite 1	0.508	0.200	0.152	0.141
Satellite 2	0.190	0.514	0.153	0.143
Wired 1	0.124	0.129	0.445	0.303
Wired 2	0.121	0.130	0.314	0.435

The nested logit overestimates diversion to the outside option and underestimates within-nest substitution (e.g., Satellite 1 to Satellite 2: 0.245 vs. 0.200). Random coefficients $\beta_i^{(2)}, \beta_i^{(3)} \sim \mathcal{N}(4,1)$ generate heterogeneous preferences and local competition. By contrast, the nested logit correlates the i.i.d. shocks ϵ_{ijt} within nests, failing to replicate heterogeneity-driven substitution: consumers with strong satellite preferences switch among satellites, while weaker-preference consumers choose the outside option.

5 Estimate the Correctly Specified Model

Use the pyBLP package to estimate the correctly specified model. Allow pyBLP to construct approximations to the optimal instruments, using the exogenous demand shifters and exogenous cost shifters.

9. Report a table with the estimates of the demand parameters and standard errors. Do this twice: once when you estimate demand alone, then again when you estimate jointly with supply.

```
# Formulations and instruments
X1_formulation = pyblp.Formulation('0 + prices + x + satellite + wired')
X2_formulation = pyblp.Formulation('0 + satellite + wired')
product_formulations1 = (X1_formulation, X2_formulation)
product_data['demand_instruments0'] = product_data['w']
product_data['demand_instruments1'] = product_data['x**2']
product_data['demand_instruments2'] = product_data['w**2']
product_data['demand_instruments3'] = product_data['x*w']
product_data['demand_instruments4'] = product_data['sum_x_competitors']
product_data['demand_instruments5'] = product_data['sum_w_competitors']
product_data['demand_instruments6'] = product_data['x_other_in_nest']
product_data['demand_instruments7'] = product_data['w_other_in_nest']
integration = pyblp.Integration('product', 10)

# Demand-only estimation with optimal instruments
problem1 = pyblp.Problem(product_formulations1, product_data,
                         integration=integration)
results1 = problem1.solve(sigma=np.eye(2), initial_update=True)
optimal_iv1 = results1.compute_optimal_instruments(seed=1995)
optimal_problem1 = optimal_iv1.to_problem()
optimal_iv_results1 = optimal_problem1.solve(sigma=np.eye(2), initial_update=True)

# Joint demand and supply estimation with additional optimal instruments
X3_formulation = pyblp.Formulation('1 + w')
product_formulations2 = (X1_formulation, X2_formulation, X3_formulation)
product_data['demand_instruments8'] = optimal_iv1.demand_instruments[:, 0]
product_data['demand_instruments9'] = optimal_iv1.demand_instruments[:, 1]
problem2 = pyblp.Problem(product_formulations2, product_data,
                         costs_type='log', integration=integration)
results2 = problem2.solve(sigma=np.eye(2), costs_bounds=(0.001, None),
                          beta=optimal_iv_results1.beta, initial_update=True)
# Re-estimate with new optimal demand instruments only
# avoid multicollinearity by not using 3 last columns: x, satellite and wired
optimal_iv2 = results2.compute_optimal_instruments(seed=1995)
columns_to_drop = [col for col in product_data.columns if 'instruments' in col]
product_data = product_data.drop(columns=columns_to_drop)
```

```
for i in range(optimal_iv2.demand_instruments.shape[1]-3):
    product_data[f'demand_instruments{i}'] =optimal_iv2.demand_instruments[:, i]
problem3 = pyblp.Problem(product_formulations2, product_data,
                         costs_type='log', integration=integration)
optimal_iv_results2 = problem3.solve(sigma=np.eye(2), beta=optimal_iv_results1.beta,
                                     costs_bounds=(0.001, None), initial_update=True)
```

Results:

Demand:

	Estimates		Std. Errors	
Parameter	Demand Only	Joint D & S	Demand Only	Joint D & S
α (prices)	-2.033	-2.022	0.068	0.069
$\beta^{(1)}$ (x)	1.011	1.017	0.045	0.045
$\beta^{(2)}$ (satellite)	3.910	3.864	0.229	0.236
$\beta^{(3)}$ (wired)	4.093	4.016	0.222	0.233
$\sigma_{satellite}$	1.349	1.372	0.227	0.231
σ_{wired}	1.002	1.109	0.255	0.253

Log-linear Marginal Costs (from joint estimation only):

Parameter	Estimate	Std. Error
$\gamma^{(0)}$ (constant)	0.793	0.016
$\gamma^{(1)}$ (w)	0.205	0.007

