Asmaa Aly

LBA

Fall 2019

Python Implementation

[4]:
```
import pystan
import numpy as np
import pandas as pd
from scipy import stats as sts
import matplotlib.pyplot as plt
from IPython.display import display
```

[5]:
```
'''
For the data pre-processing, I am assigning the neighborhood names for each␣
↪student
'''
london_students = [
"Gelana Tostaeva",
"Sara Merner",
"Gera",
"Evan Buckman",
"Frances Pak",
"Erika Sloan",
"Sonia",
"Michelle Hackl",
"Mandla",
"Nikesh Shrestha",
"Hana Mcmahon-Cole",
"Barbara"]
San_Francisco_students = [" Jingren", "Vu"]
```

Data Pre-processing

Pre-processing the responce data

[6]:
```
"""
Read the CSV file that conatins all enteries as updated in Nov 7th
"""
rawdata = pd.read_csv("CS146 LBA data gathering (Fall 2019) (Responses) - Form␣
↪Responses 1(1) .csv")
#Take a peak into the data
rawdata.head(5)
```

[6]:
```
            Unnamed: 0                       Unnamed: 1      Unnamed: 2  \
0            Timestamp                    Email Address       Your name
1                  NaN                              NaN             NaN
2  10/28/2019 13:14:59  brian.swanberg@minerva.kgi.edu  Brian Swanberg
3  10/29/2019 14:19:19    emma.stiefel@minerva.kgi.edu    Emma Stiefel
4  10/29/2019 21:22:30            taha@minerva.kgi.edu            Taha
```

```
     Unnamed: 3                              Unnamed: 4  \
0  Grocery store             Grocery store street address
1            NaN                                      NaN
2           ALDI                     Rummelsburger Str. 98
3           REWE                    Karl-Marx-Straße 92-98
4           ALDI  Hermannstraße 72, 12049 Berlin, Germany

                 Apples           Unnamed: 6              Unnamed: 7  \
0  Product 1 quantity (kg)  Product 1 price (€)  Product 2 quantity (kg)
1                     NaN                  NaN                      NaN
2                    0.88                  2.2                        1
3                       1                 2.49                        1
4                       1                 2.99                        1

          Unnamed: 8               Unnamed: 9  ...  \
0  Product 2 price (€)  Product 3 quantity (kg)  ...
1                NaN                      NaN  ...
2               1.88                      0.6  ...
3               1.49                        1  ...
4               1.79                      0.8  ...

                Unnamed: 55          Unnamed: 56  \
0  Product 2 quantity (count)  Product 2 price (€)
1                        NaN                  NaN
2                          6                 1.59
3                          1                 0.25
4                         10                 1.19

                Unnamed: 57          Unnamed: 58          Chicken breasts  \
0  Product 3 quantity (count)  Product 3 price (€)  Product 1 quantity (kg)
1                        NaN                  NaN                      NaN
2                         10                 1.59                      0.6
3                          6                 1.59                        1
4                          6                 1.59                      0.6

         Unnamed: 60              Unnamed: 61          Unnamed: 62  \
0  Product 1 price (€)  Product 2 quantity (kg)  Product 2 price (€)
1                NaN                      NaN                  NaN
2               3.99                        1                 5.99
3               13.9                        1                 9.99
4               3.99                        1                 5.99

             Unnamed: 63          Unnamed: 64
0  Product 3 quantity (kg)  Product 3 price (€)
1                     NaN                  NaN
2                     NaN                  NaN
3                       1                 9.98
```

```
4                     0.35                  3.99

[5 rows x 65 columns]
```

[7]:
```python
"""
Change the names of the columns and remove unnecssary info: TimeStamp and Email␣
↪Address
"""
rawdata.columns = rawdata.iloc[0]
rawdata = rawdata.drop([0,1], axis= 0)
rawdata.head(2)
```

[7]:
```
0             Timestamp                   Email Address       Your name  \
2  10/28/2019 13:14:59  brian.swanberg@minerva.kgi.edu  Brian Swanberg
3  10/29/2019 14:19:19    emma.stiefel@minerva.kgi.edu    Emma Stiefel

0 Grocery store Grocery store street address Product 1 quantity (kg)  \
2          ALDI       Rummelsburger Str. 98                     0.88
3          REWE      Karl-Marx-Straße 92-98                        1

0 Product 1 price (€) Product 2 quantity (kg) Product 2 price (€)  \
2                2.2                       1                1.88
3               2.49                       1                1.49

0 Product 3 quantity (kg)  ... Product 2 quantity (count) Product 2 price (€)  \
2                     0.6  ...                          6                1.59
3                       1  ...                          1                0.25

0 Product 3 quantity (count) Product 3 price (€) Product 1 quantity (kg)  \
2                         10                1.59                     0.6
3                          6                1.59                       1

0 Product 1 price (€) Product 2 quantity (kg) Product 2 price (€)  \
2               3.99                       1                5.99
3               13.9                       1                9.99

0 Product 3 quantity (kg) Product 3 price (€)
2                     NaN                 NaN
3                       1                9.98

[2 rows x 65 columns]
```

[8]:
```python
responcedata = rawdata.drop(["Timestamp", "Email Address"], axis= 1)
```

