In [1]:
```
import numpy as np
import pandas as pd
import pyblp as blp
import torch
from torch.autograd import Variable
import torch.optim as optim
from linearmodels.iv import IV2SLS
from HomogenousDemandEstimation import HomDemEst

blp.options.digits = 2
blp.options.verbose = False
nax = np.newaxis
```

Exercise 3

The file `ps1_ex3.csv` contains aggregate data on a large number $T = 1000$ of markets in which $J = 6$ products compete between each other together with an outside good $j = 0$. The utility of consumer i is given by:

$$\begin{aligned} u_{ijt} &= -\alpha p_{jt} + \mathbf{x}_{jt}\boldsymbol{\beta} + \xi_{jt} + \epsilon_{ijt} \quad j = 1, \ldots, 6 \\ u_{i0t} &= \epsilon_{i0t} \end{aligned}$$

where p_{jt} is the price of product j in market t, $\mathbf{x}_{jt}$ is an observed product characteristic (including a constant), ξ_{jt} is an unobserved product characteristic and ϵ_{ijt} is i.i.d T1EV $(0, 1)$. Our goal is to to estimate demand parameters $(\alpha, \boldsymbol{\beta})$ and perform some counterfactual exercises.

In [2]:
```
# Load the dataset.
data_ex3 = pd.read_csv('ps1_ex3.csv')
num_prod = data_ex3.Product.max()
num_T = data_ex3.market.max()
```

Part 1

Assuming that the variable z in the dataset is a valid instrument for prices, write down the moment condition that allows you to consistently estimate (α, β) and obtain an estimate for both parameters.

Under the T1EV assumption, we can derive the CCPs which corresponds to the predicted market share for product j at time t. This can be approximated from the data using the observed market share s_{jt}.

$$\Pr(i \text{ chooses } j \text{ at time } t) = \frac{\exp\left(-\alpha p_{jt} + \mathbf{x}_{jt}'\boldsymbol{\beta} + \xi_{jt}\right)}{\sum_{k \in \mathcal{J}_t} \exp\left(-\alpha p_{kt} + \mathbf{x}_{kt}'\boldsymbol{\beta} + \xi_{kt}\right)} \approx s_{jt}$$

We can invoke the normalization assumption on u_{i0} and take the logarithm of the share ratio s_{jt}/s_{0t} to obtain

$$\ln\left(\frac{s_{jt}}{s_{0t}}\right) = -\alpha p_{jt} + \mathbf{x}_{jt}'\boldsymbol{\beta} + \xi_{jt}$$

In [3]:

```
# Create outside option shares and merge into dataset.
share_total = data_ex3.groupby(['market'])['Shares'].sum().reset_index()
share_total.rename(columns={'Shares': 's0'}, inplace=True)
share_total['s0'] = 1 - share_total['s0']
data_ex3 = pd.merge(data_ex3, share_total, on='market')

# Create natural log of share ratios
data_ex3['s_ratio'] = np.log(data_ex3['Shares']/data_ex3['s0'])
```

Given that z_{jt} is a relevant instrument for p_{jt} and that $\mathbf{x}_{jt}$ is exogenous, we can impose the conditional exogeneity restriction

$$\mathbb{E}\left[\xi_{jt} \mid \mathbf{x}_{jt}, z_{jt}\right] = 0$$

in order to estimate α and $\boldsymbol{\beta}$. Using the Law of Iterated Expectations, we can conclude that

$$\mathbb{E}\left[\begin{pmatrix}\mathbf{x}_{jt}\\ z_{jt}\end{pmatrix}\xi_{jt}\right] = \mathbb{E}\left[\begin{pmatrix}\mathbf{x}_{jt}\\ z_{jt}\end{pmatrix}\left\{\ln\left(\frac{s_{jt}}{s_{0t}}\right) + \alpha p_{jt} - \mathbf{x}_{jt}'\boldsymbol{\beta}\right\}\right] = \begin{pmatrix}0\\0\end{pmatrix}$$

Given that 3 moment conditions across all products and markets, we are exactly identifying α and β.

GMM provides the minimizer corresponding to a quadratic loss function with 3 moments.

$$\begin{pmatrix}\widehat{\alpha}\\ \widehat{\boldsymbol{\beta}}\end{pmatrix} \in \underset{\begin{pmatrix}\alpha\\ \beta\end{pmatrix}}{\arg\min}\left[\frac{1}{T\times J}\sum_t\sum_j x_{jt}\left\{\ln\left(\frac{s_{jt}}{s_{0t}}\right) + \alpha p_{jt} - \mathbf{x}_{jt}'\boldsymbol{\beta}\right\}\right]$$

I perform the two-step procedure to obtain the efficient GMM estimator of the model parameters.

