IO Problem Set 1 (BLP)

Chris Ackerman*

November 4, 2021

Problem 1

Estimate the Model using OLS, with price and promotion as characteristics

```
  #1.1: Estimate using OLS with price and promotion as product characteristics.
res_log1 = smf.ols('Y ~ prices + prom_', data=otc_dataDf).fit()
```

Dep. Variable:	Y	**R-squared:**	0.158
Model:	OLS	**Adj. R-squared:**	0.158
Method:	Least Squares	**F-statistic:**	3610.
Date:	Thu, 04 Nov 2021	**Prob (F-statistic):**	0.00
Time:	05:19:39	**Log-Likelihood:**	-56307.
No. Observations:	38544	**AIC:**	1.126e+05
Df Residuals:	38541	**BIC:**	1.126e+05
Df Model:	2		
Covariance Type:	nonrobust		

	coef	**std err**	**t**	**P> |t|**	**[0.025**	**0.975]**
Intercept	-7.6267	0.014	-532.839	0.000	-7.655	-7.599
prices	-0.2496	0.003	-84.768	0.000	-0.255	-0.244
prom_	-0.0311	0.019	-1.653	0.098	-0.068	0.006

Omnibus:	1648.591	**Durbin-Watson:**	0.434
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	1461.418
Skew:	-0.415	**Prob(JB):**	0.00
Kurtosis:	2.529	**Cond. No.**	17.7

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.

*I worked on this problem set with Luna Shen, David Kerns, and Benedikt Graf

Estimate the Model using OLS, with price and promotion as characteristics, and brand dummies

```python
#1.2: Estimate using OLS with price and promotion as product characteristics and brand dummies.
res_log2 = smf.ols('Y ~ prices + prom_ + C(brand)', data=otc_dataDf).fit()
```

Dep. Variable:	Y	**R-squared:**	0.654
Model:	OLS	**Adj. R-squared:**	0.654
Method:	Least Squares	**F-statistic:**	6081.
Date:	Thu, 04 Nov 2021	**Prob (F-statistic):**	0.00
Time:	05:20:51	**Log-Likelihood:**	-39138.
No. Observations:	38544	**AIC:**	7.830e+04
Df Residuals:	38531	**BIC:**	7.841e+04
Df Model:	12		
Covariance Type:	nonrobust		

	coef	**std err**	**t**	**P>|t|**	**[0.025**	**0.975]**
Intercept	-6.0745	0.036	-167.065	0.000	-6.146	-6.003
C(brand)[T.2]	-0.0048	0.022	-0.218	0.828	-0.048	0.039
C(brand)[T.3]	-0.4578	0.040	-11.502	0.000	-0.536	-0.380
C(brand)[T.4]	-0.4303	0.017	-25.951	0.000	-0.463	-0.398
C(brand)[T.5]	-0.8868	0.024	-37.403	0.000	-0.933	-0.840
C(brand)[T.6]	-1.3850	0.051	-27.408	0.000	-1.484	-1.286
C(brand)[T.7]	-1.6527	0.018	-94.139	0.000	-1.687	-1.618
C(brand)[T.8]	-2.2856	0.016	-141.034	0.000	-2.317	-2.254
C(brand)[T.9]	-1.9340	0.017	-111.950	0.000	-1.968	-1.900
C(brand)[T.10]	-1.8983	0.022	-87.306	0.000	-1.941	-1.856
C(brand)[T.11]	-2.1754	0.019	-113.355	0.000	-2.213	-2.138
prices	-0.3412	0.010	-33.864	0.000	-0.361	-0.321
prom_	0.3294	0.013	26.122	0.000	0.305	0.354

Omnibus:	2773.498	**Durbin-Watson:**	1.024
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	3867.305
Skew:	-0.618	**Prob(JB):**	0.00
Kurtosis:	3.938	**Cond. No.**	112.

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.

Estimate the Model using OLS, with price and promotion as characteristics, and store-brand dummies

```
#1.3. Using OLS with price and promotion as product characteristics and store-brand
#(the interaction of brand and store) dummies.

res_log3 = smf.ols('Y ~ prices + prom_ + C(brand)*C(store)', data=otc_dataDf).fit()
```

Dep. Variable:	Y	**R-squared:**	0.722
Model:	OLS	**Adj. R-squared:**	0.716
Method:	Least Squares	**F-statistic:**	121.9
Date:	Thu, 04 Nov 2021	**Prob (F-statistic):**	0.00
Time:	05:21:32	**Log-Likelihood:**	-34946.
No. Observations:	38544	**AIC:**	7.150e+04
Df Residuals:	37739	**BIC:**	7.839e+04
Df Model:	804		
Covariance Type:	nonrobust		

	coef	**std err**	**t**	**P> |t|**	**[0.025**	**0.975]**
Intercept	-6.1994	0.093	-66.365	0.000	-6.382	-6.016
prices	-0.3302	0.010	-34.349	0.000	-0.349	-0.311
prom_	0.3288	0.011	28.619	0.000	0.306	0.351

Omnibus:	4044.934	**Durbin-Watson:**	1.268
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	7222.052
Skew:	-0.721	**Prob(JB):**	0.00
Kurtosis:	4.555	**Cond. No.**	4.08e+03

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 4.08e+03. This might indicate that there are strong multicollinearity or other numerical problems. Dummies omitted.

Estimate the models from parts 1–3 using wholesale cost as an instrument

```
# OLS with price and promotion as product characteristics, using wholesale cost as instrument
wholeSale_IV1 = iv.IV2SLS.from_formula('Y ~ 1 + [prices ~ cost_] + prom_ ', data=otc_dataDf).fit()
```

Dep. Variable:	Y	**R-squared:**	0.1531
Estimator:	IV-2SLS	**Adj. R-squared:**	0.1531
No. Observations:	38544	**F-statistic:**	5255.4
Date:	Thu, Nov 04 2021	**P-value (F-stat)**	0.0000
Time:	05:25:20	**Distribution:**	chi2(2)
Cov. Estimator:	robust		

	Parameter	Std. Err.	T-stat	P-value	Lower CI	Upper CI
Intercept	-7.8181	0.0150	-520.30	0.0000	-7.8476	-7.7887
prom_	-0.0068	0.0170	-0.4034	0.6866	-0.0401	0.0264
prices	-0.2066	0.0029	-71.884	0.0000	-0.2122	-0.2009

Endogenous: prices
Instruments: cost_
Robust Covariance (Heteroskedastic)
Debiased: False

```python
# OLS with price and promotion as product characteristics and brand dummies
# using wholesale cost as instrument

wholeSale_IV2 = iv.IV2SLS.from_formula('Y ~ 1 + [prices ~ cost_] + prom_ + C(brand)', data=otc_dataDf).
```