Both estimation approaches recover the true parameters well. The demand-only estimates are $\hat{\alpha} = -2.033$ (true: -2.0), $\hat{\beta}^{(1)} = 1.011$ (true: 1.0), with random coefficient means $\hat{\beta}^{(2)} = 3.910$, $\hat{\beta}^{(3)} = 4.093$ (true: 4.0 each), and standard deviations $\hat{\sigma}_{\text{satellite}} = 1.349$, $\hat{\sigma}_{\text{wired}} = 1.002$ (true: 1.0 each). Joint estimation produces nearly identical demand parameters while additionally recovering marginal cost parameters: $\hat{\gamma}^{(0)} = 0.793$ (true: 0.5) and $\hat{\gamma}^{(1)} = 0.205$ (true: 0.25). The cost parameter estimates differ somewhat from their true values, though both are precisely estimated with small standard errors.

Demand-only estimates are invariant to starting values for σ, while joint estimates are invariant to starting values for both σ and α. The close correspondence between demand-only and joint estimates indicates that supply-side moments $E[\omega_{jt} \cdot Z_t^s]$ contribute minimal additional information for demand parameter identification when optimal demand instruments are used.

10. Using your preferred estimates from the prior step (explain your preference), provide a table comparing the estimated own-price elasticities to the true own-price elasticities. Provide two additional tables showing the true matrix of diversion ratios and the diversion ratios implied by your estimates.

    Preferred Estimates: I use the joint demand and supply estimates. While standard errors are nearly identical across specifications, joint estimation produces $\hat{\alpha} = -2.022$, closer to the true value of -2.0 than the demand-only estimate of $\hat{\alpha} = -2.033$.

```
elasticities_rc_est2 = optimal_iv_results2.compute_elasticities()
avg_elasticities_rc_est2 = elasticities_rc_est2.reshape((T, J, J)).mean(axis=0)
own_elasticities_rc_est2 = np.diag(avg_elasticities_rc_est2)
diversion_rc_est2 = optimal_iv_results2.compute_diversion_ratios()
avg_diversion_rc_est2 = diversion_rc_est2.reshape((T, J, J)).mean(axis=0)
```

Own-Price Elasticities:

Product	True	Estimated
Satellite 1	-5.480	-5.354
Satellite 2	-5.315	-5.175
Wired 1	-5.377	-5.362
Wired 2	-5.458	-5.444

The estimated model underestimates own-price elasticity magnitudes by 1.4% on average. This reflects sampling variation in the parameter estimates, particularly in the random coefficient standard deviations ($\hat{\sigma}_{\text{satellite}} = 1.371$; $\hat{\sigma}_{\text{wired}} = 1.111$). Higher σ makes averaging over a wider distribution $\beta_i \sim \mathcal{N}(4, \sigma^2)$ produce smaller absolute derivatives, hence less elastic demands.

Diversion Ratios:

True Diversion (Random Coefficients Logit with True Parameters):

	Satellite 1	Satellite 2	Wired 1	Wired 2
Satellite 1	0.525	0.224	0.130	0.121
Satellite 2	0.214	0.532	0.131	0.123
Wired 1	0.124	0.130	0.532	0.214
Wired 2	0.122	0.129	0.226	0.523

Estimated Diversion (Joint Demand & Supply with Optimal Instruments):

	Satellite 1	Satellite 2	Wired 1	Wired 2
Satellite 1	0.499	0.265	0.123	0.114
Satellite 2	0.253	0.507	0.124	0.116
Wired 1	0.112	0.118	0.540	0.229
Wired 2	0.111	0.117	0.241	0.531

Despite the 37% overestimation in $\sigma_{\text{satellite}}$ and 11% overestimation in σ_{wired}, diversion ratios are close to the truth. This occurs because $D_{jk} = -(s_k/s_j) \cdot (\varepsilon_{kj}/\varepsilon_{jj})$ is a ratio: when σ increases, own-elasticity ε_{jj} becomes flatter (smaller $|\varepsilon_{jj}|$) and cross-elasticity ε_{kj} becomes flatter proportionally, leaving the ratio $\varepsilon_{kj}/\varepsilon_{jj}$ relatively stable (Conlon and Mortimer, 2021).

6 Merger Simulation

11. Suppose two of the four firms were to merge. Give a brief intuition for what theory tells us is likely to happen to the equilibrium prices of each good j.