In [4]:

```
est = HomDemEst(data_dict={
    'Data': data_ex3,
    'Choice Column': 'Product',
    'Market Column': 'market',
    'Log Share Ratio Column': 's_ratio',
    'Endogenous Columns': ['Prices'],
    'Exogenous Columns': ['x'],
    'Instrument Columns': ['z'],
    'Add Constant': True
})

results = est.run_gmm()
```

In [5]:

```
results['Coefficients']
```

Out[5]:
```
tensor([ 0.7289,  0.3047, -0.4675], dtype=torch.float64)
```

In [6]:
```
torch.sqrt(torch.diag(results['Covariance Matrix']))
```

Out[6]:
```
tensor([0.1989, 0.0084, 0.0633], dtype=torch.float64)
```

We find the following estimates for α and β.

Coefficient	Estimate	Std. Error
Constant	0.7289	0.1989
Prices	0.4675	0.0633
Product Characteristic	0.3047	0.0084

In [7]:
```
data_ex3['const'] = 1

iv = IV2SLS(dependent=data_ex3['s_ratio'],
       exog=data_ex3[['const', 'x']],
       endog=data_ex3['Prices'],
       instruments=data_ex3['z']).fit(cov_type='unadjusted')

print(iv.summary)
```

```
                          IV-2SLS Estimation Summary
==============================================================================
Dep. Variable:                s_ratio   R-squared:                      0.0018
Estimator:                    IV-2SLS   Adj. R-squared:                 0.0015
No. Observations:                6000   F-statistic:                    1575.0
Date:                Sun, Feb 06 2022   P-value (F-stat)                0.0000
Time:                        11:46:39   Distribution:                  chi2(2)
Cov. Estimator:            unadjusted

                             Parameter Estimates
==============================================================================
            Parameter  Std. Err.     T-stat    P-value    Lower CI    Upper CI
------------------------------------------------------------------------------
const          0.7289     0.1944     3.7487     0.0002      0.3478      1.1100
x              0.3047     0.0083     36.553     0.0000      0.2883      0.3210
Prices        -0.4675     0.0618    -7.5651     0.0000     -0.5886     -0.3464
==============================================================================

Endogenous: Prices
Instruments: z
Unadjusted Covariance (Homoskedastic)
Debiased: False
```

Part 2

We know that the elasticities for homogenous demand are given by

$$\varepsilon_{jk,t} = \begin{cases} -\alpha p_{j,t}\left(1 - \pi_{j,t}\right) & \text{if } j = k \\ \alpha p_{k,t}\pi_{k,t} & \text{otherwise} \end{cases}$$

In [8]:

```
α = -np.array(results['Coefficients'])[-1]
```

In [9]:

```
# Compute the own and cross-price elasticity for each market and pair of product
data_ex3['own'] = -α * data_ex3['Prices'] * (1 - data_ex3['Shares'])
data_ex3['cross'] = α * data_ex3['Prices'] * data_ex3['Shares']
e_mean = data_ex3.groupby(['Product'])[['own', 'cross']].mean()

# Generate matrix of average elasticities.
e_mat = np.tile(e_mean['cross'], (num_prod, 1))
np.fill_diagonal(e_mat, e_mean['own'])

# Convert it to a dataframe.
prod_list = list(map(str, range(1, num_prod+ 1)))
e_mat = pd.DataFrame(e_mat, index=prod_list, columns=prod_list)
```

j/k	Product 1	Product 2	Product 3	Product 4	Product 5	Product 6
Product 1	-1.249624	0.323002	0.128738	0.127154	0.125510	0.129465
Product 2	0.321076	-1.251242	0.128738	0.127154	0.125510	0.129465
Product 3	0.321076	0.323002	-1.289146	0.127154	0.125510	0.129465
Product 4	0.321076	0.323002	0.128738	-1.293869	0.125510	0.129465
Product 5	0.321076	0.323002	0.128738	0.127154	-1.291425	0.129465
Product 6	0.321076	0.323002	0.128738	0.127154	0.125510	-1.290799

The results above show that the own-price elasticities are pretty consistent across the various products, and that the magnitude of the cross-price elasticities is lower than the corresponding own-price elasticities.

