LBA: Modeling Grocery Prices In Berlin

Ash

2018/12/06

1 Data Cleaning

Data from the original recorded responses spreadsheet was cleaned in three steps. First, a neighborhood column is added using Excel's internal functions to clean the responses address (strip the entered responses leaving only the street name and numbers) then these addresses are matched with the given neighborhood in the store assignment spreadsheet. Second, Python codes are used to transcribed and encoded the brand names for each product: about 10 most numerous brands were retained (along with no brand) and all other different responses are coded as "others". Lastly, Python code is used to translate the response spreadsheet into a table with 5 columns: "product" expressing the product name (like "apple"); "store brand" expressing the name of the store (like "EDEKA"); "neighborhood" expressing tbe name of the neighborhood (like "Kreuzberg"); "brand" expressing the product brand name (like "gut and gunstig") and "price" expressing the price of the item. Some first item of the cleaned data is shown in Figure 1. The full cleaned data is included in the appendix in csv format.

```
import numpy as np
import matplotlib.pyplot as plt
import pystan
import pandas as pd

data = pd.read_csv('/Users/ash/Downloads/cleaned_146_lba.csv')
data = data.drop(columns = ['Unnamed: 0'])
data

--->Output:
Figure 1
```

2 Modeling

The model in PyStan would be specified as following:

	product	store_brand	neighborhood	brand	price
0	apple	ALDI	Neukölln	bio	4.38
1	apple	ALDI	Neukölln	evelina	1.99
2	apple	ALDI	Neukölln	others	2.79
3	apple	ALDI	Schöneberg	others	2.72
4	apple	ALDI	Schöneberg	braeburn	1.39
5	apple	ALDI	Schöneberg	others	2.99
6	apple	ALDI	Lichtenberg	no brand	2.72
7	apple	ALDI	Lichtenberg	others	2.79
8	apple	ALDI	Lichtenberg	no brand	2.39
9	apple	Lidl	Kreuzberg	bio	3.13
10	apple	Lidl	Kreuzberg	no brand	1.00
11	apple	Lidl	Kreuzberg	no brand	1.40
12	apple	EDEKA	Mitte	others	2.93
13	apple	EDEKA	Mitte	others	1.99
14	apple	EDEKA	Mitte	evelina	1.99
15	apple	Lidl	Prenzlauer Berg	no brand	2.49
16	apple	Lidl	Prenzlauer Berg	no brand	1.89
17	apple	Lidl	Prenzlauer Berg	no brand	2.49
18	apple	Lidl	Prenzlauer Berg	oaklands	1.69
19	apple	Lidl	Prenzlauer Berg	no brand	2.24
20	apple	Lidl	_London	no brand	2.24

Figure 1: Some Examples Of Cleaned Data

- For each datapoint (each row in the cleaned data), we will encoded the store product

name, store, product brand and location in terms of a vector of binary entry: 1 if the datapoint is of that product/store/brand/location, and 0 otherwise. These vectors will be submitted to Stan upon sampling

- Each multiplier (for store, brand and location) along with the product base price will be a row vector that has each item sampled from a log-normal distribution with mean 0 and standard deviation 0.25. This specification will make the log-normal distribution roughly centered around 1, as we would expect for the priors of multipliers. The log-normal can be interpreted as the product of independent random variables, which aligns with our interpretation of the multipliers. The log-normal specification makes it skewed slightly to the right: this can be justified if we assume that in general the multiplier has a tendency to decrease the price compared to the average price a bit, which might makes the product more competitive

- Each base price prior is a broad Cauchy distribution to represents our initial belief that we don't know the base price of the product very well. We also specify the base prices to be positive, so Stan will actually sample these base prices from a truncated Cauchy distribution

- Finally, the actual price of the product (that we collected) will be sampled from a normal distribution (which is also truncated given that we specify the price to be positive) centered around the product of the base price and all multiplier associated with the datapoint. Since we specified the multipliers for store/brand/location to be a row vector, multiplying this vector with the 1/0 vector for each store/brand/location will result in a single base price, store multiplier, brand multiplier and location multiplier being used to center this normal distribution for each datapoint

Since in total we will have around 100 different priors for all the brand multiplier, we temporary encode the brand to be "brand" if there is a brand associated with the datapoint and "no brand" otherwise. This will resulted in only 2 priors (and 2 posteriors) for the brand. We will investigate the effect of each brand more carefully later.

```
model_spec_2 = """

data {

//Number of store
    int<lower=1> S;
//Number of product brand
    int<lower=1> B;
//Number of location
    int<lower=1> L;
//Number of product
    int<lower=1> P;
//Number of datapoint
    int<lower=1> N;
```

```
15
16 //A vector of 0/1 in each entry expressing which product, store, brand, or
       location the datapoint is
17     vector<lower=0>[P] product[N];
18     vector<lower=0>[S] store[N];
19     vector<lower=0>[B] brand[N];
20     vector<lower=0>[L] loc[N];
21
22 //Price of the datapoint
23     real<lower=0> price[N];
24 }
25
26 parameters {
27
28 //Vector of size P of base price of each product
29     row_vector<lower=0>[P] base;
30 //Vector of size S for each store multiplier
31     row_vector<lower=0>[S] mul_S;
32 //Vector of size B for each brand multiplier
33     row_vector<lower=0>[B] mul_B;
34 //Vector of size L for each location multiplier
35     row_vector<lower=0>[L] mul_L;
36 }
37
38 model {
39
40 //Sampling each base price from a broad Cauchy
41     for (p in 1:P){
42         base[p] ~ cauchy(0,1);
43     }
44 //Sampling each multipler from a lognormal, with mean and
45 //variance chosen such that it's centered around 1
46     for (s in 1:S){
47         mul_S[s] ~ lognormal(0,0.25);
48     }
49
50     for (b in 1:B){
51         mul_B[b] ~ lognormal(0,0.25);
52     }
53
54     for (l in 1:L){
55         mul_L[l] ~ lognormal(0,0.25);
56     }
57 //For each datapoint, sampling the price from a truncated normal centered
58 //around the product of the base price with all multiplier for store, brand
       , location
59     for(i in 1:N) {
60         price[i] ~ normal(((base*product[i])*(mul_S*store[i])*(mul_B*brand[
       i])*(mul_L*loc[i])), 1);
61     }
62 }
63
64 """
65 model_2 = pystan.StanModel(model_code=model_spec_2)
```