[9]:
```python
# name the responce data to be Supermarket for simplicity in matching later on.
responcedata.rename({'Grocery store street address': 'Supermarket'}, axis=1,␣
↪inplace=True)
```

```
responcedata.head(1)
```

```
//anaconda3/lib/python3.7/site-packages/pandas/core/frame.py:4025:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: http://pandas.pydata.org/pandas-
docs/stable/indexing.html#indexing-view-versus-copy
  return super(DataFrame, self).rename(**kwargs)
```

[9]:

```
0       Your name Grocery store            Supermarket  \
2  Brian Swanberg          ALDI  Rummelsburger Str. 98

0 Product 1 quantity (kg) Product 1 price (€) Product 2 quantity (kg)  \
2                    0.88                 2.2                       1

0 Product 2 price (€) Product 3 quantity (kg) Product 3 price (€)  \
2                1.88                     0.6                1.89

0 Product 1 quantity (kg)  ... Product 2 quantity (count) Product 2 price (€)  \
2                       1  ...                          6                1.59

0 Product 3 quantity (count) Product 3 price (€) Product 1 quantity (kg)  \
2                         10                1.59                     0.6

0 Product 1 price (€) Product 2 quantity (kg) Product 2 price (€)  \
2                3.99                       1                5.99

0 Product 3 quantity (kg) Product 3 price (€)
2                     NaN                 NaN

[1 rows x 63 columns]
```

Pre-processing the original assignment data

[10]:

```
"""
Do the same pre-procesing steps to the initial assignment sheet.
Change the name of the columns and remove the excess raw that
the has NA values for each entery
"""
datacollectiondf = pd.read_csv("CS146, Fall 2019, LBA data collection - Berlin␣
 ↪supermarkets.csv")
datacollectiondf.columns = datacollectiondf.iloc[2]
datacollectiondf = datacollectiondf.drop(datacollectiondf.index[2])
assigneddata = datacollectiondf.drop([0,1], axis = 0)
assigneddata =  assigneddata.drop(["Student index","Map location", "GPS",␣
 ↪"Estimated travel time (min)"], axis = 1)
```

```
assigneddata.head(3)
```

[10]:
```
2 Student name                Supermarket Neighborhood
3       Berfin  ALDI, Eisenbahnstraße 42    Kreuzberg
4         Khoi     EDEKA, Annenstraße 4A        Mitte
5        Sanny    EDEKA, Fischerinsel 12        Mitte
```

[11]:
```
"""
Since we are trying to add the column of the neighborhood to the response data,
we will match each entry in the response data to the supermarket address. To
achieve maximized efficiency, we removed all numbers or special characters.
"""
assigneddata[["Supermarket"]] = assigneddata[["Supermarket"]].replace(r'(?:.*,␣
↪)*(.*), [0-9]+.*', r'\1', regex=True)
display(assigneddata.head())
```

```
2 Student name                     Supermarket Neighborhood
3       Berfin        ALDI, Eisenbahnstraße 42    Kreuzberg
4         Khoi           EDEKA, Annenstraße 4A        Mitte
5        Sanny          EDEKA, Fischerinsel 12        Mitte
6          Ayo         REWE, Skalitzer Str. 134    Kreuzberg
7        Trang  Lidl, Heinrich-Heine-Straße 30        Mitte
```

[12]:
```
# drop the unknown values in the pre-assigned data.
#Make a new dataframe that contains only the important infromation for matching.
neighborhoods = assigneddata[[
    "Student name", "Supermarket", "Neighborhood"]].dropna()[["Supermarket",␣
↪"Neighborhood"]]
#Make the supermarket data lower case to ease the matching
neighborhoods.Supermarket = neighborhoods.Supermarket.str.lower()
responcedata.Supermarket = responcedata.Supermarket.str.lower()
#The assignment data had the name of the store in the address which is redundent.
↪
neighborhoods[["Supermarket"]] = neighborhoods[["Supermarket"]].replace(
    r'(?:aldi|edeka|rewe|lidl), (.*)$', r'\1', regex=True)
responcedata[["Supermarket"]] = responcedata[["Supermarket"]].replace(
    r'(?:aldi|edeka|rewe|lidl|berlin)(?: |, )(.*)$', r'\1', regex=True).replace(
    r'(?:.*, )*(.*), [0-9]+.*', r'\1', regex=True)
display(neighborhoods.head(3))
display(responcedata.head(3))
```

```
//anaconda3/lib/python3.7/site-packages/pandas/core/generic.py:5096:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-
```

```
docs/stable/indexing.html#indexing-view-versus-copy
  self[name] = value
//anaconda3/lib/python3.7/site-packages/pandas/core/frame.py:3391:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: http://pandas.pydata.org/pandas-
docs/stable/indexing.html#indexing-view-versus-copy
  self[k1] = value[k2]

2          Supermarket Neighborhood
3  eisenbahnstraße 42    Kreuzberg
4      annenstraße 4a        Mitte
5     fischerinsel 12        Mitte


0       Your name Grocery store             Supermarket  \
2  Brian Swanberg          ALDI   rummelsburger str. 98
3    Emma Stiefel          REWE  karl-marx-straße 92-98
4            Taha          ALDI       hermannstraße 72

0 Product 1 quantity (kg) Product 1 price (€) Product 2 quantity (kg)  \
2                    0.88                 2.2                       1
3                       1                2.49                       1
4                       1                2.99                       1

0 Product 2 price (€) Product 3 quantity (kg) Product 3 price (€)  \
2                1.88                     0.6                1.89
3                1.49                       1                2.49
4                1.79                     0.8                1.89

0 Product 1 quantity (kg)  ... Product 2 quantity (count) Product 2 price (€)  \
2                       1  ...                          6                1.59
3                       1  ...                          1                0.25
4                       1  ...                         10                1.19

0 Product 3 quantity (count) Product 3 price (€) Product 1 quantity (kg)  \
2                         10                1.59                     0.6
3                          6                1.59                       1
4                          6                1.59                     0.6

0 Product 1 price (€) Product 2 quantity (kg) Product 2 price (€)  \
2                3.99                       1                5.99
3                13.9                       1                9.99
4                3.99                       1                5.99

0 Product 3 quantity (kg) Product 3 price (€)
2                     NaN                 NaN
```

```
3                        1              9.98
4                     0.35              3.99

[3 rows x 63 columns]
```