Answer: Horizontal mergers typically increase equilibrium prices through unilateral effects. Pre-merger, each single-product firm j sets prices according to the first-order condition $p_{jt} = mc_{jt} - (\partial s_{jt}/\partial p_{jt})^{-1} s_{jt}$, treating demand diverted to competitors as lost profit. When firm j raises p_{jt}, consumers switch to competing products, creating a negative externality on j (lost sales) and a positive externality on competitor k (gained sales). Pre-merger, firm j ignores the benefit to k. Post-merger, the combined entity internalizes this externality. The merged firm's first-order condition for product j becomes $p_{jt} = mc_{jt} - (\partial s_{jt}/\partial p_{jt})^{-1}[s_{jt} + \sum_{k\in\mathcal{J}_{\text{merged}}\setminus\{j\}}(p_{kt} - mc_{kt})\partial s_{kt}/\partial p_{jt}]$. The additional term $\sum_{k\in\mathcal{J}_{\text{merged}}\setminus\{j\}}(p_{kt} - mc_{kt})\partial s_{kt}/\partial p_{jt}$ captures profit gained on product k when raising p_{jt} diverts consumers to k. Since $\partial s_{kt}/\partial p_{jt} > 0$ for substitutes and $(p_{kt} - mc_{kt}) > 0$, this term is positive, increasing the optimal p_{jt}.

Merging firms' products experience direct price increases from this recapture effect. Consumers switching between merged products now generate profit for the same firm. Non-merging firms' products also see price increases as best responses to the merged firms' higher prices, as their products become relatively more attractive when p_j, p_k rise. The magnitude of price increases depends on substitution patterns: larger effects occur when merging firms' products are closer substitutes (higher diversion ratios D_{jk}), as more consumers are recaptured within the merged entity. This is the primary unilateral effect of horizontal mergers.

12. Suppose firms 1 and 2 are proposing to merge. Use the `pyBLP` merger simulation procedure to provide a prediction of the post-merger equilibrium prices.

```
# Pre-merger baseline
product_labels = ['Satellite 1', 'Satellite 2', 'Wired 1', 'Wired 2']
pre_merger_prices = product_data['prices'].values
pre_merger_prices_avg = pre_merger_prices.reshape((T, J)).mean(axis=0)

# Create merger firm IDs: firms 1 and 2 merge into firm 1
merger_firm_ids = product_data['firm_ids'].copy()
merger_firm_ids[merger_firm_ids == 2] = 1

# Compute post-merger equilibrium prices
post_merger_prices = optimal_iv_results2.compute_prices(
firm_ids=merger_firm_ids, iteration=pyblp.Iteration('simple', {'atol': 1e-12}))

# Calculate price changes
post_merger_prices_avg = post_merger_prices.reshape((T, J)).mean(axis=0)
price_changes = post_merger_prices_avg - pre_merger_prices_avg
pct_price_changes = (price_changes / pre_merger_prices_avg) * 100

merger_results_df = pd.DataFrame({'Product': product_labels,
    'Firm (Pre)': [1, 2, 3, 4], 'Firm (Post)': [1, 1, 3, 4],
    'Pre-Merger Price': pre_merger_prices_avg,
    'Post-Merger Price': post_merger_prices_avg,
    'Price Change ($)': price_changes, 'Price Change (%)': pct_price_changes}
    )
```

Results:

Product	Firm (Pre)	Firm (Post)	Pre-Merger	Post-Merger	Change ($)	Change (%)
Satellite 1	1	1	3.3381	3.5499	0.2118	6.34
Satellite 2	2	1	3.2858	3.4852	0.1994	6.07
Wired 1	3	3	3.3210	3.3236	0.0026	0.08
Wired 2	4	4	3.3253	3.3278	0.0025	0.08

The within-nest merger of satellite providers (firms 1 and 2) increases their prices by slightly above 6% on average as the merged entity internalizes the competitive externality between its products. When firm 1 raises its price, consumers switching to the merged firm's other satellite product now generate profit for the same entity rather than a competitor, increasing optimal markups. Non-merging wired providers (firms 3 and 4) raise prices by only 0.08% as best responses to the satellite price increases, exploiting the shift in relative attractiveness. The price effects are substantially larger for merging firms because they directly internalize diversion, while non-merging firms only respond strategically through cross-price elasticities. The modest asymmetry between Satellite 1 (6.34%) and Satellite 2 (6.07%) reflects slight differences in pre-merger market positions and diversion patterns.

13. Now suppose instead that firms 1 and 3 are the ones to merge. Re-run the merger simulation. Provide a table comparing the (average across markets) predicted merger-induced price changes for this merger and that in part 11. Interpret the differences between the predictions for the two mergers.

```python
# Create merger firm IDs: firms 1 and 3 merge into firm 1
merger_firm_ids_cross = product_data['firm_ids'].copy()
merger_firm_ids_cross[merger_firm_ids_cross == 3] = 1

# Compute post-merger equilibrium prices for cross-nest merger
post_merger_prices_cross = optimal_iv_results2.compute_prices(
firm_ids=merger_firm_ids_cross, iteration=pyblp.Iteration('simple', {'atol': 1e-12}))

# Calculate price changes
post_merger_prices_cross_avg = post_merger_prices_cross.reshape((T, J)).mean(axis=0)
pct_price_changes_cross = ((post_merger_prices_cross_avg - pre_merger_prices_avg)
                           / pre_merger_prices_avg * 100)

# Comparison table
comparison_df = pd.DataFrame({
    'Product': product_labels,
    'Within-Nest (%)': pct_price_changes,      # From Q12
    'Cross-Nest (%)': pct_price_changes_cross,
    'Difference (pp)': pct_price_changes_cross - pct_price_changes
})
```

Results:

Cross-Nest Merger (Firms 1 & 3):

Product	Firm (Pre)	Firm (Post)	Pre-Merger	Post-Merger	Change ($)	Change (%)
Satellite 1	1	1	3.3381	3.4218	0.0837	2.51
Satellite 2	2	2	3.2858	3.2902	0.0044	0.13
Wired 1	3	1	3.3210	3.4006	0.0796	2.40
Wired 2	4	4	3.3253	3.3287	0.0035	0.10

Comparison: Within-Nest (1&2) vs Cross-Nest (1&3):

Product	Within-Nest (%)	Cross-Nest (%)	Difference (pp)
Satellite 1	6.34	2.51	−3.84
Satellite 2	6.07	0.13	−5.94
Wired 1	0.08	2.40	2.32
Wired 2	0.08	0.10	0.03

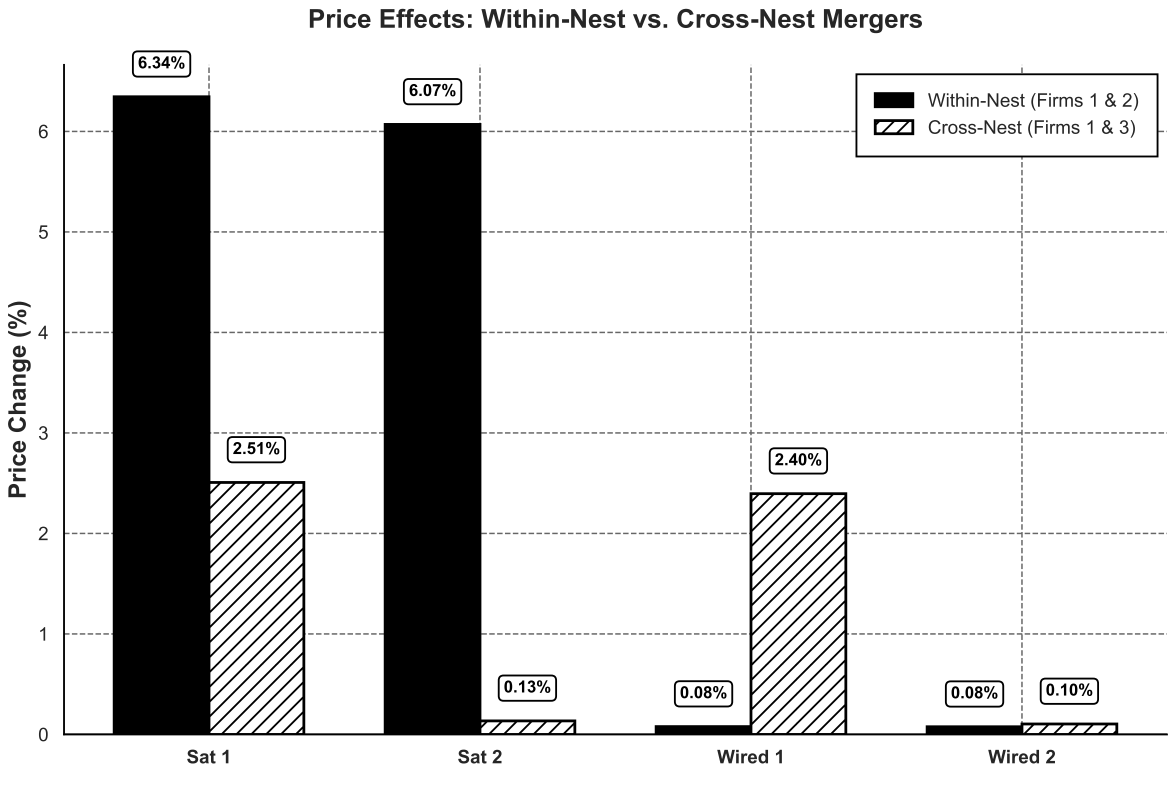

Figure 1: Price Effects: Within- vs. Cross-Nest Mergers. Grouped bars compare percentage price changes across all four products for two merger scenarios: within-nest merger of satellite providers (Firms 1 & 2, black) and cross-nest merger of satellite and wired providers (Firms 1 & 3, white).

The cross-nest merger of satellite provider (firm 1) and wired provider (firm 3) produces substantially smaller price increases than the within-nest merger. Merging firms 1 and 3 raise prices by approximately 2.5% each, compared to 6% increases when satellite competitors merged. This reflects lower diversion ratios between satellite and wired products: when firm 1 raises its satellite price, few consumers switch to firm 3's wired product, limiting the internalized externality. The within-nest merger internalizes higher diversion (satellite consumers readily substitute between satellite products), generating larger recapture effects and price increases. Non-merging firms show asymmetric responses: Satellite 2 barely adjusts (0.13%) when its competitor merges with a wired provider, while Wired 2 increases 2.40% when satellite providers merge. This pattern confirms that competitive pressure operates primarily within product categories. The cross-nest merger reduces average prices relative to the within-nest merger (\$3.360 vs. \$3.422), demonstrating that merger effects depend largely on substitution patterns between merging firms' products.

14. Thus far you have assumed that there are no "efficiencies" (reduction in costs) resulting from the merger. Explain briefly why a merger-specific reduction in marginal cost could mean that a merger is welfare-enhancing.

    Answer: Merger-specific marginal cost reductions create opposing welfare effects. The merger induces price increases through internalization of competitive externalities (unilateral effects), reducing consumer surplus. Simultaneously, lower marginal costs shift the pricing first-order condition, inducing price decreases. From the merged firm's first-order condition, $p_{jt} = mc_{jt} - (\partial s_{jt}/\partial p_{jt})^{-1}\left[s_{jt} + \sum_{k\in\mathcal{J}_{\text{merged}}\setminus\{j\}}(p_{kt} - mc_{kt})\frac{\partial s_{kt}}{\partial p_{jt}}\right]$, a reduction in mc_{jt} directly lowers optimal p_{jt} even holding the markup constant. The net price effect depends on which force dominates.

    Consumer surplus may increase if cost reductions are large enough to offset unilateral effects, causing post-merger prices to fall. Even if prices rise modestly, partial pass-through of cost savings limits consumer harm. Producer surplus typically increases from both recaptured margins on diverted sales and lower production costs on all units sold, with cost savings accruing regardless of price changes.

    For total welfare, marginal cost reductions imply identical output can be produced with fewer inputs. This productive efficiency gain increases total surplus even if prices do not fall, as cost savings accrue to producers. A merger enhances welfare when productive efficiency gains (cost savings times quantity) plus consumer surplus changes (from net price effects) exceed zero. Since consumer surplus changes depend on price changes, which depend on both cost reductions and diversion patterns, merger evaluation requires quantifying both effects. The efficiency must be merger-specific to ensure the defense does not approve anti-competitive mergers claiming hypothetical savings.

15. Consider the merger between firms 1 and 2, and suppose the firms demonstrate that by merging they would reduce marginal cost of each of their products by 15%. Furthermore, suppose that they demonstrate that this cost reduction could not be achieved without merging. Using the `pyBLP` software, re-run the merger simulation with the 15% cost saving. Show the predicted post-merger price changes (again, for each product, averaged across markets). What is the predicted impact of the merger on consumer welfare, assuming that the total measure of consumers M_t is the same in each market t?

```
# Assume M_t = 1000 consumers per market for welfare calculation
M_t = 1000

# Get estimated marginal costs and apply 15% reduction to firms 1 and 2
marginal_costs = optimal_iv_results2.compute_costs()
marginal_costs_efficiency = marginal_costs.copy()
products_1_2 = product_data['firm_ids'].isin([1, 2])
marginal_costs_efficiency[products_1_2] *= 0.85

# Compute post-merger prices with efficiency gains
post_merger_prices_efficiency = optimal_iv_results2.compute_prices(
    firm_ids=merger_firm_ids,  # From Q12
    costs=marginal_costs_efficiency,
    iteration=pyblp.Iteration('simple', {'atol': 1e-12}))

# Calculate price changes
post_merger_prices_eff_avg = post_merger_prices_efficiency.reshape((T, J)).mean(axis=0)
pct_price_changes_eff = ((post_merger_prices_eff_avg - pre_merger_prices_avg)
                          / pre_merger_prices_avg) * 100

# Welfare analysis
cs_pre = optimal_iv_results2.compute_consumer_surpluses()
cs_post_no_eff = optimal_iv_results2.compute_consumer_surpluses(
    prices=post_merger_prices)
cs_post_with_eff = optimal_iv_results2.compute_consumer_surpluses(
    prices=post_merger_prices_efficiency)
# Total CS changes = M_t * sum over all markets
total_delta_cs_no_eff = M_t * (cs_post_no_eff - cs_pre).sum()
total_delta_cs_with_eff = M_t * (cs_post_with_eff - cs_pre).sum()

# Producer surplus
ps_pre = optimal_iv_results2.compute_profits()
ps_post_no_eff = optimal_iv_results2.compute_profits(
    prices=post_merger_prices, shares=shares_post_no_eff,
    costs=marginal_costs)
ps_post_eff = optimal_iv_results2.compute_profits(
    prices=post_merger_prices_efficiency, shares=shares_post_eff,
    costs=marginal_costs_efficiency)
# Total PS changes = M_t * sum over all products/markets
delta_ps_no_eff = M_t * (ps_post_no_eff - ps_pre).sum()
delta_ps_with_eff = M_t * (ps_post_eff - ps_pre).sum()

# Total welfare = DCS + DPS
delta_w_no_eff = total_delta_cs_no_eff + delta_ps_no_eff
delta_w_with_eff = total_delta_cs_with_eff + delta_ps_with_eff
```

Results: *Price Changes with 15% Cost Reduction:*

Product	Firm (Pre)	Firm (Post)	Pre-Merger	Post-Merger	Change ($)	Change (%)
Satellite 1	1	1	3.3381	3.2625	−0.0756	−2.26
Satellite 2	2	1	3.2858	3.2075	−0.0782	−2.38
Wired 1	3	3	3.3210	3.3164	−0.0046	−0.14
Wired 2	4	4	3.3253	3.3212	−0.0041	−0.12

Comparison: No Efficiency vs 15% Cost Reduction:

Product	No Efficiency (%)	With 15% Cost Cut (%)	Difference (pp)
Satellite 1	6.34	−2.26	−8.61
Satellite 2	6.07	−2.38	−8.45
Wired 1	0.08	−0.14	−0.22
Wired 2	0.08	−0.12	−0.20

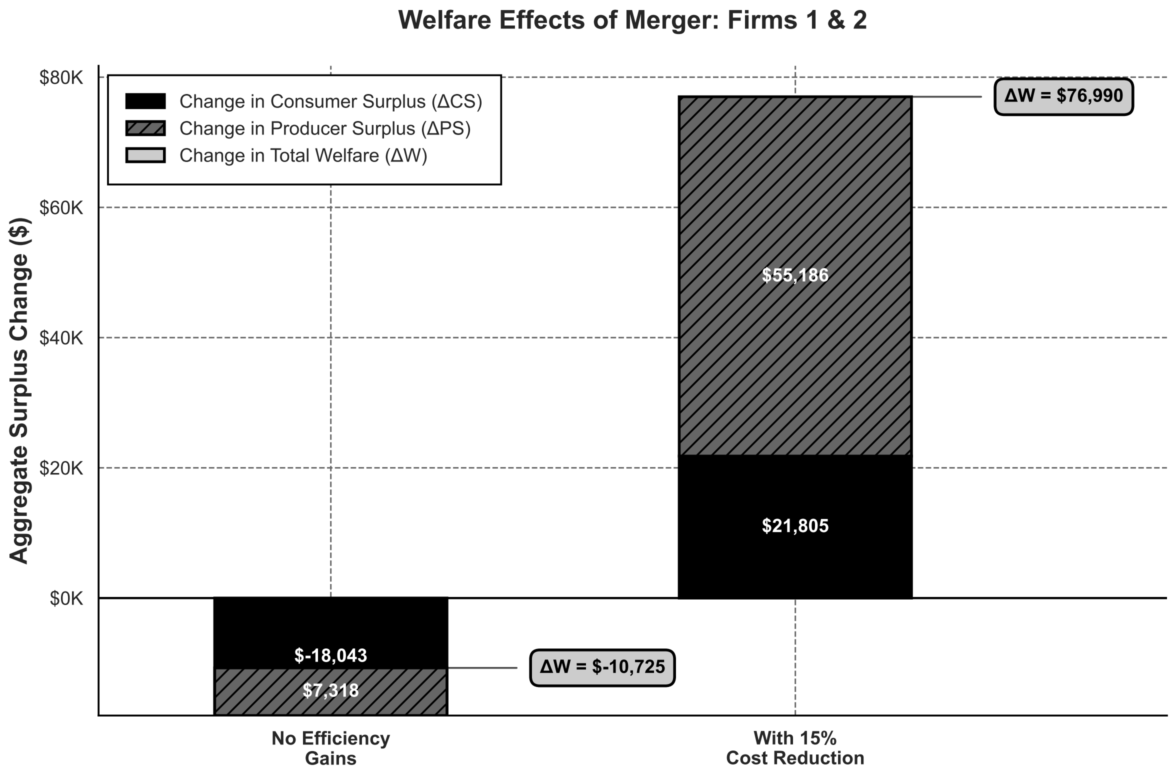

Figure 2: Welfare Effects of Satellite Merger (Firms 1 & 2). Stacked bars show consumer surplus (black) and producer surplus (grey) changes for two scenarios: merger without efficiency gains (left) and merger with 15% cost reduction (right). Total welfare change (ΔW) is shown above each bar. Values represent aggregate changes across all 600 markets with 1,000 consumers per market.

Welfare Analysis (assuming $M_t = 1{,}000$ consumers/market, $T = 600$ markets):

Scenario	ΔCS total ($)	ΔPS ($)	ΔW ($)
Without efficiency	−18,043	7,318	−10,725
With 15% cost cut	21,805	55,186	76,990

The 15% cost reduction reverses merger price effects: merging satellite firms reduce prices by 2.3% rather than increasing by 6%. Non-merging wired firms respond with slight price decreases (0.13%) as best responses. Without efficiency, the merger reduces total welfare by \$10,725 (consumer harm of \$18,043 exceeds producer gain of \$7,318).

With efficiency, total welfare increases by \$76,990, decomposing into a consumer surplus gain of \$21,805 plus a producer surplus gain of \$55,186. Cost pass-through dominates unilateral effects: lower marginal costs shift pricing first-order conditions sufficiently to overcome recapture incentives, yielding price decreases. Producer surplus increases despite lower prices because cost savings (\$0.15 reduction × quantity sold) exceed markup erosion. The productive efficiency gain (real resource savings from lower costs) transforms a welfare-reducing merger into a welfare-enhancing one.

15. cont. Explain why assuming that the total measure of consumers M_t is the same in each market t (or data on the correct values of M_t) is needed here, whereas up to this point it was without loss to assume $M_t = 1$.

Answer: All prior analysis focused on per-capita quantities: market shares s_{jt}, prices p_{jt}, elasticities ε_{jk}, and diversion ratios D_{jk}. These are invariant to M_t because they involve ratios or derivatives of shares. Market share $s_{jt} = Q_{jt}/M_t$ is a ratio, so normalizing $M_t = 1$ is without loss. Elasticity $\varepsilon_{jk} = (\partial s_j/\partial p_k) \cdot (p_k/s_j)$ involves derivatives of shares. Equilibrium prices satisfy $p_{jt} = mc_{jt} - (\partial s_{jt}/\partial p_{jt})^{-1} s_{jt}$, depending only on shares and their derivatives.

Welfare analysis requires aggregation across markets. Consumer surplus change per market is:

$$\Delta CS_t = M_t \cdot \mathbb{E}_{\beta_i,\epsilon_i}\left[\frac{1}{\alpha_i}\left(\ln \sum_j e^{\delta_j^{\text{pre}}+\nu_{ij}} - \ln \sum_j e^{\delta_j^{\text{post}}+\nu_{ij}}\right)\right].$$

Total consumer surplus change is:

$$\Delta CS = \sum_{t=1}^{T} M_t \cdot \Delta cs_t,$$

where Δcs_t is per-capita change. Without M_t, we cannot compute total dollar values or weight markets by size.

Consider two markets: $M_A = 10{,}000$ with $\Delta cs_A = -\$5$ per capita; $M_B = 1{,}000{,}000$ with $\Delta cs_B = +\$0.10$ per capita. Total change is $10{,}000 \times (-\$5) + 1{,}000{,}000 \times (+\$0.10) = +\$50{,}000$. Averaging per-capita changes gives $(-\$5 + \$0.10)/2 = -\$2.45$, reversing the sign. Proper aggregation requires weighting by market size.

Producer surplus also scales with M_t:

$$\Delta PS = \sum_{t=1}^{T} M_t \sum_j \left[(p_{jt}^{\text{post}} - mc_{jt}^{\text{post}}) s_{jt}^{\text{post}} - (p_{jt}^{\text{pre}} - mc_{jt}^{\text{pre}}) s_{jt}^{\text{pre}}\right].$$

Total welfare is:

$$\Delta W = \Delta CS + \Delta PS = \sum_{t=1}^{T} M_t \left[\Delta cs_t + \sum_j \left((p_{jt}^{\text{post}} - mc_{jt}^{\text{post}}) s_{jt}^{\text{post}} - (p_{jt}^{\text{pre}} - mc_{jt}^{\text{pre}}) s_{jt}^{\text{pre}} \right) \right].$$

This decomposes into productive efficiency gains (cost savings on units sold) minus allocative inefficiency (welfare loss from reduced consumption):

$$\Delta W = \sum_{t=1}^{T} M_t \left[\sum_j \left(mc_{jt}^{\text{pre}} - mc_{jt}^{\text{post}} \right) Q_{jt}^{\text{post}} - \sum_j \int_{Q_{jt}^{\text{post}}}^{Q_{jt}^{\text{pre}}} \left(P_j(q) - mc_{jt}^{\text{post}} \right) dq \right].$$

where the integral represents deadweight loss from foregone trades when prices rise. Both components scale with M_t. Without actual market sizes, we cannot value efficiency gains or compare them to consumer harm in dollar terms. Aggregate welfare statements require proper weighting across heterogeneous markets.

Merger Simulations Summary

In sum, the within-nest merger without efficiencies (firms 1&2) generates the largest price increase (\$0.104), while the cross-nest merger (firms 1&3) produces smaller effects (\$0.043). The 15% cost reduction reverses price effects (−\$0.041). Markups increase in all scenarios through internalization of competitive externalities, with the largest increase (\$0.050) under efficiency gains, where cost reductions permit higher percentage markups despite lower prices.

Average profit per product increases by \$0.003 (within-nest), \$0.002 (cross-nest), and \$0.023 (with efficiency), reflecting recaptured margins and, in the efficiency case, substantial cost savings. Average consumer surplus falls under both anti-competitive mergers (−\$0.030 within-nest, −\$0.017 cross-nest) but rises (\$0.036) when cost pass-through dominates unilateral effects. Three patterns emerge: (1) merger effects scale with diversion ratios between merging products; (2) cross-category mergers generate smaller competitive harm than within-category mergers; (3) merger-specific efficiencies can reverse welfare effects when cost reductions are large relative to market power increases.

Metric	Pre-Merger	Merger 1&2 (No Eff)	Merger 1&3 (Cross)	Merger 1&2 (15% Cut)	Δ (1&2 − Pre)	Δ (1&3 − Pre)	Δ (15% Cut − Pre)
Avg Price ($)	3.318	3.422	3.360	3.277	0.104	0.043	−0.041
Avg Markup ($)	0.204	0.227	0.214	0.254	0.023	0.010	0.050
Avg Profit ($)	0.112	0.115	0.113	0.135	0.003	0.002	0.023
Avg CS ($)	0.462	0.432	0.445	0.498	−0.030	−0.017	0.036

Note: Averages computed across 600 markets and 4 products. Avg Price = mean price per product; Avg Markup = mean markup ($p_{jt} - mc_{jt}$) per product; Avg Profit = mean producer surplus per product; Avg CS = mean consumer surplus per capita. Assumed market size: 1,000 *consumers per market (600,000 total consumers).*

References

Berry, S., J. Levinsohn, and A. Pakes (1995): "Automobile Prices in Market Equilibrium," *Econometrica*, 63, 841–890.

Berry, S. T. (1994): "Estimating Discrete-Choice Models of Product Differentiation," *The RAND Journal of Economics*, 25, 242–262.

Berry, S. T. and P. A. Haile (2021): "Foundations of Demand Estimation," in *Handbook of Industrial Organization*, vol. 4, 1–62.

Conlon, C. and J. Gortmaker (2020): "Best practices for differentiated products demand estimation with PyBLP," *The RAND Journal of Economics*, 51, 1108–1161.

Conlon, C. and J. H. Mortimer (2021): "Empirical Properties of Diversion Ratios," *The RAND Journal of Economics*, 52, 693–726.