3 Results & Interpretations

Using a function to convert the original cleaned data into a dictionary that Stan can understand then sampling from the model specified in the previous section shows convergence:

```
def brand_no_brand(brand):
    if brand != 'no brand':
        brand = 'brand'
    else:
        brand = 'no brand'
    return brand

data_2 = data

def produce_stan_data_2(data=data_2):
    data['brand'] = data['brand'].apply(brand_no_brand)
    processed = data
    processed = processed.sample(frac=1.0)

    product = pd.get_dummies(processed['product'])
    name_p = product.columns
    product = np.array(product, dtype=int)

    store = pd.get_dummies(processed['store_brand'])
    name_s = store.columns
    store = np.array(store, dtype=int)

    brand = pd.get_dummies(processed['brand'])
    name_b = brand.columns
    brand = np.array(brand, dtype=int)

    loc = pd.get_dummies(processed['neighborhood'])
    name_l = loc.columns
    loc = np.array(loc, dtype=int)

    price = np.array(processed['price'], dtype=float)

    data_dict = {
        'P': product.shape[1],
        'S': store.shape[1],
        'B': brand.shape[1],
        'L': loc.shape[1],
        'N': store.shape[0],

        'product': product,
        'store': store,
        'brand': brand,
        'loc': loc,

        'price': price
    }

    name_dict = {
```

```
        'product': name_p,
        'store': name_s,
        'brand': name_b,
        'loc': name_l
    }
    return data_dict, name_dict

stan_data_2, name = produce_stan_data_2()
results = model_2.sampling(data=stan_data_2)
results

--->Output:

Figure 2
```

```
Inference for Stan model: anon_model_03f3dfce78a7d84c773ca91cd8948a85.
4 chains, each with iter=2000; warmup=1000; thin=1;
post-warmup draws per chain=1000, total post-warmup draws=4000.

             mean se_mean     sd   2.5%    25%    50%    75%  97.5%  n_eff   Rhat
base[1]      1.65    0.01   0.34   1.09   1.41   1.62   1.86   2.39    814    1.0
base[2]       1.1  7.9e-3   0.23   0.72   0.93   1.08   1.24   1.62    848    1.0
base[3]      2.86    0.02   0.58   1.92   2.44   2.81   3.21   4.18    813    1.0
base[4]      7.59    0.05   1.52   5.05   6.48   7.45   8.53   11.0    811    1.0
base[5]      1.94    0.01   0.39   1.28   1.65   1.91   2.18   2.84    826    1.0
base[6]      0.78  5.6e-3   0.17   0.51   0.66   0.76   0.88   1.15    877    1.0
base[7]      0.72  5.2e-3   0.15   0.47    0.6    0.7   0.81   1.07    874    1.0
base[8]      0.96  6.8e-3    0.2   0.62   0.81   0.93   1.08    1.4    879    1.0
base[9]      1.82    0.01   0.37   1.22   1.55   1.79   2.06   2.68    830    1.0
base[10]     2.68    0.02   0.54   1.78   2.29   2.63   3.01   3.88    817    1.0
mul_S[1]      1.0  3.5e-3   0.12   0.78   0.91   0.99   1.08   1.26   1213    1.0
mul_S[2]     1.27  4.4e-3   0.15   0.99   1.16   1.26   1.37   1.59   1217    1.0
mul_S[3]      0.8  2.8e-3    0.1   0.63   0.74    0.8   0.87   1.01   1215    1.0
mul_S[4]     1.27  4.5e-3   0.15   0.99   1.16   1.26   1.37    1.6   1179    1.0
mul_B[1]     1.25  6.2e-3    0.2   0.89    1.1   1.23   1.38   1.69   1096    1.0
mul_B[2]     1.01  5.0e-3   0.17   0.71   0.89   0.99   1.11   1.37   1111    1.0
mul_L[1]     1.33  3.6e-3   0.12   1.11   1.24   1.32    1.4   1.57   1078    1.0
mul_L[2]     1.22  3.3e-3   0.09   1.05   1.16   1.22   1.28   1.42    817    1.0
mul_L[3]     1.02  2.7e-3   0.08   0.87   0.97   1.02   1.07   1.18    818    1.0
mul_L[4]     0.88  2.4e-3   0.07   0.74   0.83   0.87   0.93   1.03    926    1.0
mul_L[5]     1.08  2.9e-3   0.08   0.92   1.02   1.07   1.13   1.24    822    1.0
mul_L[6]     1.04  2.8e-3   0.08   0.89   0.98   1.04   1.09   1.21    834    1.0
mul_L[7]     1.06  2.9e-3   0.08   0.91    1.0   1.06   1.11   1.23    817    1.0
mul_L[8]     0.97  2.7e-3   0.08   0.83   0.92   0.97   1.02   1.13    841    1.0
mul_L[9]     0.92  2.6e-3   0.08   0.76   0.86   0.91   0.97   1.09   1042    1.0
mul_L[10]    0.85  2.3e-3   0.08   0.71    0.8   0.85    0.9    1.0   1064    1.0
lp__        -3014    0.09   3.49  -3022  -3016  -3014  -3012  -3008   1399    1.0

Samples were drawn using NUTS at Sat Nov 10 02:38:23 2018.
For each parameter, n_eff is a crude measure of effective sample size,
and Rhat is the potential scale reduction factor on split chains (at
convergence, Rhat=1).
```

Figure 2: Sampling Results Of The Main Model

To better visualize the results, we use a function to plot the posterior samples for each

of the product, store, brand and location multipliers:

```
def plot_multiplier_variance_2(results, name, what):
    if what == 'store':
        what_ = 'mul_S'
    elif what == 'product':
        what_ = 'base'
    elif what == 'brand':
        what_ = 'mul_B'
    else:
        what_ = 'mul_L'
    mul = results[what_]
    plt.figure(figsize=(12,8))
    for _ in range(mul.shape[1]):
        plt.hist(mul[:,_], alpha=0.35, bins=30, density=True, label=str(name[what][_])+': {0:.3f}'.format(mul[:,_].mean()))
        plt.scatter(mul[:,_].mean(),0, marker='x', s=40)
    plt.legend()
    plt.title('Multiplers for {}'.format(what))
    mul = mul.reshape(-1,1)
    plt.figure(figsize=(12,8))
    plt.hist(mul, alpha=0.35, bins=30, density=True, label='var: {0:.3f}'.format(mul.var()))
    plt.scatter(mul.mean(), 0, marker='x', s=40, label='mean: {0:.3f}'.format(mul.mean()))
    plt.title('All multipliers')
    plt.legend()
    plt.show()
    print('95% CI for all {} multipliers is:'.format(what), (np.percentile(mul,2.5),np.percentile(mul,97.5)))
    return mul

def summarize_everything(results, name):

    print('***'*39)
    print('Summary of sampling results')
    print('***'*39)

    dumb = []
    for _ in ['product', 'store', 'brand', 'loc']:
        print('___{}___'.format(_.upper()))
        dumb.append(plot_multiplier_variance_2(results,name,_))

    print('___EFFECT OF EACH MULTIPLIER___')
    plt.figure(figsize=(12,8))
    for i in range(len(dumb)-1):
        dumb_2 = ['store', 'brand', 'loc']
        plt.hist(dumb[i+1], alpha=0.35, bins=30, density=True, label='{0}, var: {1:.3f}'.format(dumb_2[i],dumb[i+1].var()))
        plt.scatter(dumb[i+1].mean(), 0, marker='x', s=40, label='mean: {0:.3f}'.format(dumb[i+1].mean()))
    plt.axvline(x=1, color='r')
    plt.legend()
    plt.title('Effect of each multipler')
```

```
    plt.show()

summarize_everything(results, name)

--->Output:
95% CI for all store multipliers is: (0.6727573935488614,
    1.5220204924058733)

95% CI for all brand multipliers is: (0.7554100564849898,
    1.634591710543536)

95% CI for all loc multipliers is: (0.7463949810185662, 1.406546021090953)

Figure 3, 4, 5, 6, 7
```

Looking at figure 3, we can see the base price of the product (by its sampled posterior values) with their means. We can clearly see the data reflecting on the posteriors: first, since we all data are in euros we can see the Cauchy priors becoming posteriors that has small means and peaks. Second, the variance of the data is indicated in the variance of the posteriors: the chicken (centered around 7.5 euros) has quite large variance compared to the others because the data clearly shows a large variance for the price of the chicken: there is a large different between "bio" chicken price and normal chicken price, and this inconsistency is expressed by the large variance of the chicken base price posterior.

In Figure 4a we can see that the posterior for EDEKA and REWE indicate an overall higher multiplier compared to Lidl. Aldi largely does not modify the base price much (with mean 0.992) which means it's very close to the base price, whereas Lidl has a multiplier that is smaller than 1 which means it decrease the price in general (it's the cheapest among the four stores in general). EDEKA and REWE has the roughly the same multiplier, meaning that these two stores share similar prices for most products. We can also see higher variance in the posteriors of REWE and EDEKA compared to the other two stores, which means either we have more data for the two other stores, or the data for these two stores are just more varied. When combining all sampled multipliers for the 4 stores we can see a large range of values in figure 4b, and by the output of the code above the 95% CI for store multipliers is around $(0.673, 1.522)$, which is quite a large range. This indicates the store multipliers contribute quite a bit to the variation in the collected prices.