[13]:
```python
"""
Do the matching to merge the neighborhood data frame using matching on␣
↪supermarket values.
If the student's name is in the London list, assign the neighborhood to be␣
↪London.
"""
meta_data_with_neighborhood = responcedata.merge(neighborhoods, how='left',␣
↪on='Supermarket')
london_mask = meta_data_with_neighborhood[["Your name"]].isin(london_students).
↪values.flatten()
meta_data_with_neighborhood.loc[london_mask, "Neighborhood"] = "London"
```

[14]:
```python
"""
Handle each of the enteries that were not stated in the assigned data.
Seoul and San Francisco is handled by the following lines of code.
"""
meta_data_with_neighborhood.loc[
    meta_data_with_neighborhood["Your name"] == "Vy Tran", "Neighborhood"] =␣
↪"Seoul"
meta_data_with_neighborhood.loc[
    meta_data_with_neighborhood["Grocery store"] == "Safeway", "Neighborhood"] =␣
↪"San Francisco"
```

[15]:
```python
"""
Print the rows that didn't have a neighborhood assigned to it.
"""
unmatched_rows = meta_data_with_neighborhood[meta_data_with_neighborhood.
↪Neighborhood.isnull()]
print("These rows don't have a neighborhood :")
display(unmatched_rows)
```

```
These rows don't have a neighborhood :

          Your name Grocery store                 Supermarket  \
12     Emma Stiefel         EDEKA            grunerstraße 20
23           Berfin          ALDI  berlin kreuzberg straße 39
24   Berfin Karaman          ALDI          eisenbahnstrasse 42
26    Dennis Kageni          Lidl            friedenstraße 94a
37            Yanal          ALDI         frankfurter allee 117
39            Yanal          ALDI         landsberger allee 277
42            Ahmed         EDEKA              kottbusser damm 80
61       Tom Kremer          ALDI             invalidenstraße 59
```

65	Ebuka	Lidl	frankfurter allee 212
67	Ebuka	REWE	litfaß-platz 4

	Product 1 quantity (kg)	Product 1 price (€)	Product 2 quantity (kg)	\
12	1	2.49	1	
23	1	2.99	1	
24	0.6	1.95	2	
26	2	2.49	0.7	
37	1	1.39	2	
39	1	2.69	1	
42	1	2.49	1	
61	1	2.49	2	
65	1	1.79	0.7	
67	0.65	2.69	1	

	Product 2 price (€)	Product 3 quantity (kg)	Product 3 price (€)	\
12	2.49	1	1.99	
23	1.79	1	2.29	
24	2.29	1	1.39	
26	2.29	1	1.99	
37	2.29	1	1.79	
39	1.89	1	1.79	
42	2.49	1	1.66	
61	2.49	1	1.99	
65	2.29	1	2.49	
67	2.49	1	1.49	

	Product 1 quantity (kg)	...	Product 2 price (€)	\
12	1	...	3.99	
23	1	...	1.19	
24	1	...	1.19	
26	1	...	2.65	
37	1	...	1.69	
39	1	...	1.69	
42	1	...	1.19	
61	1	...	1.19	
65	1	...	1.59	
67	1	...	1.59	

	Product 3 quantity (count)	Product 3 price (€)	Product 1 quantity (kg)	\
12	6	2.89	1	
23	6	1.59	0.6	
24	6	1.59	0.35	
26	6	0.99	0.8	
37	10	1.19	0.6	
39	10	1.59	0.6	
42	6	1.49	1	
61	NaN	NaN	1	

```
65                           10               2.65                   0.5
67                            6               1.29                 0.245

   Product 1 price (€) Product 2 quantity (kg) Product 2 price (€)  \
12                29.9                       1                 4.9
23                3.99                     0.4                2.99
24                2.99                     0.6                3.99
26                4.69                     0.6                2.99
37                3.99                       1                5.99
39                3.99                       1                5.99
42                9.99                       1                8.99
61                6.99                     0.6                4.99
65                2.99                     0.6                2.99
67                3.18                   0.309                4.01

   Product 3 quantity (kg) Product 3 price (€) Neighborhood
12                       1                6.65          NaN
23                     0.4                2.79          NaN
24                       1                5.99          NaN
26                     NaN                 NaN          NaN
37                     NaN                 NaN          NaN
39                     NaN                 NaN          NaN
42                     NaN                 NaN          NaN
61                     NaN                 NaN          NaN
65                       1                5.99          NaN
67                     NaN                 NaN          NaN

[10 rows x 64 columns]
```

[16]:
```
meta_data_with_neighborhood.loc[
    meta_data_with_neighborhood["Supermarket"] == "eisenbahnstrasse 42",␣
 ↪"Neighborhood"] = "Kreuzberg"
meta_data_with_neighborhood.loc[
    meta_data_with_neighborhood["Supermarket"] == "friedenstraße 94a",␣
 ↪"Neighborhood"] = "Friedrichshain"
meta_data_with_neighborhood.loc[
    meta_data_with_neighborhood["Supermarket"] == "frankfurter allee 117",␣
 ↪"Neighborhood"] = "Lichtenberg"
meta_data_with_neighborhood.loc[
    meta_data_with_neighborhood["Supermarket"] == "landsberger allee 277",␣
 ↪"Neighborhood"] = "Friedrichshain"
meta_data_with_neighborhood.loc[
    meta_data_with_neighborhood["Supermarket"] == "kottbusser damm 80",␣
 ↪"Neighborhood"] = "Kreuzberg"
```

[17]:
```python
unmatched_rows2 = meta_data_with_neighborhood[meta_data_with_neighborhood.
 ↪Neighborhood.isnull()]
print("These rows don't have a neighborhood :")
display(unmatched_rows2)
```

```
These rows don't have a neighborhood :

       Your name Grocery store                  Supermarket  \
12  Emma Stiefel         EDEKA              grunerstraße 20
23        Berfin          ALDI  berlin kreuzberg straße 39
61    Tom Kremer          ALDI          invalidenstraße 59
65         Ebuka          Lidl        frankfurter allee 212
67         Ebuka          REWE              litfaß-platz 4

   Product 1 quantity (kg) Product 1 price (€) Product 2 quantity (kg)  \
12                       1                2.49                       1
23                       1                2.99                       1
61                       1                2.49                       2
65                       1                1.79                     0.7
67                    0.65                2.69                       1

   Product 2 price (€) Product 3 quantity (kg) Product 3 price (€)  \
12                2.49                       1                1.99
23                1.79                       1                2.29
61                2.49                       1                1.99
65                2.29                       1                2.49
67                2.49                       1                1.49

   Product 1 quantity (kg)  ... Product 2 price (€)  \
12                       1  ...                3.99
23                       1  ...                1.19
61                       1  ...                1.19
65                       1  ...                1.59
67                       1  ...                1.59

   Product 3 quantity (count) Product 3 price (€) Product 1 quantity (kg)  \
12                          6                2.89                       1
23                          6                1.59                     0.6
61                        NaN                 NaN                       1
65                         10                2.65                     0.5
67                          6                1.29                   0.245

   Product 1 price (€) Product 2 quantity (kg) Product 2 price (€)  \
12                29.9                       1                4.9
23                3.99                     0.4               2.99
61                6.99                     0.6               4.99
65                2.99                     0.6               2.99
67                3.18                   0.309               4.01
```

```
   Product 3 quantity (kg) Product 3 price (€) Neighborhood
12                       1                6.65          NaN
23                     0.4                2.79          NaN
61                     NaN                 NaN          NaN
65                       1                5.99          NaN
67                     NaN                 NaN          NaN

[5 rows x 64 columns]
```

[18]:

```
"""
White space in the each of the locations might have caused the issue with
"""
meta_data_with_neighborhood.loc[
    meta_data_with_neighborhood["Supermarket"].str.contains('kreuzberg straße␣
 ↪39', na=False), "Neighborhood"] = "Schöneberg"
meta_data_with_neighborhood.loc[
    meta_data_with_neighborhood["Supermarket"].str.contains('gruner', na=False),␣
 ↪"Neighborhood"] = "Mitte"
meta_data_with_neighborhood.loc[
    meta_data_with_neighborhood["Supermarket"].str.contains('litfaß-platz',␣
 ↪na=False), "Neighborhood"] = "Mitte"
meta_data_with_neighborhood.loc[
    meta_data_with_neighborhood["Supermarket"].str.contains('invalidenstraße',␣
 ↪na=False), "Neighborhood"] = "Mitte"
meta_data_with_neighborhood.loc[
    meta_data_with_neighborhood["Supermarket"].str.contains('frankfurter allee',␣
 ↪na=False), "Neighborhood"] = "Lichtenberg"
```

[19]:

```
#double checking that the all rows are matched
unmatched_rows3 = meta_data_with_neighborhood[meta_data_with_neighborhood.
 ↪Neighborhood.isnull()]
print("These rows don't have a neighborhood :")
display(unmatched_rows3)
```

```
These rows don't have a neighborhood :

Empty DataFrame
Columns: [Your name, Grocery store, Supermarket, Product 1 quantity (kg), Product 1 price (€), P
Index: []

[0 rows x 64 columns]
```

[20]:

```
"""
This function normalises the price for each of the products
"""
```

```python
def normalise (df):
    for i in range(3,63):
        df.iloc[ : ,i]  = pd.to_numeric(df.iloc[ : ,i])
    #Normalise the price of apples
    for j in range(3,8,2):
        quanitity1 = df.iloc[ : ,j]
        cost1 = df.iloc[ : , j+1 ]
        df["Price of Apples " + str((j-1)//2)] = cost1/quanitity1
    for k in range(9,14,2):
        quanitity2 = df.iloc[ : ,k]
        cost2 = df.iloc[ : , k+1 ]
        df["Price of Bananas " + str((k-7)//2)] = cost2/quanitity2
    for l in range(15,20,2):
        quanitity3 = df.iloc[ : ,l]
        cost3 = df.iloc[ : , l+1 ]
        df["Price of Tomatoes " + str((l-13)//2)] = cost3/quanitity3
    for m in range(21,26,2):
        quanitity4 = df.iloc[ : ,m]
        cost4 = df.iloc[ : , m+1 ]
        df["Price of Potatoes " + str((m-19)//2)] = cost4/quanitity4
    for n in range(27,32,2):
        quanitity5 = df.iloc[ : ,n]
        cost5 = df.iloc[ : , n+1 ]
        df["Price of Flour " + str((n-25)//2)] = cost5/quanitity5
    for o in range(33,38,2):
        quanitity6 = df.iloc[ : ,o]
        cost6 = df.iloc[ : , o+1 ]
        df["Price of Rice " + str((o-31)//2)] = cost6/quanitity6
    for u in range(39,44,2):
        quanitity7 = df.iloc[ : ,u]
        cost7 = df.iloc[ : , u+1 ]
        df["Price of Milk " + str((u-37)//2)] = cost7/quanitity7
    for v in range(45,50,2):
        quanitity8 = df.iloc[ : ,v]
        cost8 = df.iloc[ : , v+1 ]
        df["Price of Butter " + str((v-43)//2)] = cost8/quanitity8
    for w in range(51,56,2):
        quanitity9 = df.iloc[ : ,w]
        cost9 = df.iloc[ : , w+1 ]
        df["Price of Eggs " + str((w-49)//2)] = cost9/quanitity9
    for z in range(57,62,2):
        quanitity10 = df.iloc[ : ,z]
        cost10 = df.iloc[ : , z+1 ]
        df["Price of Chicken Breasts " + str((z-55)//2)] = cost10/quanitity10
    return df
```

[21]:
```python
#normalise the prices and have them in new columns
normalised = normalise(meta_data_with_neighborhood)
#drop the rest of the columns- the original ones with prices
normalised_full = normalised.drop(normalised.iloc[:, 3:63], axis=1)
#drop the name and the location of the supermarket columns
normalised_full2 = normalised_full.drop(['Your name', 'Supermarket'], axis=1,␣
 ↪inplace=False)
normalised_full2.head(5)
```

[21]:
```
  Grocery store Neighborhood Price of Apples 1 Price of Apples 2  \
0          ALDI  Lichtenberg               2.5              1.88
1          REWE     Neukölln              2.49              1.49
2          REWE     Neukölln              2.49              1.49
3          ALDI     Neukölln              2.99              1.79
4          Lidl        Mitte              1.79           2.65333

  Price of Apples 3 Price of Bananas 1 Price of Bananas 2 Price of Bananas 3  \
0              3.15               1.69               0.99                NaN
1              2.49               0.99               1.69               1.59
2              2.49               0.99               1.69               1.59
3            2.3625               1.15               1.69              1.495
4            2.3625               1.09               1.69               1.19

  Price of Tomatoes 1 Price of Tomatoes 2  ... Price of Milk 3  \
0             3.52308                2.98  ...            0.99
1                 6.9                4.58  ...            0.79
2                 6.9                4.58  ...            0.79
3                3.58                1.99  ...            0.99
4             7.11429                1.89  ...            1.15

  Price of Butter 1 Price of Butter 2 Price of Butter 3 Price of Eggs 1  \
0              5.56              6.36              9.56           0.119
1              5.56              5.56              9.56        0.281667
2              5.56              5.56              9.56        0.281667
3              9.56              5.16              5.16           0.159
4              9.56              6.36              6.76           0.119

  Price of Eggs 2 Price of Eggs 3 Price of Chicken Breasts 1  \
0           0.265           0.159                       6.65
1            0.25           0.265                       13.9
2            0.25           0.265                       13.9
3           0.119           0.265                       6.65
4           0.169           0.265                       6.65

  Price of Chicken Breasts 2 Price of Chicken Breasts 3
0                       5.99                        NaN
1                       9.99                       9.98
```

```
2                     9.99                      9.98
3                     5.99                      11.4
4                    6.975                       NaN

[5 rows x 32 columns]
```

[22]:
```python
#isolate the price columns in a sperate dataframe
df_Product = normalised_full2.iloc[:,2::1]
df_Product.head(5)
```

[22]:
```
  Price of Apples 1 Price of Apples 2 Price of Apples 3 Price of Bananas 1  \
0               2.5              1.88              3.15               1.69
1              2.49              1.49              2.49               0.99
2              2.49              1.49              2.49               0.99
3              2.99              1.79            2.3625               1.15
4              1.79           2.65333            2.3625               1.09

  Price of Bananas 2 Price of Bananas 3 Price of Tomatoes 1  \
0               0.99                NaN             3.52308
1               1.69               1.59                 6.9
2               1.69               1.59                 6.9
3               1.69              1.495                3.58
4               1.69               1.19             7.11429

  Price of Tomatoes 2 Price of Tomatoes 3 Price of Potatoes 1  ...  \
0                2.98                1.89               0.556  ...
1                4.58             5.68571            0.563333  ...
2                4.58             5.68571            0.563333  ...
3                1.99                4.58               0.556  ...
4                1.89                3.58                0.75  ...

  Price of Milk 3 Price of Butter 1 Price of Butter 2 Price of Butter 3  \
0            0.99              5.56              6.36              9.56
1            0.79              5.56              5.56              9.56
2            0.79              5.56              5.56              9.56
3            0.99              9.56              5.16              5.16
4            1.15              9.56              6.36              6.76

  Price of Eggs 1 Price of Eggs 2 Price of Eggs 3 Price of Chicken Breasts 1  \
0           0.119           0.265           0.159                       6.65
1        0.281667            0.25           0.265                       13.9
2        0.281667            0.25           0.265                       13.9
3           0.159           0.119           0.265                       6.65
4           0.119           0.169           0.265                       6.65

  Price of Chicken Breasts 2 Price of Chicken Breasts 3
0                       5.99                        NaN
```

```
1                        9.99                       9.98
2                        9.99                       9.98
3                        5.99                       11.4
4                       6.975                        NaN

[5 rows x 30 columns]
```