Dep. Variable:	Y	**R-squared:**	0.6446
Estimator:	IV-2SLS	**Adj. R-squared:**	0.6445
No. Observations:	38544	**F-statistic:**	9.69e+04
Date:	Thu, Nov 04 2021	**P-value (F-stat)**	0.0000
Time:	05:26:08	**Distribution:**	chi2(12)
Cov. Estimator:	robust		

	Parameter	**Std. Err.**	**T-stat**	**P-value**	**Lower CI**	**Upper CI**
Intercept	-7.2171	0.0652	-110.69	0.0000	-7.3449	-7.0893
C(brand)[T.2]	-0.5161	0.0309	-16.703	0.0000	-0.5766	-0.4555
C(brand)[T.3]	-1.6627	0.0698	-23.833	0.0000	-1.7994	-1.5260
C(brand)[T.4]	-0.2833	0.0147	-19.314	0.0000	-0.3121	-0.2546
C(brand)[T.5]	-1.4660	0.0358	-40.904	0.0000	-1.5363	-1.3958
C(brand)[T.6]	-2.9699	0.0914	-32.484	0.0000	-3.1491	-2.7907
C(brand)[T.7]	-1.4158	0.0181	-78.185	0.0000	-1.4513	-1.3803
C(brand)[T.8]	-2.3613	0.0137	-172.35	0.0000	-2.3882	-2.3344
C(brand)[T.9]	-2.1365	0.0169	-126.55	0.0000	-2.1696	-2.1034
C(brand)[T.10]	-1.4120	0.0321	-44.036	0.0000	-1.4749	-1.3492
C(brand)[T.11]	-2.5260	0.0283	-89.396	0.0000	-2.5814	-2.4707
prom_	0.4307	0.0145	29.801	0.0000	0.4024	0.4590
prices	-0.0081	0.0189	-0.4287	0.6682	-0.0452	0.0290

Endogenous: prices
Instruments: cost_
Robust Covariance (Heteroskedastic)
Debiased: False

```python
# OLS with price and promotion as product characteristics and brand dummies
# using wholesale cost as instrument

wholeSale_IV3 = iv.IV2SLS.from_formula('Y ~ 1 + [prices ~ cost_] + prom_ + C(brand)*C(store)', data=otc_
```

Dep. Variable:	Y	**R-squared:**	0.7150
Estimator:	IV-2SLS	**Adj. R-squared:**	0.7090
No. Observations:	38544	**F-statistic:**	1.766e+05
Date:	Thu, Nov 04 2021	**P-value (F-stat)**	0.0000
Time:	05:27:37	**Distribution:**	chi2(804)
Cov. Estimator:	robust		

	Parameter	**Std. Err.**	**T-stat**	**P-value**	**Lower CI**	**Upper CI**
Intercept	-7.2138	0.0775	-93.106	0.0000	-7.3657	-7.0620
prom_	0.4193	0.0132	31.774	0.0000	0.3934	0.4451
prices	-0.0346	0.0178	-1.9442	0.0519	-0.0695	0.0003

Table 1: IV-2SLS Estimation Summary, dummies suppressed

Endogenous: prices
Instruments: cost_
Robust Covariance (Heteroskedastic)
Debiased: False

Estimate the models from parts 1–3 using the Hausman instrument

Dep. Variable:	Y	**R-squared:**	0.1578
Estimator:	IV-2SLS	**Adj. R-squared:**	0.1577
No. Observations:	38544	**F-statistic:**	9465.0
Date:	Thu, Nov 04 2021	**P-value (F-stat)**	0.0000
Time:	05:40:33	**Distribution:**	chi2(2)
Cov. Estimator:	robust		

	Parameter	Std. Err.	T-stat	P-value	Lower CI	Upper CI
Intercept	-7.6143	0.0135	-565.64	0.0000	-7.6407	-7.5879
prom_	-0.0327	0.0170	-1.9257	0.0541	-0.0660	0.0006
prices	-0.2524	0.0026	-97.062	0.0000	-0.2574	-0.2473

Endogenous: prices
Instruments: pricestore1, pricestore2, pricestore3, pricestore4, pricestore5, pricestore6, pricestore7, pricestore8, pricestore9, pricestore10, pricestore11, pricestore12, pricestore13, pricestore14, pricestore15, pricestore16, pricestore17, pricestore18, pricestore19, pricestore20, pricestore21, pricestore22, pricestore23, pricestore24, pricestore25, pricestore26, pricestore27, pricestore28, pricestore29, pricestore30
Robust Covariance (Heteroskedastic)
Debiased: False

Mean own-price elasticities from the estimates in models 1–3

These results make sense. As a rule of thumb, the own-price elasticities should be $\in (-2, -5)$. The IV estimates using the Hausman instrument are approximately in this range. The OLS estimates are not, indicating that endogeneity is a practical concern in this setting. The estimates with wholesale cost as an instrument are also “too small”, indicating that wholesale cost may not be a viable instrument in this context.

Dep. Variable:	Y	**R-squared:**	0.6511
Estimator:	IV-2SLS	**Adj. R-squared:**	0.6510
No. Observations:	38544	**F-statistic:**	9.529e+04
Date:	Thu, Nov 04 2021	**P-value (F-stat)**	0.0000
Time:	05:41:01	**Distribution:**	chi2(12)
Cov. Estimator:	robust		

	Parameter	Std. Err.	T-stat	P-value	Lower CI	Upper CI
Intercept	-5.4061	0.0539	-100.33	0.0000	-5.5117	-5.3005
C(product_ids)[T.2]	0.2942	0.0259	11.370	0.0000	0.2435	0.3449
C(product_ids)[T.3]	0.2471	0.0574	4.3071	0.0000	0.1347	0.3595
C(product_ids)[T.4]	-0.5162	0.0140	-36.812	0.0000	-0.5437	-0.4888
C(product_ids)[T.5]	-0.5479	0.0298	-18.367	0.0000	-0.6064	-0.4894
C(product_ids)[T.6]	-0.4578	0.0751	-6.0967	0.0000	-0.6050	-0.3107
C(product_ids)[T.7]	-1.7913	0.0169	-105.81	0.0000	-1.8245	-1.7581
C(product_ids)[T.8]	-2.2413	0.0143	-156.43	0.0000	-2.2694	-2.2132
C(product_ids)[T.9]	-1.8155	0.0156	-116.73	0.0000	-1.8460	-1.7850
C(product_ids)[T.10]	-2.1827	0.0280	-77.963	0.0000	-2.2376	-2.1279
C(product_ids)[T.11]	-1.9703	0.0231	-85.436	0.0000	-2.0155	-1.9251
prom_	0.2701	0.0143	18.947	0.0000	0.2421	0.2980
prices	-0.5361	0.0155	-34.507	0.0000	-0.5665	-0.5056

Endogenous: prices
Instruments: pricestore1, pricestore2, pricestore3, pricestore4, pricestore5, pricestore6, pricestore7, pricestore8, pricestore9, pricestore10, pricestore11, pricestore12, pricestore13, pricestore14, pricestore15, pricestore16, pricestore17, pricestore18, pricestore19, pricestore20, pricestore21, pricestore22, pricestore23, pricestore24, pricestore25, pricestore26, pricestore27, pricestore28, pricestore29, pricestore30
Robust Covariance (Heteroskedastic)
Debiased: False

Dep. Variable:	Y	**R-squared:**	0.7183
Estimator:	IV-2SLS	**Adj. R-squared:**	0.7123
No. Observations:	38544	**F-statistic:**	1.711e+05
Date:	Thu, Nov 04 2021	**P-value (F-stat)**	0.0000
Time:	05:42:30	**Distribution:**	chi2(804)
Cov. Estimator:	robust		

	Parameter	**Std. Err.**	**T-stat**	**P-value**	**Lower CI**	**Upper CI**
Intercept	-5.4606	0.0662	-82.482	0.0000	-5.5903	-5.3308
prom_	0.2630	0.0130	20.297	0.0000	0.2376	0.2884
prices	-0.5454	0.0138	-39.560	0.0000	-0.5725	-0.5184

Endogenous: prices
Instruments: pricestore1, pricestore2, pricestore3, pricestore4, pricestore5, pricestore6, pricestore7, pricestore8, pricestore9, pricestore10, pricestore11, pricestore12, pricestore13, pricestore14, pricestore15, pricestore16, pricestore17, pricestore18, pricestore19, pricestore20, pricestore21, pricestore22, pricestore23, pricestore24, pricestore25, pricestore26, pricestore27, pricestore28, pricestore29, pricestore30
Robust Covariance (Heteroskedastic) Dummies omitted.
Debiased: False

brand	OLS1	OLS2	OLS3	IV4.1	IV4.2	IV4.3	IV5.1	IV5.2	IV5.3
1	-0.852929	-1.166183	-1.128481	-0.706043	-0.027733	-0.118239	-0.862496	-1.832176	-1.864240
2	-1.232714	-1.685451	-1.630960	-1.020423	-0.040082	-0.170887	-1.246541	-2.647991	-2.694332
3	-1.750607	-2.393550	-2.316167	-1.449128	-0.056921	-0.242681	-1.770243	-3.760477	-3.826287
4	-0.739114	-1.010568	-0.977896	-0.611828	-0.024032	-0.102461	-0.747405	-1.587691	-1.615476
5	-1.283685	-1.755141	-1.698398	-1.062616	-0.041739	-0.177953	-1.298083	-2.757481	-2.805738
6	-2.036190	-2.784018	-2.694011	-1.685529	-0.066207	-0.282271	-2.059029	-4.373936	-4.450483
7	-0.666923	-0.911862	-0.882382	-0.552069	-0.021685	-0.092453	-0.674403	-1.432616	-1.457688
8	-0.900107	-1.230688	-1.190900	-0.745096	-0.029267	-0.124779	-0.910203	-1.933518	-1.967356
9	-0.989782	-1.353297	-1.309545	-0.819327	-0.032183	-0.137210	-1.000884	-2.126149	-2.163357
10	-0.481135	-0.657841	-0.636573	-0.398277	-0.015644	-0.066698	-0.486532	-1.033526	-1.051613
11	-1.109697	-1.517254	-1.468201	-0.918591	-0.036082	-0.153834	-1.122144	-2.383739	-2.425456

Problem 2

Our results for this section don't make sense, and there are a few indications that we may have errors in our code. Our GMM function reached the maximum number of iterations. It's not clear if we set the tolerance level too low or if it simply failed to converge; if it didn't converge then the estimation procedure is incorrect and our results *shouldn't* make sense.

Estimate the parameter values using BLP

α	β	σ_{ib}	σ_I	σ_I^2
1.67671625	$\begin{bmatrix} 40.5539681 \\ -37.20820459 \\ 22.10159763 \\ -9.87790967 \\ -18.56618279 \\ 17.5039916 \\ -32.63598696 \\ 34.20049778 \\ -12.49605257 \\ 0.95162148 \\ -53.64393846 \end{bmatrix}$	$\begin{bmatrix} 0.35372042 \\ 0.4527695 \\ 0.2383683 \end{bmatrix}$	0.005161	-0.121263

Note that we have one more element of β than we should. When we constructed the dummies, we neglected to exclude one dummy variable, but we were unable to re-run our code before submitting our results. Given more time, we would re-run this step, and we suspect that the coefficients would then make more sense.

What are the elasticities for store 9 in week 10?

We are going to use the approximation

$$\eta_j^k \approx \frac{p_k}{s_j} \sum_i (\alpha + \sigma_I I_i)(-s_{ij}s_{ik} + \mathbb{1}_{k=j} s_{ik}).$$

For the purposes of this question i is a singleton and there is no joint ownership.

	0	1	2	3	4	5	6	7	8
0	-39.302792	-0.019898	-0.006467	-0.015919	-0.006964	-0.000497	-0.011939	-0.004975	-0.001492
1	-0.029454	-58.177689	-0.009573	-0.023563	-0.010309	-0.000736	-0.017672	-0.007364	-0.002209
2	-0.038587	-0.038587	-76.190310	-0.030869	-0.013505	-0.000965	-0.023152	-0.009647	-0.002894
3	-0.017116	-0.017116	-0.005563	-33.804145	-0.005991	-0.000428	-0.010270	-0.004279	-0.001284
4	-0.031994	-0.031994	-0.010398	-0.025595	-63.174270	-0.000800	-0.019197	-0.007999	-0.002400
5	-0.050743	-0.050743	-0.016492	-0.040595	-0.017760	-100.178617	-0.030446	-0.012686	-0.003806
6	-0.016390	-0.016390	-0.005327	-0.013112	-0.005737	-0.000410	-32.367476	-0.004098	-0.001229
7	-0.020201	-0.020201	-0.006565	-0.016160	-0.007070	-0.000505	-0.012120	-39.884948	-0.001515
8	-0.024011	-0.024011	-0.007804	-0.019209	-0.008404	-0.000600	-0.014406	-0.006003	-47.403955
9	-0.010221	-0.010221	-0.003322	-0.008177	-0.003577	-0.000256	-0.006133	-0.002555	-0.000767
10	-0.027156	-0.027156	-0.008826	-0.021725	-0.009505	-0.000679	-0.016293	-0.006789	-0.002037

These elasticities are not believable. The important/theoretical difference from the logit elasticities is that elasticities are no longer simply a function of shares, and are now a function of product characteristics as well (so for instance consumers are more likely to switch from one branded product to another than from a branded product to a generic, even if the branded product and generic have similar market shares). The striking problems are the massive magnitudes for own-price elasticities and the uniform negativity of the cross-price elasticities. This is most likely due to an error in our parameter estimates from the previous section.

Back out the marginal costs for store 9 in week 10. How are they different from wholesale costs?

Since we have a single-ownership structure, we can use scalars everywhere instead of matrices. The expression for marginal cost is then

$$mc = \frac{1}{\eta_k} \cdot \left(\frac{s_k}{p_k}\right) + p_k.$$

The marginal costs that we estimate for these firms are

Firm	Marginal Cost
0	3.289996
1	4.869998
2	6.380000
3	2.829996
4	5.289999
5	8.390000
6	2.709997
7	3.339999
8	3.970000
9	1.689992
10	4.490000

These marginal costs are well above the wholesale costs. Because of our incredibly large elasticity estimates, the implied markups in our model are very very small, so $mc \approx p$. However we see that wholesale cost is notably cheaper than actual prices.

Problem 3

Predict the post-merger prices using the logit model, but only for store 9 in week 10

	store	week	brand	prices	new_prices	price_change
33	9	10	1	3.29	3.235631	-0.054369
34	9	10	2	4.87	4.815631	-0.054369
35	9	10	3	6.38	6.325631	-0.054369
36	9	10	4	2.83	2.797369	-0.032631
37	9	10	5	5.29	5.257369	-0.032631
38	9	10	6	8.39	8.357369	-0.032631
39	9	10	7	2.71	2.694294	-0.015706
40	9	10	8	3.34	3.324294	-0.015706
41	9	10	9	3.97	3.954294	-0.015706
42	9	10	10	1.69	1.671601	-0.018399
43	9	10	11	4.49	4.471601	-0.018399

How to predict the change in prices after the merger using the random coefficients model?

The procedure for predicting the effects of a merger using estimates from the random coefficients model is exactly the same. When we estimate the logit model, we end up with unrealistic elasticities, and hence unrealistic substitution patterns. The random coefficients model gives us a more reasonable matrix of elasticities, but we use it in the same way. When we want to assess the change of moving from two single-product firms to a multi-product firm, we switch from looking at firms that take FOCs with respect to a single price to a single firm that takes FOCs with respect to two prices (the price of each good that it produces). What does this mean? With a more realistic estimate of substitution patterns, we have a better idea of how consumers will respond to changes in prices. Pre-merger, a firm would be "unwilling" to raise its price because it would lose customers. Now, however, the merged firm *may* recognize that increasing the price of one of it's products will cause consumers to switch to its other product, so its profit maximizing price may be higher in the merged case than in the pre-merger case. The random coefficients model is giving us a better insight into these actual substitution patterns.

```
!find / -iname 'libdevice'
!find / -iname 'libnvvm.so'

#Add two libraries to numba environment variables:
import os
from scipy.optimize import minimize
os.environ['NUMBAPRO_LIBDEVICE'] = "/usr/local/cuda-10.0/nvvm/libdevice"
os.environ['NUMBAPRO_NVVM'] = "/usr/local/cuda-10.0/nvvm/lib64/libnvvm.so"

#install linear and pyBLP models
!pip install linearmodels
!pip install pyBLP

"""note: must restart runtime"""

# Import Packages

import pandas as pd                     # for data handling
import numpy as np                      # for numerical methods and data structures
import matplotlib.pyplot as plt         # for plotting
import seaborn as sea                   # advanced plotting
import patsy                            # provides a syntax for specifying models
import linearmodels.iv as iv            # provides IV statistical modeling
import statsmodels.api as sm            # provides statistical models like ols, gmm, anova, etc...
import statsmodels.formula.api as smf   # provides a way to directly spec models from formulas
import pyblp                            # for BLP
import time
from numba import njit, prange, cuda    # for acceleration of functions
from statsmodels.iolib.summary2 import summary_col

# Login to drive

from google.colab import auth
auth.authenticate_user()

import gspread
from oauth2client.client import GoogleCredentials

gc = gspread.authorize(GoogleCredentials.get_application_default())

# Download data

otc_data = gc.open_by_url('https://docs.google.com/spreadsheets/d/1YP2uhQ-l4MF2HzjesLa0qcZG0olk-Opqdb1Ew
otc_data = otc_data.worksheet('Sheet1')
otc_dataDf = pd.DataFrame(otc_data.get_all_records())
print(otc_dataDf.head())

otc_demographics = gc.open_by_url('https://docs.google.com/spreadsheets/d/1RL7sbL4YJs9C6hDiDlVZSDb5SWpN6
otc_demographics = otc_demographics.worksheet('Sheet1')
otc_demographicsDf = pd.DataFrame(otc_demographics.get_all_records())
print(otc_demographicsDf.head())

otc_dataInstruments = gc.open_by_url('https://docs.google.com/spreadsheets/d/1H3I-wfVjNQFlUhxkkUF58cWD67
otc_dataInstruments = otc_dataInstruments.worksheet('OTCDataInstruments')
```

```python
otc_dataInstrumentsDf = pd.DataFrame(otc_dataInstruments.get_all_records())
print(otc_dataInstrumentsDf.head())

# Recreate the Summary Table in the PS

def get_summary(data):

  salesTotal = data.groupby('brand')['sales_'].sum()
  data['fullPrice'] = data.price_ + data.prom_
  priceAvg = data.groupby(['brand'])['price_'].mean()
  fullPriceAvg = data.groupby(['brand'])['fullPrice'].mean()
  promoAvg = data.groupby(['brand'])['prom_'].mean()
  costAvg = data.groupby(['brand'])['cost_'].mean()
  marketShare = salesTotal/sum(salesTotal)

  sumTable = pd.concat((salesTotal, marketShare, priceAvg, promoAvg, fullPriceAvg, costAvg), axis=1)
  sumTable = pd.DataFrame(sumTable)
  sumTable.columns = ['salesTotal', 'marketShare', 'priceAvg', 'promoAvg', 'fullPriceAvg', 'costAvg']
  sumTable['sizeTab'] = [25,50,100,25,50,100,25,50,100,50,100]
  sumTable['brandName'] = ['Tylenol', 'Tylenol', 'Tylenol', 'Advil', 'Advil', 'Advil', 'Bayer', 'Bayer'
  sumTable['brandID'] = [1, 1, 1, 2, 2, 2, 3, 3, 3, 4, 4]

  return(sumTable)

sumTable = get_summary(otc_dataDf)
sumTable.to_latex("summary_statistics.tex")

# make a copy of the sales column
adjusted_sales = otc_dataDf.sales_
# and insert it in dataframe
otc_dataDf.insert(loc=4, column='adjusted_sales', value=adjusted_sales)

def adjust_sales(data):
  '''
  3 package sizes - normalize to 25 tabs for market share calculations
  '''

  # divide 50 tab sales by 2
  tab_50 = [2,5,8,10]
  for i in tab_50:
    data.loc[data['brand'] == i, 'adjusted_sales'] = data['adjusted_sales']/2

  # divide 100 tab sales by 4
  tab_100 = [3,6,9,11]
  for i in tab_100:
    data.loc[data['brand'] == i, 'adjusted_sales'] = data['adjusted_sales']/4

  return(data)

otc_dataDf = adjust_sales(otc_dataDf)

def get_shares(data):

  # calucalte adj. shares by market (store-week)
```

```
  data['quant_store_week'] = data.groupby(['store', 'week'])['adjusted_sales'].transform('sum')
  # calculate weekly share
  data['weekly_share'] =  data['adjusted_sales'] / data['quant_store_week']
  # calculate outside share
  data['outshr'] = 1-(data['quant_store_week']/data['count'])
  data['weekly_share'] = data['weekly_share']*(1-data['outshr'])

  # generate logged relative purchase probabilities
  data['Y'] = np.log(data['weekly_share']) - np.log(data['outshr'])

  return data

otc_dataDf = get_shares(data = otc_dataDf)

otc_dataDf.head()

# dummy for branded product
otc_dataDf['branded'] = 1
otc_dataDf.loc[otc_dataDf['brand'] == 10, 'branded'] = 0
otc_dataDf.loc[otc_dataDf['brand'] == 11, 'branded'] = 0

# constant because thats what the example did
otc_dataDf["cons"] = 1

# market variable
otc_dataDf['market_ids'] = otc_dataDf['store'].astype(str) + "." + otc_dataDf['week'].astype(str)
otc_dataDf['market_ids'].astype(str)

# rename some variables
otc_dataDf = otc_dataDf.rename({'price_': 'prices', 'weekly_share': 'shares'}, axis='columns')

# Merge dfs

def combine_data(otc_dataDf, instruments, demographics, sumTable):

  # brand name
  otc_dataDf = pd.merge(otc_dataDf, sumTable['brandID'], left_on=['brand'], right_index=True)
  # instruments
  merged = pd.merge(otc_dataDf, instruments, left_on=["brand", "store", "week"], right_on=["brand", "sto
  # demographics
  full_data = pd.merge(merged, demographics, left_on=["store", "week"], right_on=["store", "week"])

  return(full_data)

full_data = combine_data(otc_dataDf, otc_dataInstrumentsDf, otc_demographicsDf,  sumTable=sumTable)

from google.colab import drive
drive.mount('/content/drive')

# rename some variables
full_data = full_data.rename({'brand': 'product_ids'}, axis='columns')
full_data.head()

"""## 1. Logit"""
```

```
#1.1: Estimate using OLS with price and promotion as product characteristics.
res_log1 = smf.ols('Y ~ prices + prom_', data=otc_dataDf).fit()
res_log1.summary().as_latex()
# print(res_log1.summary())

#1.2: Estimate using OLS with price and promotion as product characteristics and brand dummies.
res_log2 = smf.ols('Y ~ prices + prom_ + C(brand)', data=otc_dataDf).fit()
res_log2.summary().as_latex()

#1.3. Using OLS with price and promotion as product characteristics and store-brand
#(the interaction of brand and store) dummies.

res_log3 = smf.ols('Y ~ prices + prom_ + C(brand)*C(store)', data=otc_dataDf).fit()
res_log3.summary().as_latex()

outputOLS = summary_col([res_log1,res_log2,res_log3],stars=False)
outputOLS

"""4. Estimate the models of 1, 2 and 3 using wholesale cost as an instrument."""

# OLS with price and promotion as product characteristics, using wholesale cost as instrument
wholeSale_IV1 = iv.IV2SLS.from_formula('Y ~ 1 + [prices ~ cost_] + prom_ ', data=otc_dataDf).fit()
wholeSale_IV1.summary.as_latex()
#print(wholeSale_IV1.first_stage)

# OLS with price and promotion as product characteristics and brand dummies
# using wholesale cost as instrument

wholeSale_IV2 = iv.IV2SLS.from_formula('Y ~ 1 + [prices ~ cost_] + prom_ + C(brand)', data=otc_dataDf).
wholeSale_IV2.summary.as_latex()
#print(wholeSale_IV2.first_stage)

# OLS with price and promotion as product characteristics and brand dummies
# using wholesale cost as instrument

wholeSale_IV3 = iv.IV2SLS.from_formula('Y ~ 1 + [prices ~ cost_] + prom_ + C(brand)*C(store)', data=otc
wholeSale_IV3.summary.as_latex()

# blah = pd.read_csv(foo)
#print(wholeSale_IV3.first_stage)

"""5. Estimate the models of 1, 2 and 3 using Hausman instrument."""

hausman_IV1 = iv.IV2SLS.from_formula('Y ~ 1 + [prices ~ pricestore1 + pricestore2 + pricestore3 + prices
pricestore5 + pricestore6 + pricestore7 + pricestore8 + pricestore9 + pricestore10 + pricestore11 + pric
+ pricestore13 + pricestore14 + pricestore15 + pricestore16 + pricestore17 + pricestore18 + pricestore1
pricestore20 + pricestore21 + pricestore22 + pricestore23 + pricestore24 + pricestore25 + pricestore26 
 pricestore27 + pricestore28 + pricestore29 + pricestore30] + prom_', data=full_data).fit()
hausman_IV1.summary.as_latex()
#print(hausman_IV1.first_stage)

hausman_IV2 = iv.IV2SLS.from_formula('Y ~ 1 + [prices ~ pricestore1 + pricestore2 + pricestore3 + prices
hausman_IV2.summary.as_latex()
```

```
#print(hausman_IV2.first_stage)

hausman_IV3 = iv.IV2SLS.from_formula('Y ~ 1 + [prices ~ pricestore1 + pricestore2 + pricestore3 + prices
hausman_IV3.summary.as_latex()
#print(hausman_IV3.first_stage)

def problem1_6(
    model,
    clean_data=otc_dataDf
                ):
  '''
  Take parameter estimates from given model
  Use the analytic form for logit elasticity to calculate elasticities
  Analytic form for logit elasticity from Levin's notes (slide 13)
  https://web.stanford.edu/~jdlevin/Econ%20257/Demand%20Estimation%20Slides%20B.pdf
  '''
  brand_means = otc_dataDf.groupby(by='brand').mean()
  brand_means
  # otc_dataDf

  model_elasticities = \
  1 * model.params['prices'] \
    * brand_means['prices'] \
    * (1 - brand_means['shares'])

  return(model_elasticities)

# list all logit models
model_list = [res_log1, res_log2, res_log3,
              wholeSale_IV1, wholeSale_IV2, wholeSale_IV3,
              hausman_IV1, hausman_IV2, hausman_IV3]

# new dataframe for results
elasticities_df = pd.DataFrame(columns=['OLS1', 'OLS2', 'OLS3',
                                        'IV4.1', 'IV4.2', 'IV4.3',
                                        'IV5.1', 'IV5.2', 'IV5.3'])

# run elasticity function on all models
columnnr = 0
for i in model_list:
  #print(elasticities_df.columns[columnnr])
  elasticities_df[elasticities_df.columns[columnnr]] = problem1_6(model=i, clean_data=otc_dataDf)
  columnnr += 1

elasticities_df.to_latex()

"""## 2: Random-Coefficients Logit, a.k.a. BLP"""

#Manual BLP:
X1 = np.hstack((otc_dataDf[["prices"]], pd.get_dummies(otc_dataDf["brand"])))

# non-linear, note order is different from Nevo paper
X2 = otc_dataDf[["cons", "prices", "prom_"]].to_numpy()
```

```
k = 2

# price
p = otc_dataDf.prices.values

# number of goods per market
J = otc_dataDf.groupby("market_ids").sum().cons.values

# number of simulations per market
N = 3

# number of markets
T = len(J)

# find the share of the outside good
# otc_dataDf["outside"] = .9

# initial delta_0 estimate: log(share) - log(share outside good)
delta_0 = np.log(full_data["shares"]) - np.log(full_data["outshr"])

# markets for itj
markets = otc_dataDf.market_ids.values

# unique markets
marks = np.unique(otc_dataDf.market_ids)

# firms
firms = np.reshape(otc_dataDf.brand.values, (-1,1))

class delta:
    def __init__(self, delta):
        self.delta = np.array(delta)

# initialize a delta object using the delta_0 values
d = delta(delta_0)

# set seed
np.random.seed(4096542)

# matrix of simulated values
#  number of rows = number of simulations
#  treat last column is price
# different draws for each market
V = np.reshape(np.random.standard_normal((k + 1) * N * T), (T * N, k + 1))

# draws for income if same draws in every market
otc_demographicsDf["average_hhincome"] = otc_demographicsDf[[i for i in full_data.columns if i[:-2] == '
incomeMeans = otc_demographicsDf["average_hhincome"].values

demog = incomeMeans
demog2 = demog * demog

demogDf = pd.DataFrame()
demogDf['demog'] = demog
```

```python
demogDf['demog2'] = demog2

demog = demogDf.to_numpy()

sigma_v = np.std(incomeMeans)
m_t = np.repeat(incomeMeans, N)

#@cuda.jit(nopython = True, parallel = True)
def util_iter(out, x2, v, p, demog, delta, sigma, pi, J, T, N):
    # first iterate over the individuals
    for i in prange(N):
        # iterator through t and j
        tj = 0

        # iterate over the markets
        for t in prange(T):
            # market size of market t
            mktSize = J[t]

            # iterate over goods in a particular market
            for j in prange(mktSize):
                # if N * t + i < demog.shape[0]:
                # calculate utility
                #  log of the numerator of equation (11) from Aviv's RA guide
                out[tj, i] = delta[tj] + \
                x2[tj,0] * (v[N * t + i , 0] * sigma[0] +
                             np.dot(pi[0, :], demog[ t,:])) + \
                x2[tj, 1] * (v[N * t + i, 1] * sigma[1]  +
                            np.dot(pi[1,:], demog[ t ,:])) + \
                x2[tj, 2] * (v[N * t + i, 2] * sigma[2]  +
                            np.dot(pi[2,:], demog[ t ,:]))
                # else:
                # print(f"{t + i}")

                tj += 1
    return out

# computes indirect utility given parameters
#  x: matrix of demand characteristics
#  v: monte carlo draws of N simulations
#  p: price vector
#  delta: guess for the mean utility
#  sigma: non-linear sigma (sigma - can think of as stdev's)
#  J: vector of number of goods per market
#  T: numer of markets
#  N: number of simulations

#@cuda.jit
def compute_indirect_utility(x2, v, p, demog, delta, sigma, pi, J, T, N):
    # make sure sigma are positive
    sigma = np.abs(sigma)

    # output matrix
    out = np.zeros((sum(J), N))
```

```
    # call the iteration function to calculate utilities
    out = util_iter(out, x2, v, p, demog, delta, sigma, pi, J, T, N)

    return out



# computes the implied shares of goods in each market given inputs
# same inputs as above function

#@cuda.jit
def compute_share(x2, v, p, demog, delta, sigma, pi, J, T, N):
    q = np.zeros((np.sum(J), N))

    # obtain vector of indirect utilities
    u = compute_indirect_utility(x2, v, p, demog, delta, sigma, pi, J, T, N)

    # exponentiate the utilities
    exp_u = np.exp(u)

    # pointer to first good in the market
    first_good = 0

    for t in range(T):
        # market size of market t
        mktSize = J[t]

        # calculate the numerator of the share eq
        numer = exp_u[first_good:first_good + mktSize,:]

        # calculate the denom of the share eq
        denom = 1 + numer.sum(axis = 0)

        # calculate the quantity each indv purchases of each good in each market
        q[first_good:first_good + mktSize,:] = numer/denom

        first_good += mktSize

    # to obtain shares, assume that each simulation carries the same weight.
    # this averages each row, which is essentially computing the sahres for each
    #  good in each market.
    s = np.matmul(q, np.repeat(1/N, N))

    return [q,s]

#@cuda.jit
def solve_delta(s, x2, v, p, demog, delta, sigma, pi, J, T, N, tol):
    # define the tolerance variable
    eps = 10

    # renaming delta as delta^r
    delta_old = delta
```

```python
    while eps > tol:
        # Aviv's step 1: obtain predicted shares and quantities
        q_s = compute_share(x2, v, p, demog, delta_old,
                            sigma, pi, J, T, N)

        # extract the shares
        sigma_jt = q_s[1]

        # step 2: use contraction mapping to find delta
        delta_new = delta_old + np.log(s/sigma_jt)

        # update tolerance
        eps = np.max(np.abs(delta_new - delta_old))

        delta_old = delta_new.copy()

    return delta_old

# This is the objective function that we optimize the non-linear parameters over
def objective(params, s, x1, x2, v, p, demog, J, T, N, marks, markets, tol,
              Z, weigh, firms):

    # optim flattens the params, so we have to redefine inside
    sigma = params[0:3]

    alpha = sigma[-1]

    pi = params[3:].reshape((3,2))

    # number of observation JxT
    obs = np.sum(J)

    # force these params to be 0:

    if np.min(sigma) < 0:
        return 1e20

    else:
        # Aviv's step 1 & 2:
        d.delta = solve_delta(s, x2, v, p, demog, d.delta, sigma, pi, J, T, N, tol)

        # since we are using both demand and supply side variables,
        #  we want to stack the estimated delta and estimated mc
        y2 = d.delta.reshape((-1,1))

        # get linear parameters (this FOC is from Aviv's appendix)
        b = np.linalg.inv(x1.T @ Z @ weigh @ Z.T @ x1) @ (x1.T @ Z @ weigh @ Z.T @ y2)

        # Step 3: get the error term xi (also called omega)
        xi_w = y2 - x1 @ b

        g = Z.T @ xi_w / np.size(xi_w, axis = 0)

        obj = float(obs ** 2 * g.T @ weigh @ g)
```

```
        return obj

demand_inst_cols = [i for i in otc_dataInstrumentsDf.columns if i[:10] == "pricestore"]
Z = np.hstack((otc_dataInstrumentsDf[demand_inst_cols], pd.get_dummies(full_data["product_ids"])))

# linear parameters
#  this is the FOC on page 5 of Aviv's Appendix
# compare parameters to Table iv of BLP
#  first 5 are the demand side means
#  last 6 are the cost side params


# initial estimates for sigma
sigma = [0, 0, 0]

# initial estimates for pi, by row
pi1 = [0, 0, 0]
pi2 =   [0, 0, 0]


# optim must read all params in as a one dimensional vector
params = np.hstack((sigma, pi1, pi2))

# Recommended initial weighting matrix from Aviv's appendix
w1 = np.linalg.inv(Z.T @ Z)



y2 = d.delta.reshape((-1, 1))
b = np.linalg.inv(X1.T @ Z @ Z.T @ X1) @ (X1.T @ Z @ Z.T @ y2)
b

obj = objective(params,
                full_data.shares.values,
                X1, X2,
                V, p, demog,
                J, T, N,
                marks, markets, 1e-6,
                Z, w1, firms)

pi = params[3:].reshape((3,2))


util = compute_indirect_utility(X2, V, p, demog, d.delta,
                         sigma, pi, J, T, N)
#
#
#
share = compute_share(X2, V, p,
                      demog, d.delta, sigma, pi,
                      J, T, N)
#
delta = solve_delta(full_data.shares.values,
```

```
                    X2, V, p,
                    demog, d.delta, sigma, pi,
                    J, T, N, 1e-4)

delta

res_init_wt = minimize(objective,
                      params,
                      args = (full_data.shares.values,
                            X1, X2,
                            V, p, demog,
                            J, T, N,
                            marks, markets, 1e-4,
                            Z, w1, firms),
                      method = "Nelder-Mead")

res_init_wt

obs = np.sum(J)

# approximate optimal weighting matrix
params_2 = res_init_wt.x
sigma_2 = params_2[0:4]
pi_2 = params_2[3:].reshape((3,2))


# calculate mean utility given the optimal parameters (with id weighting matrix)
d.delta = solve_delta(full_data.shares.values,
                    X2, V, p,
                    demog, d.delta, sigma_2, pi_2,
                    J, T, N, 1e-4)

# since we are using both demand and supply side variables,
#  we want to stack the estimated delta and estimated mc
y2 = d.delta.reshape((-1,1))

# this is the first order condition that solves for the linear parameters
b = np.linalg.inv(X1.T @ Z @ w1 @ Z.T @ X1) @ (X1.T @ Z @ w1 @ Z.T @ y2)

# obtain the error
xi_w = y2 - X1 @ b

# update weighting matrix
g_ind = Z * xi_w
vg = g_ind.T @ g_ind / obs

# obtain optimal weighting matrix
weight = np.linalg.inv(vg)

res = minimize(objective,
              params,
              args = (full_data.shares.values,
                    X1, X2,
                    V, p, demog,
```

```
                    J, T, N,
                    marks, markets, 1e-4,
                    Z, weight, firms),
              method = "Nelder-Mead")

params_3

params_3 = res.x
sigma_3 = params_3[0:3]
pi_3 = params_3[3:].reshape((3,2))

# obtain the actual implied quantities and shares from converged delta
q_s = compute_share(X2, V, p, demog, d.delta, sigma_3, pi_3, J, T, N)

delta

q_s[1]

sigma_3

pd.DataFrame(pi_3)

b

full_data_w10 = full_data.loc[full_data['week']==10].reset_index()
full_data_w10_store9 = full_data_w10[full_data_w10['store']==9].reset_index()
full_data_w10_store9['relative_share'] = full_data_w10_store9['shares']/full_data_w10_store9['shares'].s
full_data_w10_store9['average_income'] = full_data_w10_store9[[i for i in full_data_w10_store9.columns i
full_data_w10_store9

sigma_I = 0.005161
sigma_I_sq = -0.121263
alpha = 1.67671625

def calculate_elasticity_matrix(full_data_w10_store9=full_data_w10_store9):
  my_etas = []
  Q2_elasticty_matrix = pd.DataFrame()
  for j in full_data_w10_store9.index:
    eta_j = []
    for k in full_data_w10_store9.index:
      if j != k:
        eta = (full_data_w10_store9['prices'][k]/full_data_w10_store9['shares'][k]) * \
          (alpha + sigma_I * full_data_w10_store9['average_income'][k] + sigma_I_sq * full_data_w10_stor
          (full_data_w10_store9['shares'][k] * full_data_w10_store9['shares'][j])
      if j == k:
        eta = (full_data_w10_store9['prices'][k]/full_data_w10_store9['shares'][k]) * \
          (alpha + sigma_I * full_data_w10_store9['average_income'][k] + sigma_I_sq * full_data_w10_stor
          (full_data_w10_store9['shares'][k] * full_data_w10_store9['shares'][j] + full_data_w10_store9
      eta_j.append(eta)
    Q2_elasticty_matrix[f'{j}'] = eta_j
  return Q2_elasticty_matrix


Q2_elasticty_matrix = calculate_elasticity_matrix()
```

```
Q2_elasticty_matrix

def calculate_mc(df, elasticity_df=Q2_elasticty_matrix):
  elasticities = elasticity_df.values.diagonal()
  shares = df['shares']
  prices = df['prices']
  marginal_costs = []
  k = 0
  while k < len(df.index):
    mc = 1 / elasticities[k] * shares[k] / prices[k] + prices[k]
    marginal_costs.append(mc)
    k += 1
  return marginal_costs

marginal_costs = calculate_mc(df=full_data_w10_store9)
marginal_costs

marginal_costs_df = pd.DataFrame(data=marginal_costs)
marginal_costs_df.to_latex()

mc = 1 / (elasticity[k] * share[k] / price[k])

df = full_data_w10_store9
elasticity_df = Q2_elasticty_matrix
elasticities = elasticity_df.values.diagonal()


marginal_costs

len(elasticities)

# Define the function to retrieve costs
def find_costs(nobs, ahat, price, share, owner):
    """Solve for marginal costs
    nobs = number of products in market
    ahat = estimated price coefficient
    price = vector of prices in market
    share = market shares in market
    owner = ownership vector
    """
    # Initialize dsdp
    temp = np.zeros(shape=(nobs,nobs))
    dsdp = pd.DataFrame(temp)

    # dsdp matrix where element (row=j,col=r) = ds_r/dp_j
    for j in range(nobs):
        for r in range(nobs):
            if (owner.at[j]==owner.at[r]):
                if (j==r):
                    dsdp[j][r] = ahat*share.at[j]*(1-share.at[j])
                else:
                    dsdp[j][r] = -ahat*share.at[j]*share.at[r]

    # Apply inverse. If you've had linear algebra, you'll see what I'm doin.
```

```python
    # If not, take my word for it: We're just rearranging the system of FOCs
    # to solve for c just like you would in the monopolist case.
    inv_dsdp = np.linalg.inv(dsdp)

    # Solve for dollar markups.
    markup = -np.dot(inv_dsdp,share)

    # Solve for chat
    chat = price - markup

    return chat

params_M1 = res_log3.params
params_M1

# restrict data to week 10
full_data_w10 = full_data.loc[full_data['week']==10].reset_index()
full_data_w10['chat'] = 0

# Identify inputs
ahat = params_M1['prices']
price = full_data_w10['prices']
share = full_data_w10['shares']
owner = full_data_w10['brandID']
nobs = len(share)

full_data_w10['chat'] = find_costs(nobs, ahat, price, share, owner)
full_data_w10['chat'].describe()

# Define function to solve for market shares
def mkt_shar(nobs,ahat,price,util):
    numerator = np.exp(ahat*price + util)
    denominator = sum(numerator)
    denominator = denominator + 1
    share = numerator/denominator
    return share

# Define the system of FOCs
def FOC(price, nobs, ahat, util, owner, chat):
    """Solve for the FOCs at a price guess
    nobs = number of products in market
    ahat = estimated price coefficient
    price = vector of prices in market
    util = X-times-Beta
    owner = ownership vector
    chat = marginal cost estimate
    """

    # Solve for market share conditional on price
    share = mkt_shar(nobs,ahat,price,util)

    # Initialize dsdp
    temp = np.zeros(shape=(nobs,nobs))
    dsdp = pd.DataFrame(temp)
```

```
    # dsdp matrix where element (row=j,col=r) = ds_r/dp_j
    for j in range(nobs):
        for r in range(nobs):
            if (owner.at[j]==owner.at[r]):
                if (j==r):
                    dsdp[j][r] = ahat*share.at[j]*(1-share.at[j])
                else:
                    dsdp[j][r] = -ahat*share.at[j]*share.at[r]

    # Apply inverse. If you've had linear algebra, you'll see what I'm doin.
    # If not, take my word for it: We're just rearranging the system of FOCs
    # to solve for c just like you would in the monopolist case.
    inv_dsdp = np.linalg.inv(dsdp)

    # Solve for dollar markups.
    markup = -np.dot(inv_dsdp,share)

    # Solve for residual. If the FOCs jointly hold, this is a vector of zeros
    resid = price - (chat + markup)

#     print(sum(resid))

    return resid

# Solve for new prices using a numerical equation solver (fsolve)
import time
from scipy.optimize import fsolve

# Account for the merger via the ownership matrix
full_data_w10.loc[(full_data_w10['brandID']== 2),'brandID'] = 1
full_data_w10.loc[(full_data_w10['brandID']== 3),'brandID'] = 1

# Define util (X-times-Beta)
util = full_data_w10['Y'] - ahat*price

chat = full_data_w10['chat']
p0 = price # initial guess of equilibrium prices (prices from data)

"""This is a faster way to solve

"""

def contraction(price, nobs, ahat, util, owner, chat):
    """Solve for the FOCs at a price guess
    nobs = number of products in market
    ahat = estimated price coefficient
    price = vector of prices in market
    util = X-times-Beta
    owner = ownership vector
    chat = marginal cost estimate
    """
    iter = 1
    maxiter = 1000
```

```
    norm = 1
    tol = 1e-6

    pnew = price # Initial guess

    while (iter<=maxiter) & (norm > tol):

        pold = pnew

        # Solve for market share conditional on price
        share = mkt_shar(nobs,ahat,pold,util)

        # Initialize dsdp
        temp = np.zeros(shape=(nobs,nobs))
        dsdp = pd.DataFrame(temp)

        # dsdp matrix where element (row=j,col=r) = ds_r/dp_j
        for j in range(nobs):
            for r in range(nobs):
                if (owner.at[j]==owner.at[r]):
                    if (j==r):
                        dsdp[j][r] = ahat*share.at[j]*(1-share.at[j])
                    else:
                        dsdp[j][r] = -ahat*share.at[j]*share.at[r]

        # Apply inverse. If you've had linear algebra, you'll see what I'm doin.
        # If not, take my word for it: We're just rearranging the system of FOCs
        # to solve for c just like you would in the monopolist case.
        inv_dsdp = np.linalg.inv(dsdp)

        # Solve for dollar markups.
        markup = -np.dot(inv_dsdp,share)

        # Solve for residual. If the FOCs jointly hold, this is a vector of zeros
        pnew = chat + markup

        # Check if we're done
        norm = max(abs(pnew-pold))
        iter = iter + 1
        #print(norm)

    return pnew

# Solve the faster way
start_time = time.time()  # Start timer
p_prime2 = contraction(price, nobs, ahat, util, owner, chat)
finish_time = time.time()  # end timer
print('FOC Algorithm Finished. Execution Time: {0:.2f} seconds'.format(finish_time-start_time))

pchange = 100*(p_prime2/price - 1)  # Percentage change
p_delta = p_prime2-price

# Raw stats
pchange.describe()
```

```python
# Need to merge prices back into frame

# price changes in week 10 - store 9
p_prime2.name = 'new_prices'
df = pd.concat([full_data_w10, p_prime2], axis=1)
df = pd.concat([df, p_delta], axis=1)
df = df.rename({0: 'price_change', "product_ids" : "brand"}, axis='columns')
df[df["store"] == 9][['store', 'week', 'brand', 'prices', 'new_prices', 'price_change']]
```