In general by looking at figure 5 we can see by having a brand the price of the product can increase quite a lot (by a factor of 1.2) whereas having no brand will leave the price nearly unchanged. However we can see that there is much overlap between the two histograms, indicating that we are not too sure about this effect. There are two possible cause for this: either our coding scheme of the data and the process of cleaning it cross-contaminate the prices of brand/no brand product, or there are brands that actually does not increase, or even decrease, the product price. The first hypothesis is quite plausible since in the process of cleaning the data we might have included cate-

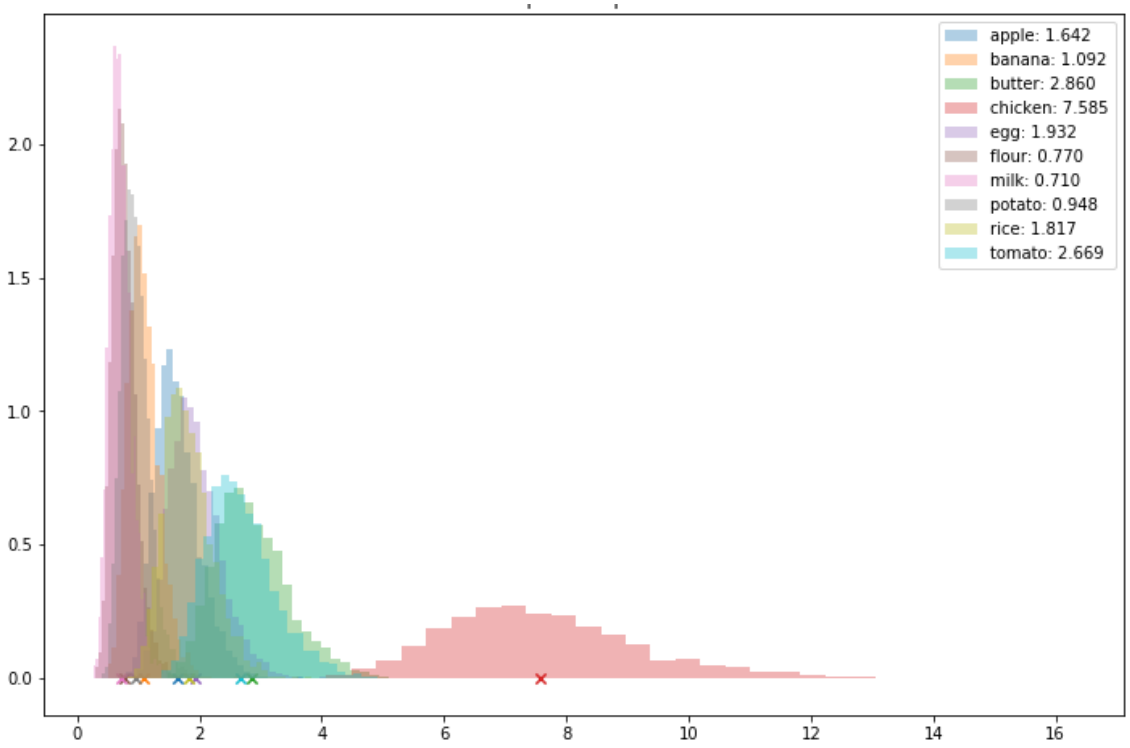
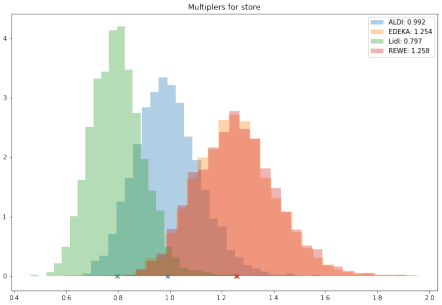


Figure 3: Posteriors Of Product Base Prices

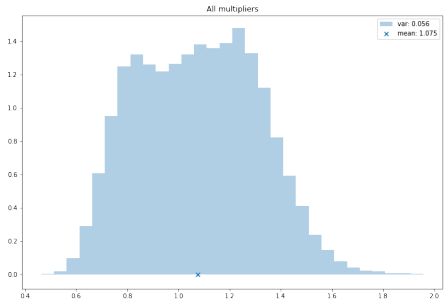
gories of products to be brand (essentially transforming a brandless product to having a brand by its categories, like classifying "royal gala" apples to be apples with brand "royal gala") or the other way around: the collector of the data mistakenly recorded some products to be "no brand" because he/she cannot find the brand name on the products, but they essentially had brands. Of course, it is also entirely possible that the effect of having a brand is different for each product (for example, it's imaginable that a brandless bannana won't have much effect on its price compared to a brandless chicken, because many people do not really pay attention to a banana brand when buying it).

In figure 6a we can see the Treptow and Friedrichshain neighborhood have bigger multipliers than the rest (around 1.2), but they also has bigger variances. Interestingly, only Lichtenberg has significant lower price multiplier, and we can see see that these posteriors indicate no spatial correlation between the regions: Treptow and Tempelhof is quite close, but they have different multipliers, as they should be since we sampled each from an independent log-normal distribution.

Finally, plotting all multipliers in a plot (Figure 7) it seems that location has the least contribution to the price variations we observed in the data, because its posterior has smaller variance and largely center around 1. The brand mostly tends to increase the price (its posterior skews to the left) and the store can either increase or decrease the

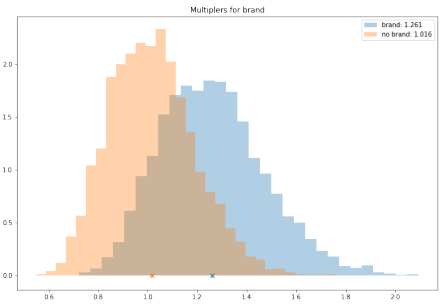


(a) Each Multiplier Posterior With Its Mean

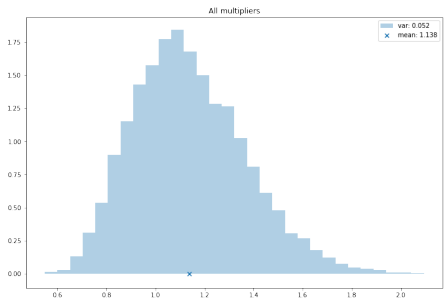


(b) All Posteriors Combined

Figure 4: Posterior Histograms For Store Multipliers

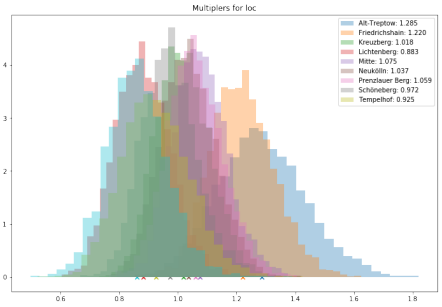


(a) Each Multiplier Posterior With Its Mean

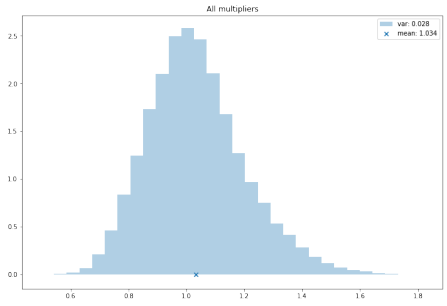


(b) All Posteriors Combined

Figure 5: Posterior Histograms For Brand Multipliers



(a) Each Multiplier Posterior With Its Mean



(b) All Posteriors Combined

Figure 6: Posterior Histograms For Location Multipliers

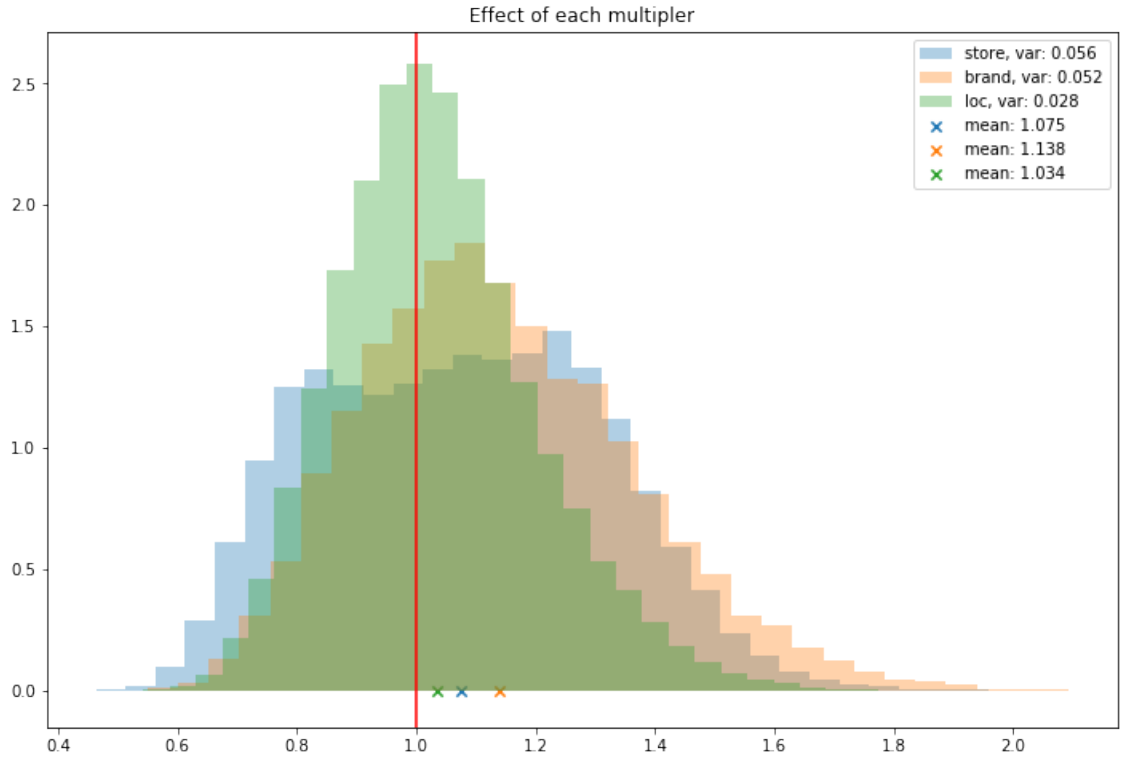


Figure 7: Posteriors Of All Multipliers

price. The store and the brand have large variance, meaning that they contribute a lot to the price variations we observed.

To determine the correlation between the sampled multiplier and the general cost of living in each neighborhood (reflects by the rental price for each neighborhood), we will use both the Pearson's r correlation coefficient and the covariance between the multiplier means and the transcribed data for rental price of each neighborhood:

```
#euro/m^2/month -->http://www.gateberlin.de/immobilienmarkt-trends

rental_price = pd.DataFrame(index = range(1), columns = name['loc']).drop(
    columns = ['_London'])

rental_price.iloc[0,:] = np.array([7.77, 11.00, 11.00, 8.05, 10.70, 8.78,
    9.47, 8.91, 8.92])
# rental_price.to_csv('/Users/ash/Downloads/rental_price_berlin.csv')

x_1, x_2 = np.array(rental_price.iloc[0,:], dtype=float), np.array(loc_mean
    , dtype=float)
X = np.stack((x_1, x_2), axis=0)

corr = np.corrcoef(X)
cov = np.cov(X)
```

```
13
14 print("Pearson r:", corr[0,1])
15 print("Covariance:")
16 print('')
17 print(cov)
18
19 ---->Output:
20
21 Pearson r: 0.10897407561687115
22 Covariance:
23
24 [[1.51865    0.01886825]
25  [0.01886825 0.01974055]]
```

From Pearson's r we can see there is a slight positive correlation between the location multipliers and the rental price, reflecting a slight tendency for linear correlation between these variables. The covariance is also positive, indicating that even if the correlation is non-linear, we also observe a positive correlation between these two variables. To conclude, the multipliers for location positively correlates with the rental price of each location, reflecting the trend that if a neighborhood has higher rental price (higher cost of living), the price of the product will also tend to be higher. We can also summarize the data in more intuitive way in Figure 8.

```
1 def present_data(results, name, what):
2     if what == 'store':
3         what_ = 'mul_S'
4     elif what == 'product':
5         what_ = 'base'
6     elif what == 'brand':
7         what_ = 'mul_B'
8     else:
9         what_ = 'mul_L'
10    index = ['mean', 'variance', '2.5%', '50.0%', '97.5%']
11    columns = name[what]
12    table = pd.DataFrame(index=index, columns=columns)
13    data_ = results[what_]
14    table.iloc[0,:] = np.mean(data_, axis=0)
15    table.iloc[1,:] = np.var(data_, axis=0)
16    table.iloc[2,:] = np.percentile(data_, 2.5, axis=0)
17    table.iloc[3,:] = np.percentile(data_, 50.0, axis=0)
18    table.iloc[4,:] = np.percentile(data_, 97.5, axis=0)
19    return table.T
20
21 p = present_data(results, name, 'product')
22 s = present_data(results, name, 'store')
23 b = present_data(results, name, 'brand')
24 loc_data = present_data(results, name, 'loc')
25 loc_mean = loc_data['mean'].iloc[0:-1]
26
27 pd.concat((p,s,b,loc_data), axis=0)
28
29 ---->Output:
```

```

Figure 8
```

4 Extending The Model

We can modify the model to sample for each brand of each product separately, would make more sense if we are assuming for example the store multiplier for apples would be different from the store multiplier for milk. This way, we can sample for each separated brand (no longer only brand/no brand) and get a posterior for each brand of each product, and we also sample the variance for the likelihood function from a gamma distribution:

```
model_spec = """

data {
//Number of store
    int<lower=1> S;
//Number of product brand
    int<lower=1> B;
//Number of location
    int<lower=1> L;
//Number of datapoint
    int<lower=1> N;

//A vector of 0/1 in each entry expressing which store, brand, or location the datapoint is
    vector<lower=0>[S] store[N];
    vector<lower=0>[B] brand[N];
    vector<lower=0>[L] loc[N];

//Price of the datapoint
    real<lower=0> price[N];
}

parameters {

//Variance for the normal distribution centered around the multiplied price
    real<lower=0> v;
//Base price for the
    real<lower=0> base;
    row_vector<lower=0>[S] mul_S;
    row_vector<lower=0>[B] mul_B;
    row_vector<lower=0>[L] mul_L;
}

model {

    base ~ cauchy(0,1);
    v ~ gamma(1,1);
    for (s in 1:S){
        mul_S[s] ~ lognormal(0,0.25);
```

```
    }

    for (b in 1:B){
        mul_B[b] ~ lognormal(0,0.25);
    }

    for (l in 1:L){
        mul_L[l] ~ lognormal(0,0.25);
    }

    for(i in 1:N) {
        price[i] ~ normal(base*((mul_S*store[i])*(mul_B*brand[i])*(mul_L*
    loc[i])), v);
    }
}

"""
model_1 = pystan.StanModel(model_code=model_spec)
```

As we suspected, if we sample for each of the product, the multipliers for the stores and locations are pretty different (full code and results in the appendix). Let just talk about the brand multipliers for apples and bananas in Figure 9. We can see the multiplier for "no brand" for each product is different, and even if some of the brand is shared the multiplier posteriors are different between different products, as we would expect from prices of different grocery item.

5 Stretch Goal

To create spatially correlated priors, we first have to calculate the distances between the neighborhoods. There are many ways to do this, such as using walking time as distance, absolute distance (straight line connecting two points, bird distance), etc. We will establish an Euclidean coordinate system centered at Kreuzberg (Kreuzberg at (0,0)) to calculate the absolute distance between the neighborhoods (Figure 10). After that, we will calculate a spatially correlated covariance matrix between each neighborhood using a radial basis function (exponentiate the negative squared of Euclidean distances) for each entries (Figure 11):

```
from sklearn.metrics.pairwise import rbf_kernel

distances = pd.read_csv('/Users/ash/Downloads/distance.csv')
x = np.array(distances['x '], dtype=float)
y = np.array(distances['y'], dtype=float)


distances = np.zeros((x.shape[0],2))
distances[:,0] = x
distances[:,1] = y
K = rbf_kernel(distances, gamma=0.1)

```

```
13 ---->Output:
14
15 Figure 11
```

Next, we will build the model with the same specification as above, only instead of sampling the location multipliers from independent log-normals, we are sampling them at the same time using only one multivariate Gaussians with a mean vector of 1 and a covariance matrix given in Figure 11. This way, the sampled multipliers for locations will correlated with the distances between them, as specified by the matrix.

```
1  model_spec_3 = """
2
3  data {
4      int<lower=1> S;
5      int<lower=1> B;
6      int<lower=1> L;
7      int<lower=1> N;
8
9      vector<lower=0>[S] store[N];
10     vector<lower=0>[B] brand[N];
11     vector<lower=0>[L] loc[N];
12
13     real<lower=0> price[N];
14
15     vector[L] multi_m;
16     cov_matrix[L] multi_var;
17 }
18
19 parameters {
20
21     real<lower=0> v;
22     real<lower=0> base;
23     row_vector<lower=0>[S] mul_S;
24     row_vector<lower=0>[B] mul_B;
25     row_vector<lower=0>[L] mul_L;
26 }
27
28 model {
29
30     base ~ cauchy(0,1);
31     for (s in 1:S){
32         mul_S[s] ~ lognormal(0,0.25);
33     }
34
35     for (b in 1:B){
36         mul_B[b] ~ lognormal(0,0.25);
37     }
38
39     mul_L ~ multi_normal(multi_m, multi_var);
40
41     for(i in 1:N) {
42         price[i] ~ normal(base*((mul_S*store[i])*(mul_B*brand[i])*(mul_L*
       loc[i])), v);
43     }
```

```
}

"""
model_sg = pystan.StanModel(model_code=model_spec_3)
```

Full summaries for the product apple is presented in the appendix. In figure 12, we can see that the location multipliers are now correlated with each other, and by figure 12b we can also see that now the location multipliers has the most effect on the price variance. Interestingly, the overall range of the location multiplier is also higher (and the ranges of other multipliers are lower to compensate) indicating that the models "blame" a a large part of the price variance, especially when the price increase, to the location. The base price of apples also decrease in this model, also to compensate for the surge of the location multipliers. Another interesting point is that for this model the sampled posterior ranges for the location multipliers are highly dependent on the covariance matrix, which in turn depends on the specification of the RBF kernel. If the entries of the covariance is small, the range will be small and vice versa.

Appendix

CSV Files

Here are links to CSV files used for analysis in this assignment:

- Cleaned Data: https://drive.google.com/open?id=1ajYxBBTTOelIAR68pCgWp7kWdx-ecRDy

- Transcribed Rental Price: https://drive.google.com/open?id=1H6LDNVOMTv9rkN_C2dWa6iWvEnoVOczH

- Calculated Distances Between Neighborhoods, Centered At Kreuzberg (Kreuzberg at (0,0)): https://drive.google.com/open?id=1NsH9N_7v-3m7IHnUqWk4LzQDgrr9mDmP

Metadata

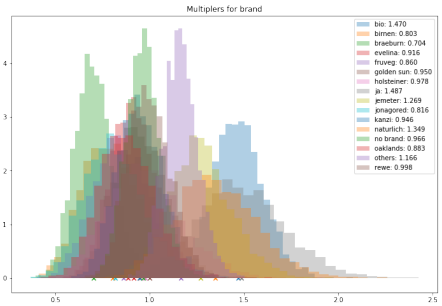
Both stores were visited on Tuesday, October 30th, 2018 from 1:30PM to 5:00PM CEST. Here's the link to the data collection images of this assignment: https://drive.google.com/open?id=1CjyOpsBBeNYwjAeOs-soks1le6AP2Z_l

Code

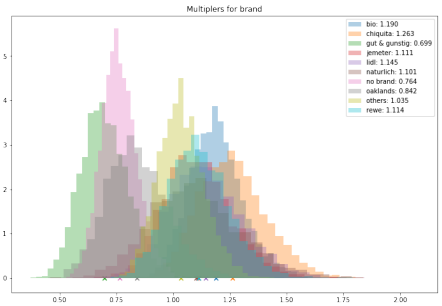
Here's the link to the full code of this assignment: https://gist.github.com/AshNguyen/43e8d474626f7f04f1e3ce60e67e15bb

	mean	variance	2.5%	50.0%	97.5%
apple	1.65154	0.110943	1.10327	1.61986	2.43779
banana	1.10337	0.0518559	0.730972	1.08334	1.63832
butter	2.86778	0.33292	1.92925	2.81424	4.22805
chicken	7.59725	2.32131	5.11794	7.46636	11.2062
egg	1.94323	0.15412	1.30097	1.90787	2.85611
flour	0.780936	0.0280016	0.51444	0.761104	1.16539
milk	0.716483	0.0232486	0.469851	0.701078	1.06366
potato	0.957503	0.0399434	0.633959	0.935612	1.43724
rice	1.82792	0.135399	1.23113	1.79342	2.69203
tomato	2.67771	0.288736	1.81432	2.62624	3.93792
ALDI	0.997854	0.0150515	0.782517	0.986298	1.2596
EDEKA	1.26599	0.0241023	0.991883	1.25201	1.59225
Lidl	0.802462	0.00981156	0.630239	0.793662	1.01187
REWE	1.26905	0.0241463	0.996866	1.25538	1.59236
brand	1.25075	0.0408668	0.888459	1.23895	1.6809
no brand	1.00996	0.026612	0.71692	1.00055	1.36196
Alt-Treptow	1.31878	0.0134318	1.1	1.3147	1.56195
Friedrichshain	1.21698	0.00907869	1.03556	1.21441	1.41772
Kreuzberg	1.01389	0.00648581	0.860682	1.01119	1.18293
Lichtenberg	0.874149	0.00558655	0.735817	0.870564	1.02959
Mitte	1.07123	0.007028	0.910049	1.06841	1.24192
Neukölln	1.03411	0.0067228	0.879411	1.03096	1.20323
Prenzlauer Berg	1.05573	0.00702217	0.89984	1.05197	1.2287
Schöneberg	0.970154	0.00620539	0.820059	0.968381	1.13246
Tempelhof	0.91399	0.00756767	0.749738	0.909771	1.08679
_London	0.849653	0.00585347	0.706046	0.847345	1.00653

Figure 8: Summary Of Posterior Samples For Main Model

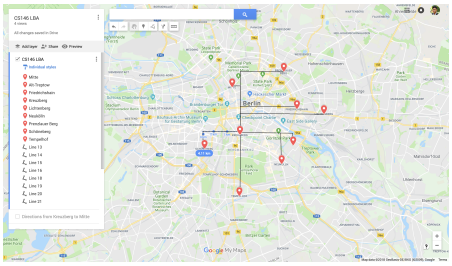


(a) Apple



(b) Banana

Figure 9: Posterior Histograms For Brand Multipliers, Extended Model



(a) 2D Space Centered At Kreuzberg

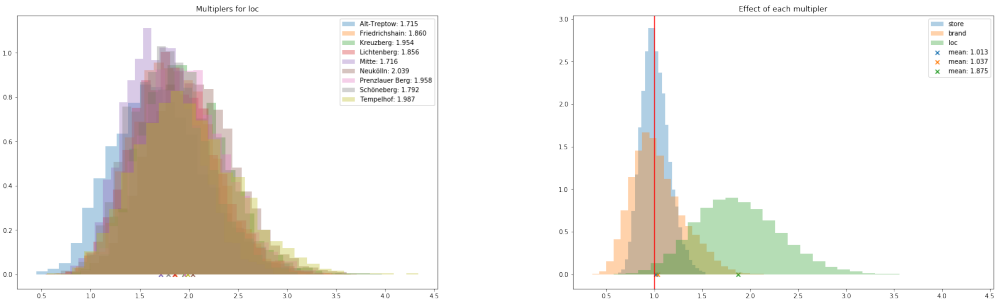
	x	y
Alt-Treptow	3.94	-0.91
Friedrichshain	4.21	1.92
Kreuzberg	0.00	0.00
Lichtenberg	6.24	1.36
Mitte	-0.69	3.45
Neukölln	3.02	-2.01
Prenzlauer Berg	2.96	4.57
Schöneberg	-2.45	-1.66
Tempelhof	0.00	-4.47

(b) Trascribed Coordinates By Using Google Distance Measures

Figure 10: Modeling Spatial Correlation

	Alt-Treptow	Friedrichshain	Kreuzberg	Lichtenberg	Mitte	Neukölln	Prenzlauer Berg	Schöneberg	Tempelhof
Alt-Treptow	1.000000	0.445668	0.194919	0.351942	0.017516	0.814126	0.045091	0.015931	0.059623
Friedrichshain	0.445668	1.000000	0.117531	0.641818	0.071713	0.185241	0.423797	0.003289	0.002864
Kreuzberg	0.194919	0.117531	1.000000	0.016929	0.290004	0.268193	0.051579	0.416524	0.135594
Lichtenberg	0.351942	0.641818	0.016929	1.000000	0.005304	0.113889	0.121693	0.000211	0.000681
Mitte	0.017516	0.071713	0.290004	0.005304	1.000000	0.012809	0.232773	0.053881	0.001799
Neukölln	0.814126	0.185241	0.268193	0.113889	0.012809	1.000000	0.013168	0.049571	0.219325
Prenzlauer Berg	0.045091	0.423797	0.051579	0.121693	0.232773	0.013168	1.000000	0.001105	0.000118
Schöneberg	0.015931	0.003289	0.416524	0.000211	0.053881	0.049571	0.001105	1.000000	0.249110
Tempelhof	0.059623	0.002864	0.135594	0.000681	0.001799	0.219325	0.000118	0.249110	1.000000

Figure 11: Covariance Matrix Of Spatially Correlated Priors



(a) Location Multiplier Posteriors

(b) Effect Of Multipliers On Price

Figure 12: Apples Summaries Of Spatially Correlated Model