[23]:
```
#Treat each price as an individual entity by splitting each into a sperate row
df_Product_Price = pd.melt(df_Product).rename(
    columns={"value":"Product Price"}).rename(
    columns={"variable":"Product"})
df_Product_Price.head(3)
```

[23]:
```
             Product Product Price
0  Price of Apples 1           2.5
1  Price of Apples 1          2.49
2  Price of Apples 1          2.49
```

[24]:
```
#change the shape of the store brands and neighborhood ones.
neighborhood_column = list(meta_data_with_neighborhood.Neighborhood.values.
 ↪astype(str)) * 10 * 3
stores_brand_column = list(meta_data_with_neighborhood["Grocery store"].values.
 ↪flatten().astype(str)) * 10 * 3
#connected the product prices with the neighborhood and the store brands.
df_full_info = pd.concat([df_Product_Price,
                     pd.DataFrame({'Neighborhood' : neighborhood_column}),
                         pd.DataFrame({'Store_Name' :␣
 ↪stores_brand_column})],axis = 1)
# change the values from "Price of Apples 1" to "Apples"
df_full_info["Product"] = df_full_info["Product"].replace(       # Raplace␣
 ↪product value with numbers
    r'Price of', '', regex=True).replace(       # Raplace product value with␣
 ↪numbers
    r'[0-9]', '', regex=True)
df_full_info["Store_Name"]
#There are some inconsistencies in the store names that will effect creating the␣
 ↪dictionary values
#Standlise the name for each store for that purpose.
df_full_info.loc[
    df_full_info["Store_Name"].str.contains('Sainsbury', na=False, case =␣
 ↪False), "Store_Name"] = "Sainsbury's"
df_full_info.loc[
    df_full_info["Store_Name"].str.contains('Waitrose', na=False, case = False),␣
 ↪"Store_Name"] = "Waitrose & Partners"
df_full_info.loc[
```

```
    df_full_info["Store_Name"].str.contains('Tesco', na=False, case = False),␣
↪"Store_Name"] = "Tesco Express"
df_valid_full_info = df_full_info.dropna().reset_index(drop=True)
```

[25]:
```
df_valid_full_info.head(3)
```

[25]:

	Product	Product Price	Neighborhood	Store_Name
0	Apples	2.5	Lichtenberg	ALDI
1	Apples	2.49	Neukölln	REWE
2	Apples	2.49	Neukölln	REWE

[26]:
```
'''
This function encodes any column in any data frame as an input.
It assigns different numbers for each unique value found in these columns.

'''
def IDfy(df, columns):
    mapper = {}
    for column in columns:
        mapper[column] = {}
        unique_values = list(df[column].unique())
        for i, key in enumerate(unique_values):
            mapper[column][i + 1] = key
            df.loc[df[column].values == key, column] = i + 1

        print("There are {:d} unique values in the {:s} column.".
↪format(len(df[column].unique()), column))
    return(df, mapper)
#Encode the columns of interest.
df, mapper = IDfy(df_valid_full_info.copy(),
                  ["Product", "Store_Name", "Neighborhood"])

display(df.head(5))
```

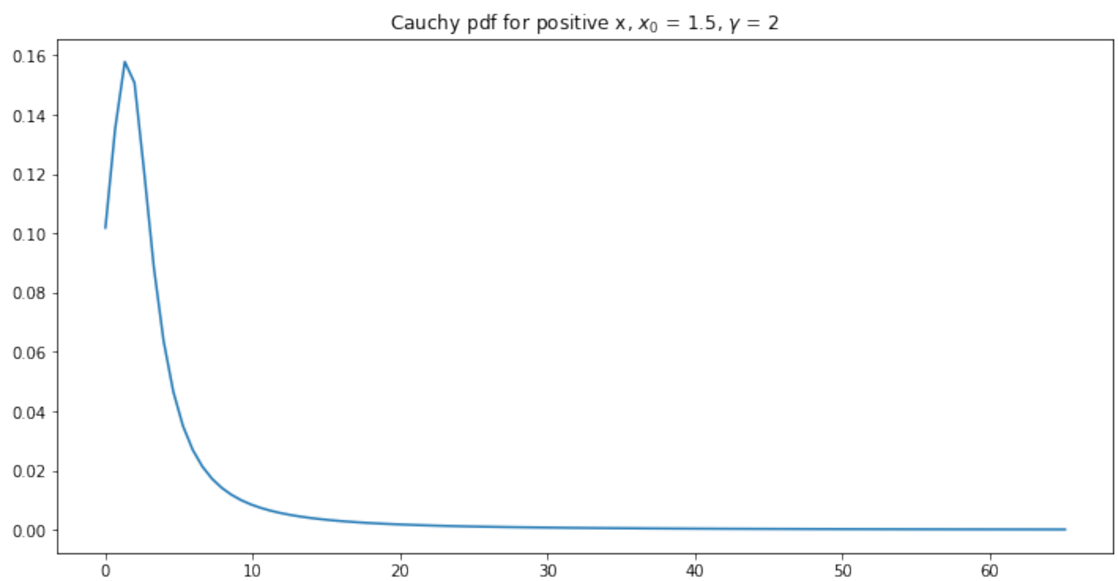
```
There are 10 unique values in the Product column.
There are 9 unique values in the Store_Name column.
There are 12 unique values in the Neighborhood column.
```

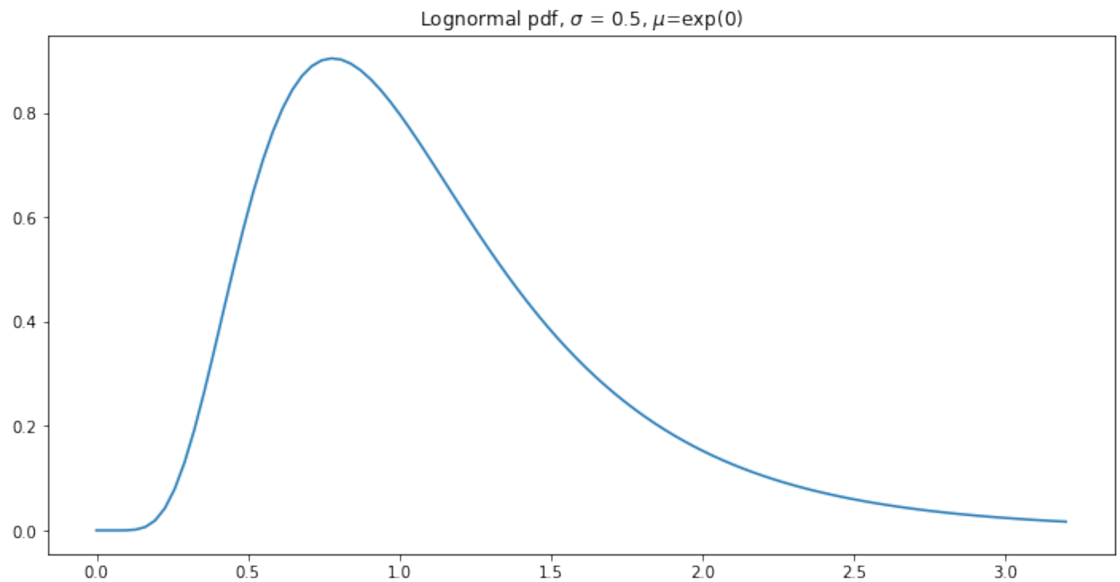
	Product	Product Price	Neighborhood	Store_Name
0	1	2.5	1	1
1	1	2.49	2	2
2	1	2.49	2	2
3	1	2.99	2	1
4	1	1.79	3	3

Building the model

[32]:
```
# Cauchy prior
base_price_prior = sts.cauchy(1.5, 2)
plt.figure(figsize=(12, 6))
x = np.linspace(0,base_price_prior.ppf(0.99), 100)
plt.plot(x, base_price_prior.pdf(x))
plt.title("Cauchy pdf for positive x, $x_0$ = 1.5, $\gamma$ = 2")
plt.show()

# Lognormal prior
multiplier_prior = sts.lognorm(s=0.5, scale=np.exp(0))
plt.figure(figsize=(12, 6))
x = np.linspace(0, multiplier_prior.ppf(0.99), 100)
plt.plot(x, multiplier_prior.pdf(x))
plt.title("Lognormal pdf, $\sigma$ = 0.5, $\mu$=exp(0)")
plt.show()
```





[28]:
```
stan_code = """

// The data block contains all known quantities and any constant hyperparameters.
data {
    int<lower=0> n_data;                            // number of data
    int<lower=0> n_products;                        // number of products
    int<lower=0> n_store_brands;                    // number of store brands
    int<lower=0> n_neighborhoods;                   // number of neighborhoods
    int<lower=1> product_id[n_data];                // product ids
    int<lower=1> neighborhood_id[n_data];           // neighborhood ids
    int<lower=1> store_id[n_data];                  // store brand ids
    real<lower=0> prices[n_data];                   // prices
}

// The parameters block contains all unknown quantities - typically the
// parameters of the model. Stan will generate samples from the posterior
// distributions over all parameters.
parameters {
    real<lower=0> base_price[n_products];                       // base price of␣
 ↪the product
    real<lower=0> store_multiplier[n_store_brands];             // store brande␣
 ↪multiplier
    real<lower=0> neighborhood_multiplier[n_neighborhoods];  // neighborhood␣
 ↪multiplier
    real<lower = 0> sigma;                               // standard deviation of the␣
 ↪normal likelihood function
```

```
}

// The model block contains all probability distributions in the model.
// This of this as specifying the generative model for the scenario.
model {

    // Priors
    sigma ~ inv_gamma(3, 2); // generate random noise from the inverse gamma
    // use half-cauchy distribution
    for (i in 1:n_products) {
        base_price[product_id[i]] ~ cauchy(1.5, 2);         //generate base price
    };

    for (i in 1:n_store_brands) {
        store_multiplier[store_id[i]] ~ lognormal(0, 0.5);  //generate store␣
 ↪multiplier
    };

    for (i in 1:n_neighborhoods) {
        neighborhood_multiplier[neighborhood_id[i]] ~ lognormal(0, 0.5);    //
 ↪generate area multiplier
    };

    // Price Model
    for (i in 1:n_data) {
            prices[i] ~ normal(base_price[product_id[i]]*␣
 ↪store_multiplier[store_id[i]]*neighborhood_multiplier[neighborhood_id[i]],␣
 ↪sigma);

        }
}

"""
```