Part 3

To back out the marginal costs for producing each product in market t, we must first construct the conduct matrix $\mathbf{H}_t$ corresponding to the entire choice set. Since we assume that firms are single-product producers in all markets, the conduct matrix will be an identity matrix of dimension 6. Furthermore, we will need the matrix $\mathbf{\Omega}_t$ containing partial derivatives $-\partial q_{kt}/\partial p_{jt}$ multiplied by the corresponding entries $\mathbf{H}_t$. However, since $\mathbf{H}_t$ is an identity matrix, we only need to compute the diagonal entries of $\mathbf{\Omega}_t$ as all the off-diagonals entries will equal 0. Therefore, we can back out the following expression for diagonal element j of $\mathbf{\Omega}_t$ assuming that N_t represents the total size of the market.

$$\Omega_{jj,t} = -\frac{\partial q_{jt}}{\partial p_{jt}} = -\varepsilon_{jj,t}\frac{q_{jt}}{p_{jt}} = -\varepsilon_{jj,t}\frac{s_{jt}}{p_{jt}}N_t$$

The firm's profit maximization problem yields the following FOC:

$$\mathbf{p}_t - \mathbf{mc}_t \; = \; \mathbf{\Omega}_t^{-1}\mathbf{q}(\mathbf{p}_t) \; = \; \mathbf{\Omega}_t^{-1}\mathbf{s}(\mathbf{p}_t)N_t$$

Since $\mathbf{\Omega}$ is a diagonal matrix, we can back out the marginal cost of each product j in market t independently of the other products.

$$p_{jt} - \mathrm{mc}_{jt} \; = \; \Omega_{jj,t}^{-1} s_{jt} N_t$$

$$\mathrm{mc}_{jt} \; = \; p_{jt} + \frac{1}{\varepsilon_{jj,t} N_t} \frac{p_{jt}}{s_{jt}} s_{jt} N_t \; = \; p_{jt}\left(1 + \frac{1}{\varepsilon_{jj,t}}\right)$$

In [11]:

```
data_ex3['mc'] = data_ex3['Prices'] * (1 + 1/data_ex3['own'])
mc_avg = data_ex3.groupby(['Product'])['mc'].mean()
```

In [12]:

```
all_avg = data_ex3.groupby(['Product'])[['Prices', 'Shares', 'mc']].mean()
```

We obtain the following average (across markets) marginal cost for each product. They are highly positively correlated with the average (across markets) prices and shares.

Product	Average Price	Average Share	Average MC
1	3.35995	0.202451	0.667126
2	3.36753	0.203076	0.671897
3	3.03306	0.0903493	0.678684
4	3.03977	0.0889407	0.688906
5	3.03103	0.0881716	0.682632
6	3.03815	0.0906875	0.682689

Part 4

Suppose that product $j = 1$ exits the market, and marginal costs and product characteristics for the other products remain unchanged. Since the elasticities of demand for each product remain unchanged and given that all firms are single-product producers, the matrix Ω does not change for the remaining products, which implies that prices do not change for the products and average prices stay the same. To compute the new shares, we will recompute s_0 from the log share ratios for products $j = 2, \cdots, 6$. Under the new regime, we have that $\sum_{j=0,2}^{6} s_j = 1$. Therefore, we can write that

$$\sum_{j=2}^{6} \exp \log \frac{s_j}{s_0} = \frac{1 - s_0}{s_0} \quad \Rightarrow \quad s_0 \; = \; \left(1 + \sum_{j=2}^{6} \frac{s_j}{s_0}\right)^{-1}$$

In [ ]:
```
data_new = data_ex3.query('Product != 1')
data_new.drop(columns=['s0'], inplace=True)
data_new['exp_s_ratio'] = np.exp(data_new['s_ratio'])

# Create sum of share ratios
new_s0 = data_new.groupby(['market'])['exp_s_ratio'].sum().reset_index()
new_s0.rename(columns={'exp_s_ratio': 's0'}, inplace=True)
new_s0.loc[:, 's0'] = 1 / (1 + new_s0.loc[:, 's0'])
data_new = pd.merge(data_new, new_s0, on='market')
data_new['Shares_new'] = data_new['exp_s_ratio'] * data_new['s0']
# data_new.drop(columns=['exp_s_ratio'])

data_new['Profit_old'] = (data_new['Prices'] - data_new['mc']) * data_new['Share
data_new['Profit_new'] = (data_new['Prices'] - data_new['mc']) * data_new['Share
data_new['Profit_change'] = data_new['Profit_new'] - data_new['Profit_old']
```

In [14]:
```
new_avg = data_new.groupby(['Product'])[['Prices', 'Shares_new', 'mc', 'Profit_o
```

Product	Average Price	Average Share	Average Change in Profits
2	3.36753	0.254303	0.139413
3	3.03306	0.113245	0.054411
4	3.03977	0.111318	0.053106
5	3.03103	0.110616	0.053617
6	3.03815	0.113617	0.054554

We see that all products (and correspondingly the firms that produce them) face an increase in profits, especially product 2. Consumers will face a reduction in welfare due to the increase in concentration of market power among the remaining firms.

In [14]: