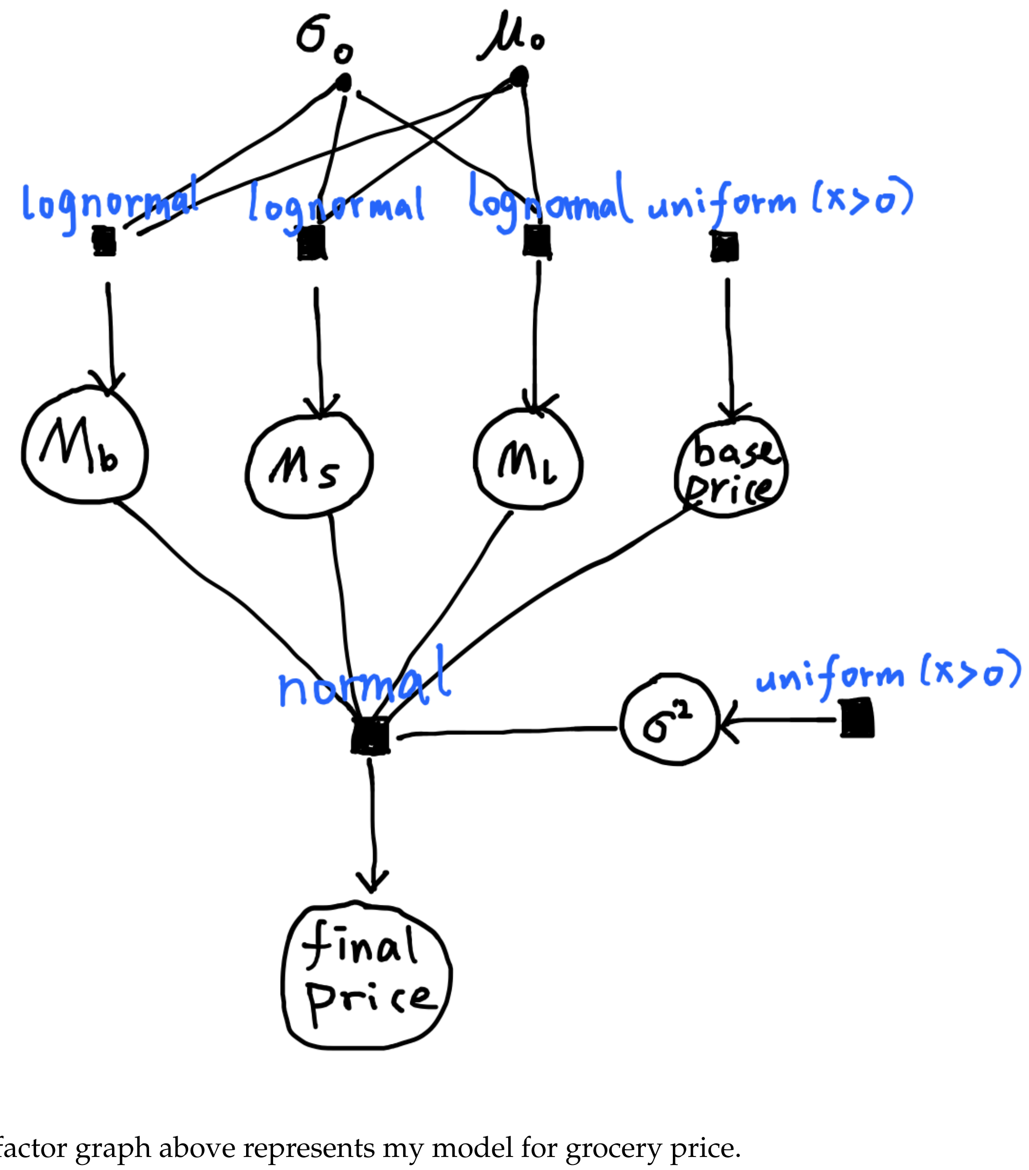
LBA

November 11, 2018

1 Model choice



The factor graph above represents my model for grocery price.

- The final price for each grocery item comes from a normal distribution with a mean of $M_bM_sM_lb$ where M_b is the multiplier for product brand, M_s is the multiplier for store brand,

M_l is the multiplier for location/neighborhood and b is the base price of the product. The normal distribution has a variance of σ^2, which is also a parameter of the model that will be inferred and will be discussed below.

- M_b, M_s and M_l all come from a lognormal prior distribution. The justification behind using lognormal as prior for multipliers goes as follows: We want the logarithm of the multipliers to be (a) symmetrically distributed (b) centered around zero (c) more likely to be close to zero than further away. These conditions naturally give us the normal distribution for the logarithms. Intuitively, our prior assumption is that a multiplier is as likely to make an item twice as expensive as it is to make it half as expensive (thereby the symmetry). The hyperparameters for the lognormal prior are $\mu_0 = 0$ and $\sigma_0 = \frac{\ln(3)}{3}$, as we assume that 99.7% of the multipliers are within $\frac{1}{3}$ and 3.

- For the base price, we use an uninformative uniform prior because as the assignment instruction said, we don't know the scale of the prices. The only restriction is that it's a positive number. Similarly, for the random fluctuations, since we don't know the scale, we just use a uniform prior over positive real numbers. In Stan, if we don't specify a distribution for some parameter, the default is a uniform prior. This is why in my Stan code there's no specification for the distributions for the base price and σ^2.

2 Preprocessing data

```
In [1]: import pandas as pd
        import numpy as np
        # Loading neighborhood data
        berlin_nbh_data = pd.read_csv("berlin_supermarkets.csv", header=3)
        london_nbh_data = pd.read_csv("london_supermarkets.csv", header=3)
        nbh_data = pd.concat([berlin_nbh_data, london_nbh_data])
        # Dropping rows that weren't assigned to any student
        nbh_data = nbh_data.dropna(subset = ['Student name'])
```

```
In [4]: # Loading grocery data collected by students
        data = pd.read_csv("data.csv")
        # Dropping Jennifer's duplicate
        data = data.drop([0])
        # Dropping Cape Town
        data = data[data["Your name"] != "Josh Broomberg"]
```

```
In [8]: from difflib import SequenceMatcher

        # Trying to match rows from data.csv to supermarkets.csv
        # to obtain neighborhood information
        data['ID'] = data["Your name"]+data["Grocery store brand"]
                     +data["Grocery store street address"]
        nbh_data['ID'] = nbh_data['Student name'] + nbh_data['Supermarket']
        neighborhood_column = []
        for id1 in data["ID"]:
            max_score = 0
            neighborhood = ""
            for _, nbh_row in nbh_data.iterrows():
                id2 = nbh_row["ID"]
                # Compare similarity between two strings
                score = SequenceMatcher(None, id1, id2).ratio()
                if score>max_score:
                    neighborhood = nbh_row["Neighborhood"]
                    add2 = id2
                    max_score = score
            neighborhood_column.append(neighborhood)
        # Adding neighborhood column to `data` dataframe
        data['Neighborhood'] = neighborhood_column
```

```
In [9]: # Creating a new dataframe where each row contains an item
        item_data = pd.DataFrame(columns=['price','brand_raw','store','location','product'])
        product_list = ['apple', 'banana', 'tomato', 'potato','flour','rice','milk','butter'
                        ,'egg','chicken']
        item_index = 0
        for _, row in data.iterrows():
            start_index = 4
            for product in product_list:#for each product
                for i in range(3):#for each brand
                    brand = row[start_index]
                    price = row[start_index+1]
                    store = row['Grocery store brand']
                    location = row['Neighborhood']
                    item_data.loc[item_index] = [price, brand, store, location, product]
                    start_index += 2
                    item_index += 1
```

```
In [10]: # Dropping empty entries
         item_data = item_data.dropna(subset = ['price'])
         # Convert all brands to lowercase
         item_data.loc[:,'brand_raw'] = item_data['brand_raw'].str.lower()
```

```
In [11]: # Importing csv which encodes manually cleaned brand names
         brand_dict_csv = pd.read_csv("brand_clean_dict.csv")
         brand_dict = {}
         for _, row in brand_dict_csv.iterrows():
             brand_dict[row["original"]] = row["convert"]
         item_data["brand"] = item_data["brand_raw"].map(brand_dict)
```

```
In [12]: # Getting rid of NaN and trailing space
         item_data["brand"] = item_data["brand"].map(
                                 lambda x: 'no brand' if pd.isnull(x) else x)
         item_data["brand"] = item_data["brand"].map(lambda x: x.strip())
```

```
In [13]: print "{} out of {} brands only occur once".format(
             sum(item_data["brand"].value_counts()<2), len(item_data["brand"].unique()))
```

```
115 out of 261 brands only occur once
```

```
In [14]: # Removing brands that only occur once
         discarded_brands = set(item_data["brand"].value_counts()
                                .index[item_data["brand"].value_counts()<2])
         item_data = item_data[~item_data["brand"].isin(discarded_brands)]

In [15]: print len(item_data)

1797


In [16]: # Encoding each brand, store brand, location and product
         unique_brands = item_data["brand"].unique()
         brand_code = dict(zip(unique_brands, range(1, len(unique_brands)+1)))
         brand_code_rev = dict(zip(range(1, len(unique_brands)+1), unique_brands))

         unique_stores = item_data["store"].unique()
         store_code = dict(zip(unique_stores, range(1, len(unique_stores)+1)))
         store_code_rev = dict(zip(range(1, len(unique_stores)+1), unique_stores))

         unique_locs = item_data["location"].unique()
         loc_code = dict(zip(unique_locs, range(1, len(unique_locs)+1)))
         loc_code_rev = dict(zip(range(1, len(unique_locs)+1), unique_locs))

         unique_prods = item_data["product"].unique()
         prod_code = dict(zip(unique_prods, range(1, len(unique_prods)+1)))
         prod_code_rev = dict(zip(range(1, len(unique_prods)+1), unique_prods))

In [17]: # Reformatting data to be used as input for Stan model
         n_item = len(item_data)
         item_info = np.zeros((n_item, 4)).astype(int)
         item_price = np.array(item_data["price"])
         item_info[:,0] = item_data["brand"].map(brand_code)
         item_info[:,1] = item_data["store"].map(store_code)
         item_info[:,2] = item_data["location"].map(loc_code)
         item_info[:,3] = item_data["product"].map(prod_code)
```

3 Building Stan model

```
In [18]: import pystan

In [24]: stan_input = {
             'n_item': n_item,
             'n_brand': len(unique_brands),
             'n_store': len(unique_stores),
             'n_loc': len(unique_locs),
             'n_prod': len(unique_prods),
             'item_price': item_price,
             'item_info': item_info,
    'sigma_0': np.log(3)/3,
    'mu_0': 0
}
```

```
In [20]: # Specifying Stan model
         stan_code = """
         data {
             int<lower=1> n_item;  // number of items in data collected
             int<lower=1> n_brand;  // number of brands
             int<lower=1> n_store; // number of stores
             int<lower=1> n_loc; // number of locations
             int<lower=1> n_prod; // number of products

             real<lower=0> item_price[n_item]; // array for price of each item
             int<lower=0> item_info[n_item, 4]; // 2D array for info of each item

             real<lower=0> sigma_0; // hyperparameter for lognormal priors
             real mu_0; // hyperparameter for lognormal priors
         }

         parameters {
             real<lower=0> brand_mult[n_brand]; // brand multipliers
             real<lower=0> store_mult[n_store]; // store brand multipliers
             real<lower=0> loc_mult[n_loc]; // location multipliers
             real<lower=0> base_price[n_prod]; // product base price
             real<lower=0> sigma2; // random fluctuation around the price
         }

         model {
             brand_mult ~ lognormal(mu_0, sigma_0); // prior for brand_mult
             store_mult ~ lognormal(mu_0, sigma_0); // prior for store_mult
             loc_mult ~ lognormal(mu_0, sigma_0); // prior for loc_mult
             for (i in 1:n_item) {
                 item_price[i] ~ normal(brand_mult[item_info[i,1]]
                     * store_mult[item_info[i,2]]
                     * loc_mult[item_info[i,3]]
                     * base_price[item_info[i,4]], sigma2);
             }
         }
         """
```

```
In [21]: # Compiling Stan model
         stan_model = pystan.StanModel(model_code=stan_code)
```

```
INFO:pystan:COMPILING THE C++ CODE FOR MODEL anon_model_4be7b0968f0f0d8278e7c41a16c84e60 NOW.
```

```
In [25]: # Sampling from posterior
         stan_output = stan_model.sampling(data=stan_input)
```

4 Inference results

```
In [69]: import matplotlib.pyplot as plt
         import seaborn as sns
         plt.rcParams['figure.dpi'] = 200
         plt.rcParams['figure.figsize'] = (9,6)
         sns.set_context("paper")
         sns.set_style("whitegrid")
```

```
In [91]: # Getting a list of the 30 most popular brands
         selected_brands = item_data["brand"].value_counts()[:30].index
         selected_brands_index = np.array([brand_code[x] for x in selected_brands])
```

```
In [92]: # Extract samples from Stan and convert to dataframes to be used for plotting
         samples = stan_output.extract()

         brand_mult_samples = pd.DataFrame.from_dict({
             "multiplier":samples['brand_mult'].transpose(
                 )[selected_brands_index-1, :].ravel(),
             "brand":np.tile(unique_brands, (4000,1)).transpose(
                 )[selected_brands_index-1, :].ravel()
         })

         store_mult_samples = pd.DataFrame.from_dict({
             "multiplier":samples['store_mult'].transpose().ravel(),
             "store":np.tile(unique_stores, (4000,1)).transpose().ravel()
         })

         loc_mult_samples = pd.DataFrame.from_dict({
             "multiplier":samples['loc_mult'].transpose().ravel(),
             "location":np.tile(unique_locs, (4000,1)).transpose().ravel()
         })

         base_price_samples = pd.DataFrame.from_dict({
             "base price":samples['base_price'].transpose().ravel(),
             "product":np.tile(unique_prods, (4000,1)).transpose().ravel()
         })
```

4.1 Product base price

Below is a violin plot reflecting the base prices for each product. The white dot represents the medium and the black rectangle represents interquartile range. The violin reflects the distribution of the data over 95% confidence interval.

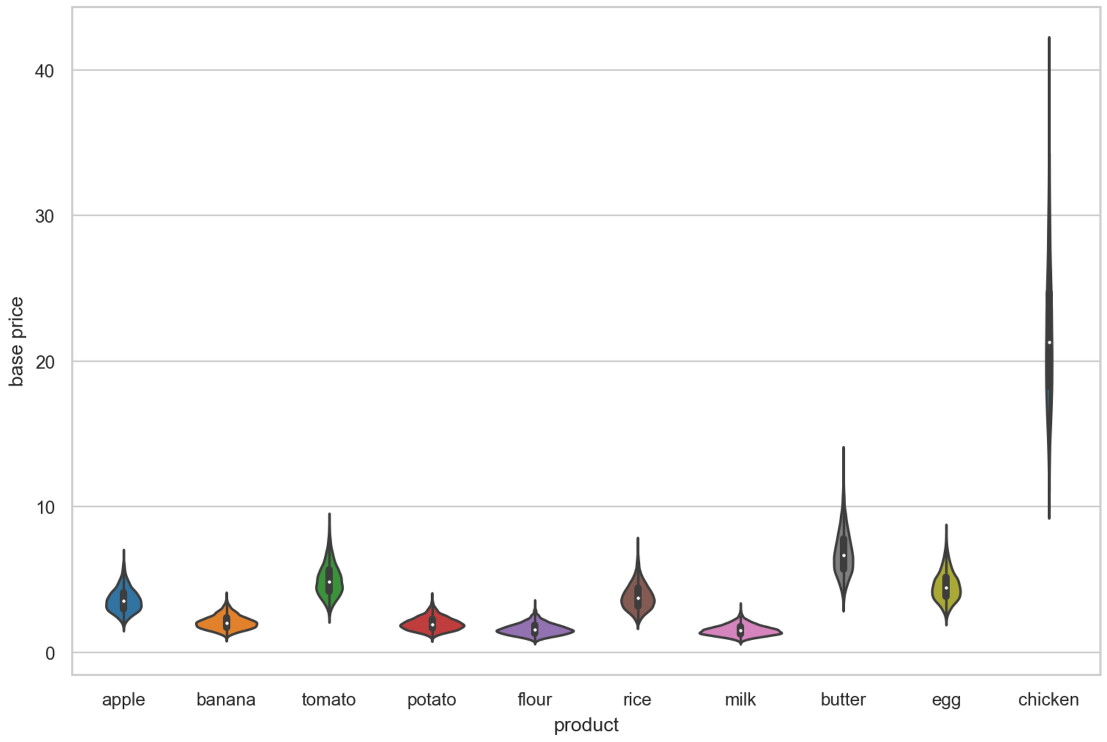
This result seems reasonable. Chicken is inferred to be the most pricy item.

```
In [116]: base_price_samples.groupby(['product'])['base price'].mean()

Out[116]: product
          apple       3.589381
```

```
banana      2.075971
butter      6.835867
chicken    21.779932
egg         4.558841
flour       1.602038
milk        1.527940
potato      1.987764
rice        3.826609
tomato      4.976856
Name: base price, dtype: float64
```

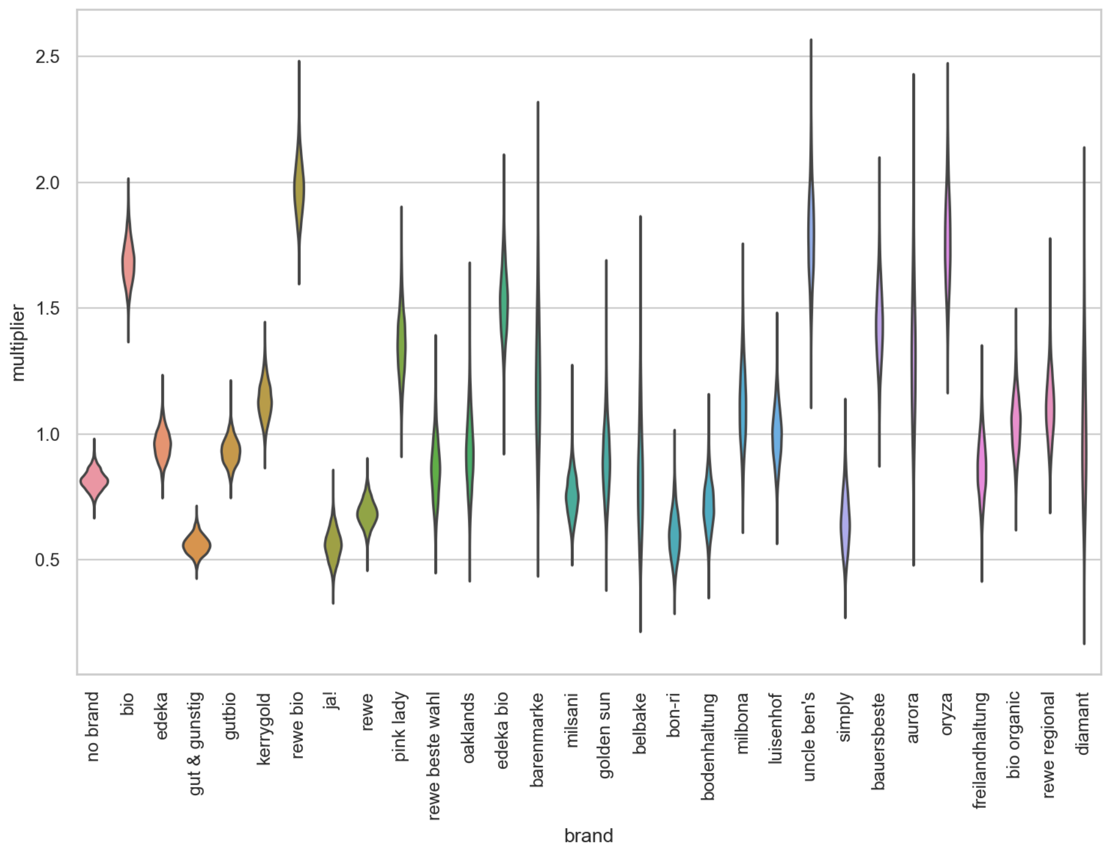
```
In [70]: ax = sns.violinplot(x="product", y="base price", data=base_price_samples)
```



4.2 Product brand multiplier

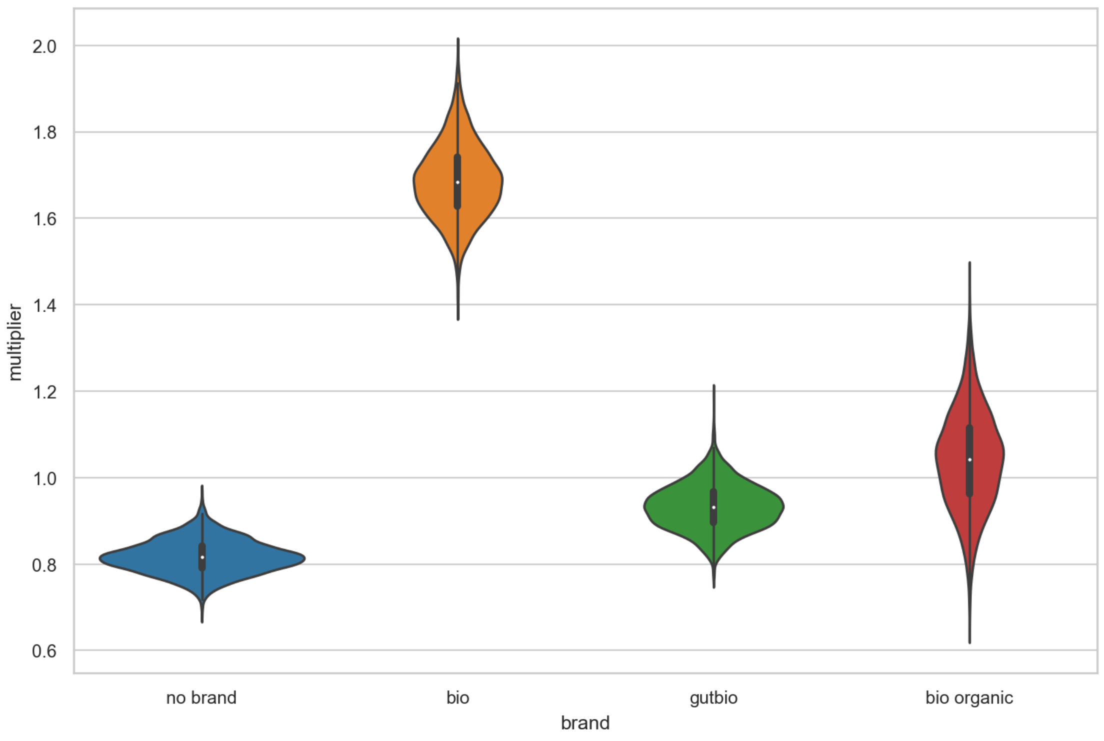
Since there are too many brands to display together, I picked the 30 most common brands and plotted them below.

```
In [106]: ax = sns.violinplot(x="brand", y="multiplier", data=brand_mult_samples, inner=None)
          plt.xticks(rotation=90)
```



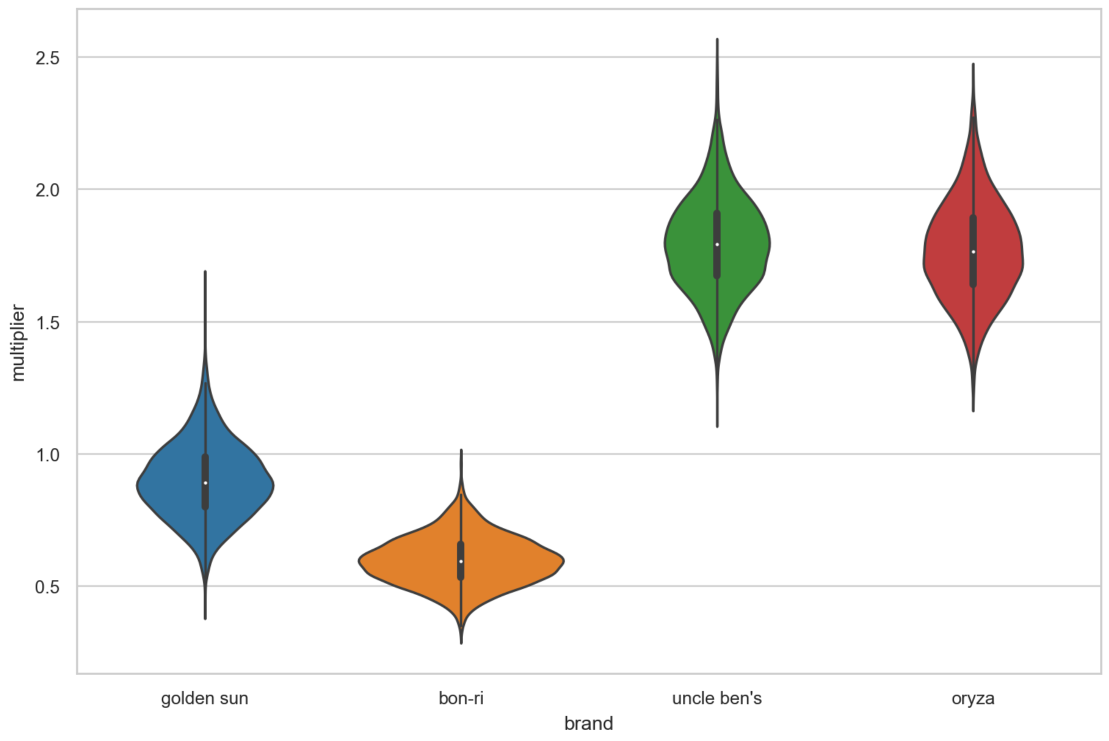
Specifically, comparing "no brand" with different versions of bio (organic) products, we see that bio products are a lot more expensive.

```
In [113]: ax = sns.violinplot(x="brand", y="multiplier",
                  data=brand_mult_samples[brand_mult_samples["brand"].isin(
                      ["no brand", "bio", "gutbio", "bio organic"])])
```



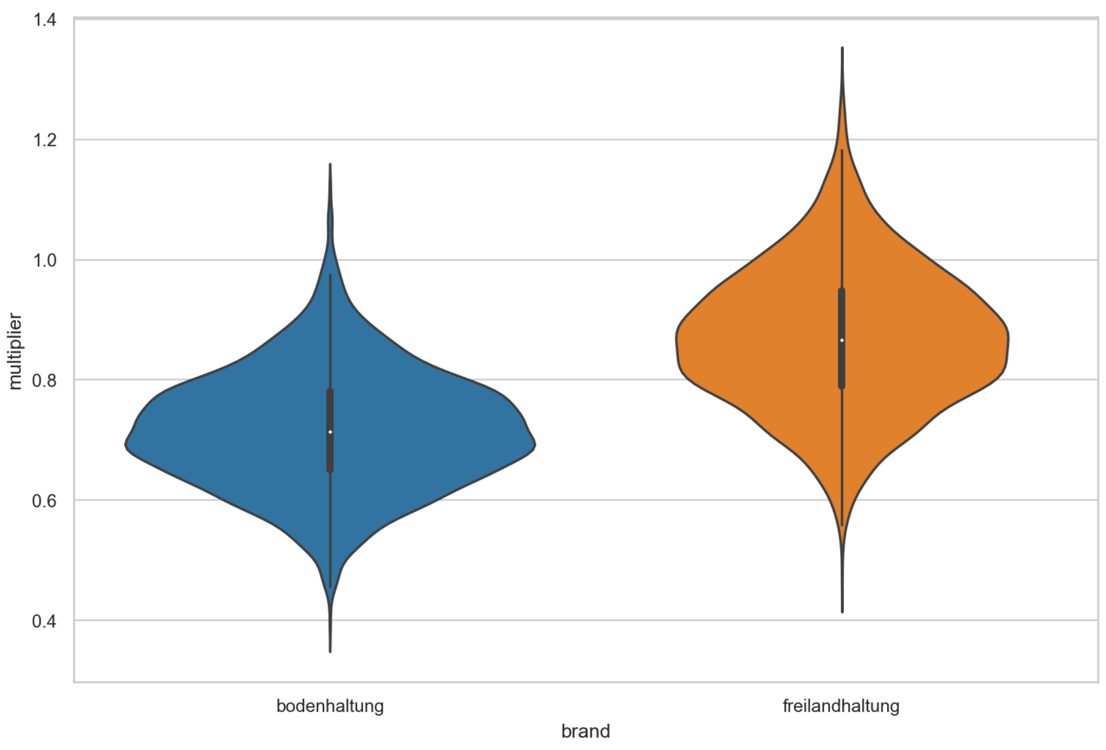
Comparing the four rice brands, we can see that bon-ri is the cheapest:

```
In [112]: ax = sns.violinplot(x="brand", y="multiplier",
                  data=brand_mult_samples[brand_mult_samples["brand"].isin(
                      ["golden sun", "uncle ben's", "oryza", "bon-ri"])])
```



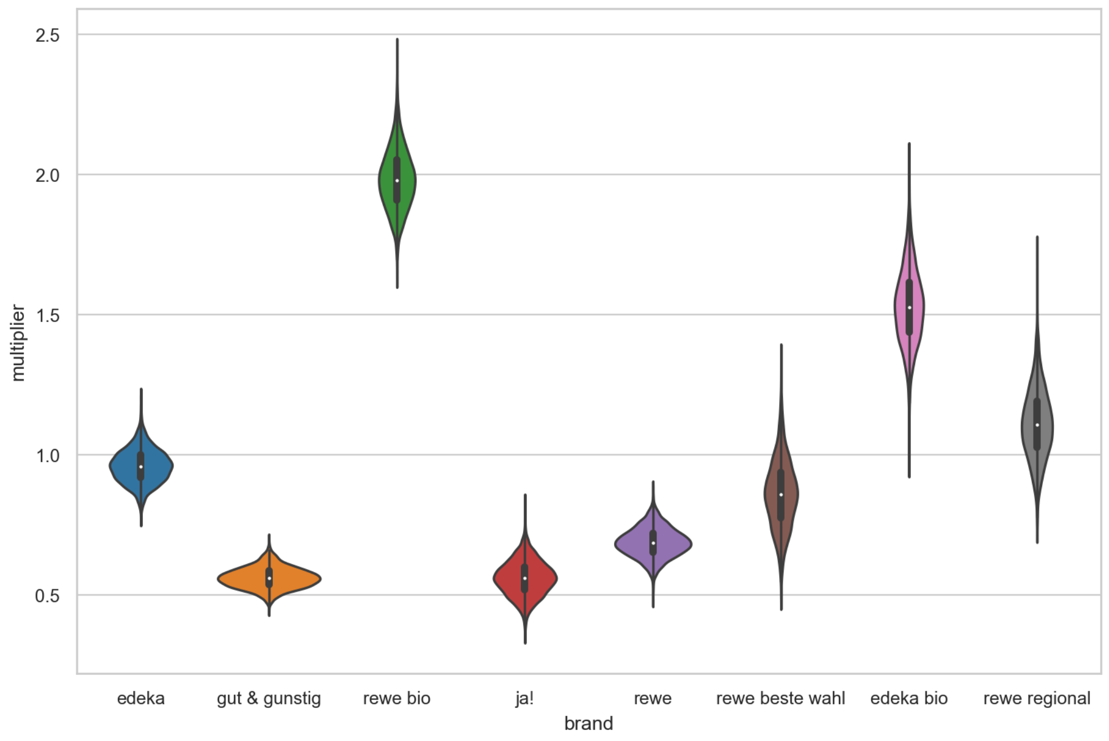
Comparing Bodenhaltung (barn) eggs and Freilandhaltung (free range) eggs, we can see that Freilandhaltung has higher price.

```
In [111]: ax = sns.violinplot(x="brand", y="multiplier",
                 data=brand_mult_samples[brand_mult_samples["brand"].isin(
                     ["bodenhaltung", "freilandhaltung"])])
```



Finally, these are the common labels under store brands Edeka and Rewe. "Gut & günstig" is Edeka's label for products at the lowest price level and "ja!" is the corresponding label for Rewe. This is well reflected in the plot below. We can see that for Rewe, "ja!" < "rewe" < "rewe beste wahl"(best value) < "rewe regional" (local products) < "rewe bio" (organic).

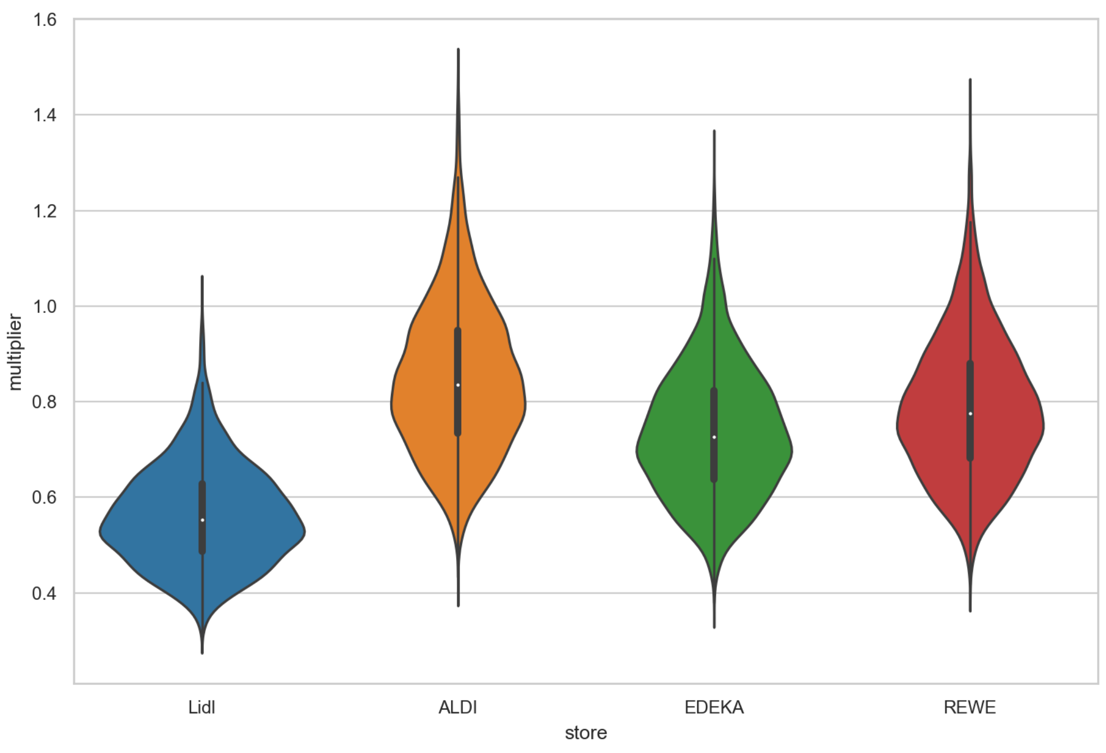
```
In [110]: ax = sns.violinplot(x="brand", y="multiplier",
                  data=brand_mult_samples[brand_mult_samples["brand"].isin(
                      ["edeka", "gut & gunstig", "rewe", "rewe bio", "ja!", "rewe beste wahl",
                      "edeka bio", "rewe regional"])])
```



4.3 Store brand multiplier

The results for store brands seem counterintuitive. Usually Aldi is perceived as the cheapest store and Lidl the second cheapest. On the contrary, in the plot below we see that the model inferred Aldi to be the most expensive store. This result, however, is associated with high uncertainty, as we can see the distributions are very wide-spread.

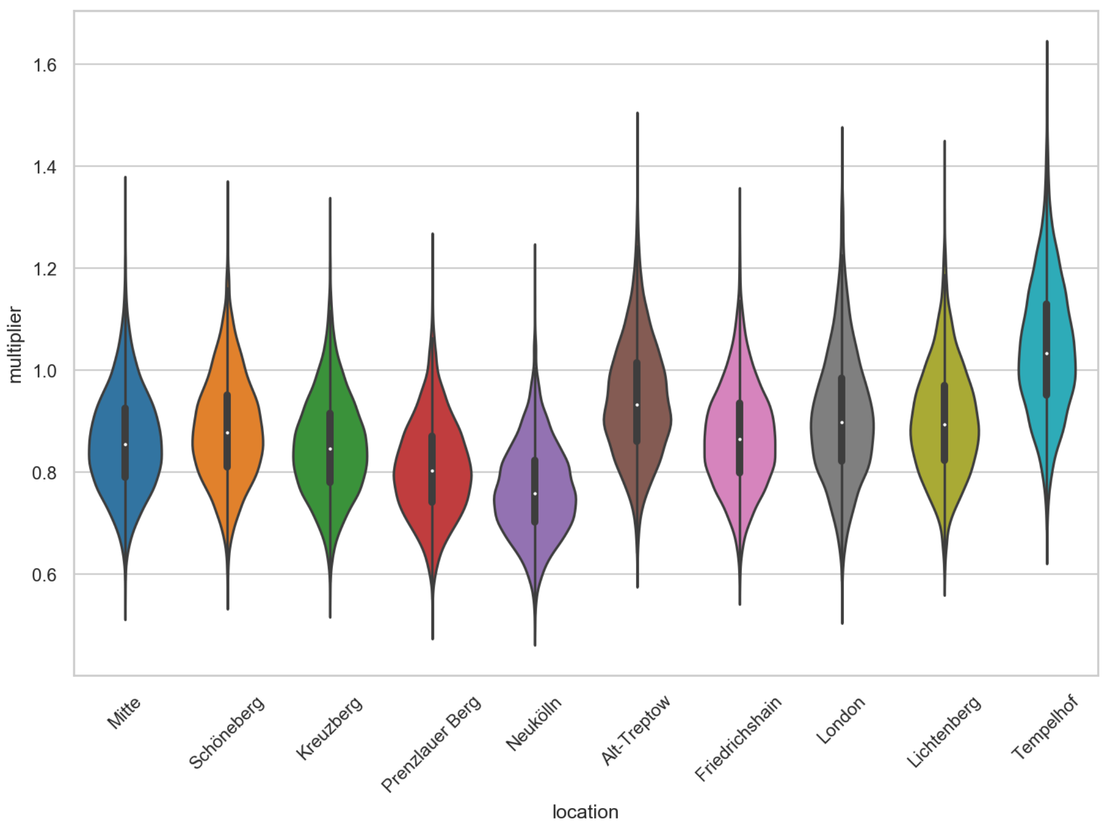
```
In [107]: ax = sns.violinplot(x="store", y="multiplier", data=store_mult_samples)
```



4.4 Location multiplier

The location multipliers are also associated with very high uncertainty as most of the distributions overlap to a large degree. The multipliers for locations don't seem to correlate with the housing price, as Tempelhof and Alt-Treptow are associated with low housing price and Prenzlauer has one of the highest housing price. The hypothesized correlation is not reflected in the inference result, but again since the variance is too high, we cannot conclude that much.

```
In [109]: ax = sns.violinplot(x="location", y="multiplier", data=loc_mult_samples)
          plt.xticks(rotation=45)
```



5 Appendix: model summary

```
In [26]: print stan_output.stansummary()
```

```
Inference for Stan model: anon_model_4be7b0968f0f0d8278e7c41a16c84e60.
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
brand_mult[1]       0.93  2.3e-3   0.18   0.61   0.81   0.93   1.05    1.3 5892.0    1.0
brand_mult[2]       1.11  3.4e-3   0.32   0.57   0.88   1.08   1.31   1.81 8687.0    1.0
brand_mult[3]       0.82  1.6e-3   0.04   0.74   0.79   0.82   0.84    0.9  611.0   1.01
brand_mult[4]       1.69  3.3e-3   0.09   1.53   1.63   1.68   1.74   1.87  686.0    1.0
brand_mult[5]       0.83  2.5e-3   0.22   0.48   0.68   0.81   0.96   1.31 7469.0    1.0
brand_mult[6]       1.07  3.9e-3   0.35   0.51   0.82   1.02   1.28   1.87 8192.0    1.0
brand_mult[7]        0.9  2.0e-3   0.14   0.64    0.8   0.89   0.99    1.2 5086.0    1.0
brand_mult[8]       1.12  2.5e-3   0.16   0.82   1.01   1.11   1.22   1.44 3960.0    1.0
brand_mult[9]       1.32  4.0e-3   0.32   0.75    1.1   1.31   1.52   2.02 6534.0    1.0
brand_mult[10]      1.22  3.1e-3   0.26   0.76   1.04    1.2   1.38   1.76 6926.0    1.0
brand_mult[11]      1.14  2.2e-3   0.07    1.0   1.08   1.13   1.19   1.29 1143.0    1.0
```

```
brand_mult[12]     1.0  2.5e-3   0.15   0.72    0.9    1.0    1.1    1.3 3658.0     1.0
brand_mult[13]    0.72  1.4e-3    0.1   0.53   0.65   0.71   0.78   0.92 4651.0     1.0
brand_mult[14]    0.67  1.4e-3   0.05   0.58   0.64   0.67   0.71   0.78 1375.0     1.0
brand_mult[15]    1.02  2.9e-3   0.25   0.57   0.84    1.0   1.18   1.54 7364.0     1.0
brand_mult[16]    0.69  1.8e-3   0.15   0.42   0.59   0.68   0.78   1.01 6531.0     1.0
brand_mult[17]    0.87  2.6e-3   0.23   0.48   0.71   0.85   1.01   1.38 7550.0     1.0
brand_mult[18]    0.76  1.6e-3   0.08    0.6    0.7   0.75   0.81   0.92 2571.0     1.0
brand_mult[19]    0.93  1.9e-3   0.05   0.83    0.9   0.93   0.97   1.04  805.0     1.0
brand_mult[20]    0.99  2.1e-3    0.1   0.82   0.93   0.99   1.06   1.19 2152.0     1.0
brand_mult[21]    1.36  2.5e-3   0.13   1.12   1.27   1.36   1.44   1.63 2720.0     1.0
brand_mult[22]    0.56  1.2e-3   0.04   0.49   0.53   0.56   0.58   0.64  987.0     1.0
brand_mult[23]    1.01  3.2e-3   0.26   0.56   0.82   0.98   1.18   1.58 6671.0     1.0
brand_mult[24]    1.53  3.1e-3   0.13   1.28   1.44   1.53   1.61   1.81 1815.0     1.0
brand_mult[25]    0.96  2.0e-3   0.06   0.85   0.92   0.96    1.0   1.08  925.0     1.0
brand_mult[26]    1.13  4.1e-3   0.39   0.53   0.84   1.08   1.36   2.01 8999.0     1.0
brand_mult[27]    0.86  2.2e-3   0.17   0.55   0.74   0.85   0.97   1.22 6220.0     1.0
brand_mult[28]    1.44  2.9e-3   0.15   1.15   1.34   1.44   1.54   1.75 2744.0     1.0
brand_mult[29]    1.25  3.0e-3   0.21   0.86    1.1   1.24   1.39   1.68 4982.0     1.0
brand_mult[30]    1.77  3.6e-3   0.18   1.43   1.64   1.76   1.89   2.15 2615.0     1.0
brand_mult[31]    1.82  3.9e-3   0.23   1.38   1.66   1.81   1.97   2.28 3587.0     1.0
brand_mult[32]    1.12  3.9e-3   0.37   0.54   0.85   1.07   1.34   1.94 8844.0     1.0
brand_mult[33]    1.08  2.3e-3   0.12   0.85   0.99   1.07   1.16   1.32 2700.0     1.0
brand_mult[34]    1.69  3.5e-3   0.21   1.28   1.55   1.69   1.83   2.13 3722.0     1.0
brand_mult[35]     1.0  2.1e-3   0.11   0.79   0.92    1.0   1.07   1.22 2627.0     1.0
brand_mult[36]     0.8  1.8e-3   0.07   0.67   0.75   0.79   0.84   0.94 1353.0     1.0
brand_mult[37]    1.97  4.1e-3   0.11   1.77   1.89   1.96   2.03   2.19  677.0     1.0
brand_mult[38]    0.87  2.7e-3   0.25   0.44   0.68   0.84   1.02   1.45 8600.0     1.0
brand_mult[39]     0.6  1.4e-3   0.09   0.43   0.53    0.6   0.66   0.79 4386.0     1.0
brand_mult[40]    0.93  2.6e-3   0.24   0.52   0.75   0.91   1.09   1.46 8698.0     1.0
brand_mult[41]    0.87  2.0e-3   0.12   0.65   0.79   0.87   0.95   1.11 3374.0     1.0
brand_mult[42]    0.45  9.6e-4   0.04   0.38   0.42   0.45   0.47   0.52 1360.0     1.0
brand_mult[43]    1.04  2.3e-3   0.11   0.82   0.96   1.04   1.12   1.27 2468.0     1.0
brand_mult[44]    1.33  4.6e-3   0.39   0.66   1.05    1.3   1.59   2.18 7244.0     1.0
brand_mult[45]    1.25  2.9e-3   0.16   0.94   1.14   1.25   1.36   1.58 3074.0     1.0
brand_mult[46]    0.93  2.6e-3   0.23   0.53   0.77   0.91   1.08   1.45 8303.0     1.0
brand_mult[47]    0.87  2.6e-3   0.23   0.48    0.7   0.85   1.02   1.37 7533.0     1.0
brand_mult[48]    1.06  2.6e-3   0.15   0.77   0.95   1.05   1.16   1.36 3332.0     1.0
brand_mult[49]    1.09  2.5e-3   0.12   0.86   1.01   1.09   1.18   1.34 2428.0     1.0
brand_mult[50]    0.95  2.0e-3   0.06   0.84   0.91   0.95   0.98   1.06  816.0     1.0
brand_mult[51]    0.83  2.8e-3   0.25   0.43   0.65    0.8   0.98   1.36 7604.0     1.0
brand_mult[52]    1.02  3.5e-3   0.32   0.51   0.78   0.98   1.22   1.72 8054.0     1.0
brand_mult[53]    1.04  2.4e-3   0.13    0.8   0.95   1.04   1.12   1.31 2985.0     1.0
brand_mult[54]     0.6  1.4e-3    0.1   0.41   0.53    0.6   0.67   0.81 5276.0     1.0
brand_mult[55]    0.45  9.8e-4   0.04   0.38   0.42   0.45   0.48   0.54 1830.0     1.0
brand_mult[56]    0.86  1.9e-3   0.12   0.63   0.78   0.86   0.94   1.11 3908.0     1.0
brand_mult[57]    0.89  2.4e-3   0.21   0.52   0.74   0.87   1.02   1.33 7364.0     1.0
brand_mult[58]    1.21  3.2e-3   0.25   0.74   1.04    1.2   1.37   1.72 6284.0     1.0
brand_mult[59]    1.98  3.8e-3   0.11   1.79   1.91   1.98   2.05    2.2  762.0     1.0
```

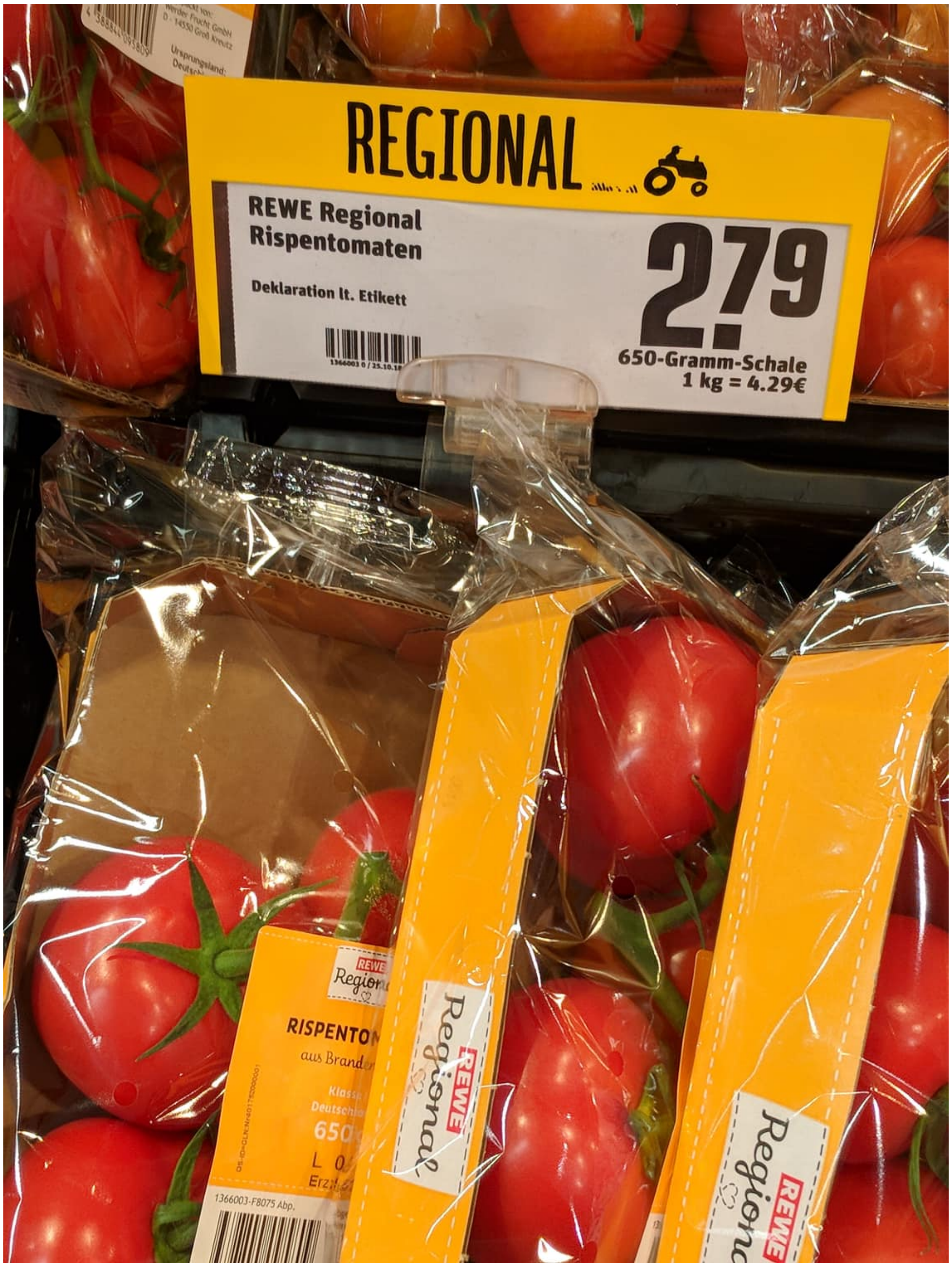
```
brand_mult[60]    0.56  1.2e-3   0.06   0.44   0.52   0.56    0.6   0.68 2791.0     1.0
brand_mult[61]    1.05  4.0e-3   0.34    0.5   0.81   1.02   1.25   1.83 6979.0     1.0
brand_mult[62]     1.8  3.7e-3   0.18   1.46   1.67   1.79   1.91   2.16 2344.0     1.0
brand_mult[63]    0.91  2.1e-3   0.17    0.6    0.8    0.9   1.02   1.24 5925.0     1.0
brand_mult[64]    0.72  1.8e-3   0.14   0.45   0.62   0.71   0.81   1.02 6498.0     1.0
brand_mult[65]    0.97  2.7e-3   0.25   0.54    0.8   0.95   1.13   1.49 8302.0     1.0
brand_mult[66]    1.13  4.2e-3   0.35   0.55   0.88   1.09   1.34   1.93 6948.0     1.0
brand_mult[67]    1.31  2.9e-3   0.26   0.85   1.13    1.3   1.48   1.87 7895.0     1.0
brand_mult[68]    1.25  2.9e-3   0.19   0.91   1.12   1.25   1.38   1.63 4073.0     1.0
brand_mult[69]    1.14  3.7e-3   0.34   0.58    0.9   1.11   1.35    1.9 8439.0     1.0
brand_mult[70]    1.09  3.6e-3    0.3   0.59   0.88   1.07   1.28   1.74 7026.0     1.0
brand_mult[71]    0.69  1.4e-3   0.05   0.59   0.65   0.69   0.72   0.79 1357.0     1.0
brand_mult[72]    0.62  1.3e-3   0.04   0.54   0.59   0.62   0.65   0.71 1074.0     1.0
brand_mult[73]    0.72  1.9e-3   0.15   0.45   0.61   0.71   0.82   1.03 6374.0     1.0
brand_mult[74]    1.22  2.6e-3   0.07   1.09   1.17   1.22   1.27   1.37  797.0     1.0
brand_mult[75]    1.55  3.8e-3   0.27   1.06   1.37   1.54   1.73   2.11 4968.0     1.0
brand_mult[76]    1.41  4.0e-3   0.32   0.84   1.18   1.39   1.61   2.08 6457.0     1.0
brand_mult[77]    0.77  2.4e-3   0.18   0.45   0.65   0.76   0.89   1.17 6064.0     1.0
brand_mult[78]    0.96  3.3e-3   0.28    0.5   0.75   0.92   1.14   1.58 7465.0     1.0
brand_mult[79]    0.85  2.0e-3   0.16   0.56   0.74   0.85   0.95   1.19 6566.0     1.0
brand_mult[80]    1.11  2.3e-3   0.13   0.88   1.02   1.11   1.19   1.37 2841.0     1.0
brand_mult[81]    1.09  2.7e-3   0.18   0.76   0.96   1.09   1.21   1.46 4484.0     1.0
brand_mult[82]    0.91  2.8e-3   0.25   0.49   0.73   0.89   1.07   1.47 8163.0     1.0
brand_mult[83]    0.96  2.6e-3    0.2    0.6   0.82   0.95   1.09    1.4 5830.0     1.0
brand_mult[84]    0.93  2.0e-3   0.14   0.67   0.83   0.93   1.02   1.22 5254.0     1.0
brand_mult[85]    1.06  3.8e-3   0.33   0.53   0.82   1.02   1.26    1.8 7695.0     1.0
brand_mult[86]    0.65  1.5e-3   0.11   0.44   0.57   0.65   0.73   0.89 6045.0     1.0
brand_mult[87]    0.82  2.8e-3   0.23   0.45   0.66    0.8   0.96   1.32 6745.0     1.0
brand_mult[88]    0.79  2.0e-3   0.14   0.53   0.69   0.79   0.88   1.09 5236.0     1.0
brand_mult[89]    0.72  1.7e-3   0.13    0.5   0.64   0.72   0.81   0.98 5172.0     1.0
brand_mult[90]    1.02  2.7e-3   0.18   0.67    0.9   1.01   1.13    1.4 4623.0     1.0
brand_mult[91]    0.77  1.9e-3   0.15   0.49   0.66   0.76   0.86   1.08 6178.0     1.0
brand_mult[92]    0.65  1.3e-3   0.06   0.55   0.61   0.65   0.69   0.77 1987.0     1.0
brand_mult[93]     1.3  3.4e-3   0.31   0.72   1.08   1.28    1.5   1.94 8305.0     1.0
brand_mult[94]    1.18  4.0e-3   0.32   0.62   0.95   1.15   1.38   1.89 6729.0     1.0
brand_mult[95]    1.05  4.4e-3   0.36    0.5    0.8    1.0   1.26   1.91 6787.0     1.0
brand_mult[96]    2.29  4.9e-3   0.28   1.76   2.09   2.28   2.47   2.84 3205.0     1.0
brand_mult[97]    1.16  4.5e-3   0.39   0.54   0.88   1.11   1.39   2.08 7494.0     1.0
brand_mult[98]    1.18  4.1e-3   0.38   0.58   0.91   1.13   1.39   2.03 8213.0     1.0
brand_mult[99]    1.01  3.0e-3   0.25   0.57   0.84   0.99   1.17   1.55 7010.0     1.0
brand_mult[100]   1.47  3.8e-3    0.3    0.9   1.26   1.46   1.67   2.08 6405.0     1.0
brand_mult[101]   1.13  2.8e-3   0.22   0.73   0.97   1.12   1.27   1.58 6148.0     1.0
brand_mult[102]   1.08  3.2e-3   0.29   0.58   0.88   1.05   1.26   1.73 7955.0     1.0
brand_mult[103]    0.4  9.2e-4   0.04   0.32   0.37    0.4   0.43   0.49 2262.0     1.0
brand_mult[104]   1.06  3.7e-3   0.34   0.53   0.82   1.02   1.26   1.82 8458.0     1.0
brand_mult[105]   1.15  2.9e-3   0.24   0.72   0.99   1.14    1.3   1.65 6487.0     1.0
brand_mult[106]   1.09  2.6e-3   0.19   0.74   0.95   1.08   1.21   1.48 5206.0     1.0
brand_mult[107]   1.48  4.2e-3   0.32    0.9   1.25   1.46   1.69   2.16 5901.0     1.0
```

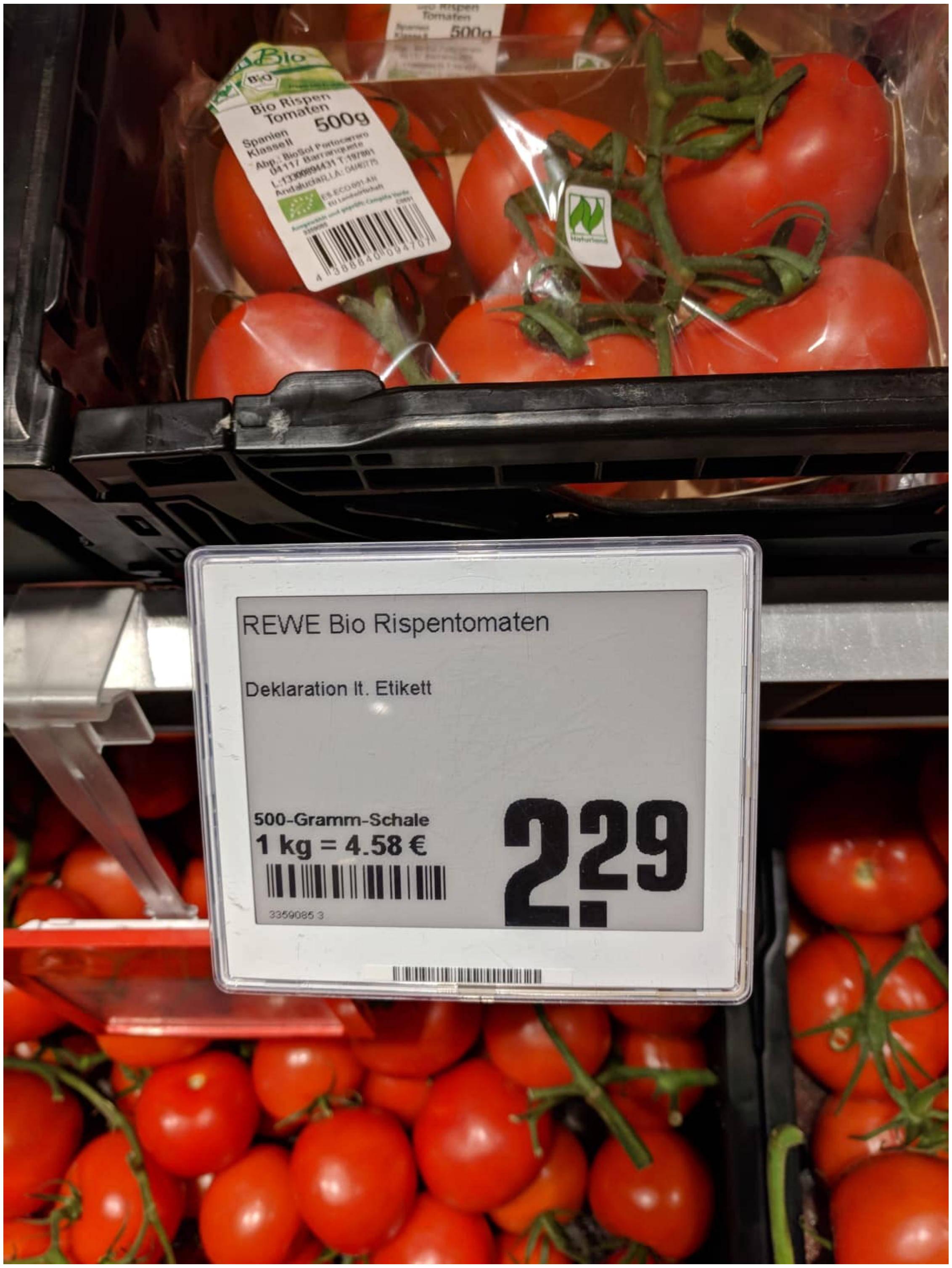
```
brand_mult[108]   1.05  4.1e-3   0.34   0.51    0.8    1.0   1.25   1.83 7173.0     1.0
brand_mult[109]   1.02  4.1e-3   0.34   0.48   0.78   0.97   1.21   1.78 6727.0     1.0
brand_mult[110]    0.8  1.9e-3   0.14   0.53    0.7   0.79   0.89   1.08 5459.0     1.0
brand_mult[111]   0.95  2.9e-3   0.23   0.53    0.8   0.94    1.1   1.43 5916.0     1.0
brand_mult[112]    1.1  4.4e-3   0.37   0.52   0.84   1.05   1.32   1.99 7284.0     1.0
brand_mult[113]   1.23  2.6e-3   0.15   0.96   1.13   1.23   1.33   1.54 3110.0     1.0
brand_mult[114]   1.24  3.3e-3   0.25   0.78   1.06   1.23    1.4   1.75 5896.0     1.0
brand_mult[115]   0.85  2.4e-3    0.2    0.5   0.71   0.83   0.98   1.28 6537.0     1.0
brand_mult[116]   0.97  3.4e-3   0.31   0.48   0.75   0.93   1.15    1.7 8316.0     1.0
brand_mult[117]   0.82  2.7e-3   0.22   0.45   0.67    0.8   0.96    1.3 6607.0     1.0
brand_mult[118]   1.18  3.7e-3   0.33   0.63   0.95   1.15    1.4    1.9 8011.0     1.0
brand_mult[119]    1.0  4.0e-3   0.33   0.49   0.76   0.96   1.19   1.76 6613.0     1.0
brand_mult[120]   2.65  6.2e-3   0.42   1.87   2.36   2.64   2.91   3.48 4491.0     1.0
brand_mult[121]   1.36  3.3e-3   0.23   0.92    1.2   1.36   1.52   1.85 5041.0     1.0
brand_mult[122]   0.78  2.2e-3   0.17   0.47   0.67   0.77    0.9   1.15 6156.0     1.0
brand_mult[123]    1.2  4.3e-3   0.38   0.59   0.92   1.16   1.44   2.04 7802.0     1.0
brand_mult[124]   1.49  4.5e-3   0.39   0.79   1.21   1.47   1.75    2.3 7701.0     1.0
brand_mult[125]   1.54  4.4e-3   0.33   0.93   1.31   1.52   1.76   2.23 5685.0     1.0
brand_mult[126]   2.22  5.4e-3   0.37   1.51   1.96   2.21   2.46   2.98 4744.0     1.0
brand_mult[127]   2.57  6.2e-3   0.39   1.85    2.3   2.56   2.83   3.36 3947.0     1.0
brand_mult[128]   0.98  4.2e-3   0.34   0.47   0.74   0.93   1.18   1.78 6627.0     1.0
brand_mult[129]   0.88  2.8e-3   0.25   0.46    0.7   0.85   1.03   1.43 8121.0     1.0
brand_mult[130]   0.79  2.1e-3   0.19   0.47   0.66   0.78   0.91   1.21 8021.0     1.0
brand_mult[131]   0.91  2.7e-3   0.24   0.49   0.74   0.89   1.06   1.44 8118.0     1.0
brand_mult[132]   0.64  1.5e-3   0.08   0.49   0.59   0.64    0.7   0.82 3352.0     1.0
brand_mult[133]   0.56  1.2e-3   0.08   0.42   0.51   0.56   0.61   0.72 4058.0     1.0
brand_mult[134]   1.07  3.3e-3   0.26    0.6   0.89   1.06   1.25   1.63 6369.0     1.0
brand_mult[135]   0.82  2.6e-3   0.24   0.43   0.64   0.79   0.97   1.34 8670.0     1.0
brand_mult[136]   1.24  4.6e-3    0.4   0.59   0.94   1.19   1.48   2.14 7553.0     1.0
brand_mult[137]   1.36  4.0e-3   0.32   0.78   1.15   1.35   1.56   2.05 6432.0     1.0
brand_mult[138]   1.42  3.1e-3    0.1   1.24   1.36   1.42   1.49   1.62  972.0     1.0
brand_mult[139]   1.06  3.9e-3   0.35    0.5   0.81   1.02   1.25   1.83 7779.0     1.0
brand_mult[140]   1.15  4.8e-3   0.41   0.54   0.85   1.09   1.38   2.11 7461.0     1.0
brand_mult[141]   1.01  3.4e-3   0.29   0.54    0.8   0.98   1.18   1.65 7197.0     1.0
brand_mult[142]   1.25  5.5e-3   0.46   0.56   0.92   1.18   1.51   2.35 7040.0     1.0
brand_mult[143]   0.68  2.0e-3   0.17   0.39   0.56   0.66   0.78   1.03 7226.0     1.0
brand_mult[144]   1.39  4.0e-3   0.32    0.8   1.17   1.38    1.6   2.07 6206.0     1.0
brand_mult[145]   0.44  1.0e-3   0.07   0.31   0.39   0.44   0.48   0.58 4799.0     1.0
brand_mult[146]   0.53  1.2e-3   0.08   0.39   0.48   0.53   0.58    0.7 3763.0     1.0
store_mult[1]     0.56  4.4e-3    0.1   0.39   0.49   0.55   0.63   0.79  569.0     1.0
store_mult[2]     0.85  6.5e-3   0.16   0.58   0.73   0.84   0.95   1.19  570.0     1.0
store_mult[3]     0.74  5.7e-3   0.14   0.51   0.64   0.73   0.82   1.03  557.0     1.0
store_mult[4]     0.79  6.0e-3   0.14   0.55   0.68   0.78   0.88   1.11  574.0     1.0
loc_mult[1]       0.86  4.4e-3    0.1   0.68   0.79   0.86   0.93   1.07  498.0     1.0
loc_mult[2]       0.88  4.5e-3    0.1    0.7   0.81   0.88   0.95    1.1  537.0     1.0
loc_mult[3]       0.85  4.4e-3    0.1   0.67   0.78   0.85   0.92   1.05  498.0     1.0
loc_mult[4]       0.81  4.2e-3   0.09   0.64   0.74    0.8   0.87   1.01  511.0     1.0
loc_mult[5]       0.77  4.0e-3   0.09    0.6    0.7   0.76   0.82   0.95  509.0     1.0
```

```
loc_mult[6]        0.94  4.9e-3   0.11   0.74   0.86   0.93   1.01   1.18  521.0    1.0
loc_mult[7]        0.87  4.5e-3    0.1   0.69    0.8   0.87   0.94   1.08  504.0    1.0
loc_mult[8]        0.91  5.0e-3   0.12   0.69   0.82    0.9   0.98   1.18  605.0    1.0
loc_mult[9]         0.9  4.6e-3   0.11   0.71   0.82   0.89   0.97   1.12  550.0    1.0
loc_mult[10]       1.04  5.4e-3   0.13   0.81   0.95   1.03   1.13   1.31  597.0    1.0
base_price[1]      3.59    0.04   0.79   2.27    3.0   3.51   4.08   5.34  384.0    1.0
base_price[2]      2.08    0.02   0.47    1.3   1.73   2.03   2.36   3.14  374.0    1.0
base_price[3]      4.98    0.06   1.09   3.19   4.17   4.86   5.66   7.45  377.0    1.0
base_price[4]      1.99    0.02   0.46   1.25   1.65   1.94   2.27   3.01  407.0    1.0
base_price[5]       1.6    0.02   0.39   0.97   1.32   1.55   1.84    2.5  445.0    1.0
base_price[6]      3.83    0.04   0.86    2.4   3.19   3.73   4.37   5.74  393.0    1.0
base_price[7]      1.53    0.02   0.36   0.93   1.27   1.49   1.75   2.32  436.0    1.0
base_price[8]      6.84    0.08   1.51   4.34   5.72   6.65    7.8  10.24  379.0    1.0
base_price[9]      4.56    0.05    1.0   2.88   3.82   4.45   5.17   6.78  380.0    1.0
base_price[10]    21.78    0.25   4.74  13.95  18.28  21.29   24.7  32.45  368.0    1.0
sigma2             1.21  2.5e-4   0.02   1.16   1.19   1.21   1.22   1.25 7307.0    1.0
lp__              -1310    0.26   9.69  -1330  -1316  -1309  -1303  -1292 1359.0    1.0

Samples were drawn using NUTS at Sat Nov 10 12:26:15 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).
```

6 Appendix: supermarket photos (Data collection: Nov 6, 2018, 10AM)





Bio Rispen Tomaten
500g
Spanien Klasse I
REWE Bio Rispentomaten
Deklaration lt. Etikett
500-Gramm-Schale
1 kg = 4.58 €
2.29
3359085 3