[29]:
```
stan_model = pystan.StanModel(model_code=stan_code)
```

```
INFO:pystan:COMPILING THE C++ CODE FOR MODEL
anon_model_01cdf048c1f7f4acef403acc1b101bf9 NOW.
```

[30]:
```
stan_data = {
    "n_data" : df.shape[0],
    "n_products" : len(df["Product"].unique()),
    "n_store_brands" : len(df["Store_Name"].unique()),
    "n_neighborhoods" : len(df["Neighborhood"].unique()),
    "product_id" : list(df["Product"]),
    "neighborhood_id" : list(df["Neighborhood"]),
```

```
    "store_id" : list(df["Store_Name"]),
    "prices" : list(df["Product Price"])
    }
```

[31]:
```
results = stan_model.sampling(data=stan_data)
print(results)
```

```
Inference for Stan model: anon_model_8f6c0b08aa7228d78c67669444f06b41.
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_price[1]               2.37    0.15   0.37   1.68   2.07   2.41   2.65   3.02      6   1.52
base_price[2]               1.79    0.14   0.32   1.21   1.55   1.81   2.03   2.38      5   1.46
base_price[3]               3.89    0.26   0.62   2.74   3.36    4.0   4.37   4.91      6   1.62
base_price[4]               1.29     0.1   0.26   0.82   1.11   1.29   1.46   1.79      7   1.38
base_price[5]                1.1    0.08   0.24   0.68   0.92   1.09   1.27   1.55     10   1.31
base_price[6]               3.14    0.22   0.52   2.17    2.7   3.23   3.54   3.99      6   1.59
base_price[7]               0.97    0.05   0.21   0.62   0.82   0.96   1.11    1.4     15   1.21
base_price[8]               7.22     0.5   1.15   5.01   6.24   7.44   8.13   8.98      5   1.69
base_price[9]               0.25    0.06   0.16 7.8e-3   0.13   0.23   0.34   0.62      7   1.29
base_price[10]              8.14    0.57    1.3   5.71   7.02   8.38   9.18  10.13      5   1.69
store_multiplier[1]         0.85    0.03   0.13   0.65   0.75   0.82   0.93   1.15     17   1.28
store_multiplier[2]          1.1    0.04   0.17   0.84   0.97   1.07   1.19   1.51     17   1.28
store_multiplier[3]         0.83    0.03   0.13   0.65   0.73    0.8   0.91   1.15     18   1.27
store_multiplier[4]         1.36    0.09   0.31   0.75   1.14   1.35   1.54   2.08     12   1.63
store_multiplier[5]         1.25    0.05    0.2   0.97    1.1   1.21   1.36    1.7     17   1.27
store_multiplier[6]         1.64    0.11   0.38   0.92   1.37   1.63   1.86   2.55     12   1.57
store_multiplier[7]       1.1e20  1.3e20 2.6e20 8.0e-9   0.18  307.8 1.7e19 9.2e20      4   1.89
store_multiplier[8]       9.6e15  9.0e15 8.8e16  29.89  2.5e5  1.1e9 2.8e14 4.8e16     95   1.05
store_multiplier[9]          2.0    0.13   0.46   1.12   1.66   1.99   2.28   3.04     12    1.6
neighborhood_multiplier[1]  1.01    0.04   0.16    0.8   0.89   0.97   1.08   1.42     13   1.36
neighborhood_multiplier[2]  1.08    0.04   0.16   0.87   0.97   1.05   1.15   1.49     13   1.35
neighborhood_multiplier[3]  1.12    0.05   0.17   0.89   0.99   1.09    1.2   1.55     12   1.38
neighborhood_multiplier[4]  0.98    0.04   0.15   0.77   0.86   0.94   1.06   1.36     13   1.35
neighborhood_multiplier[5]  0.81    0.06   0.21   0.55   0.66   0.76   0.91   1.44     12   1.43
neighborhood_multiplier[6]   1.3e7   1.4e7  5.6e71.4e-212.3e-16 4.1e-3 5662.1  1.6e8     16    1.3
neighborhood_multiplier[7]  1.02    0.04   0.16   0.81   0.91   0.99    1.1   1.43     13   1.35
neighborhood_multiplier[8]  1.15    0.05   0.18    0.9   1.02   1.11   1.24    1.6     12   1.37
neighborhood_multiplier[9]  0.96    0.04   0.16   0.75   0.85   0.93   1.04   1.36     13    1.3
neighborhood_multiplier[10] 0.02    0.02   0.145.0e-179.1e-15 1.7e-8 1.1e-5   0.13     57   1.06
neighborhood_multiplier[11] 1.23    0.05   0.19   0.97   1.09   1.19   1.32    1.7     13   1.37
neighborhood_multiplier[12] 1.17    0.05   0.24   0.76    1.0   1.15   1.31   1.74     22   1.21
error_term                  2.78  2.2e-3   0.04   2.71   2.75   2.78   2.81   2.86    304   1.01
sigma                        inf     nan    inf3.7e3064.6e3079.0e3071.3e3081.8e308    nan    nan
lp__                       -3193    0.62   4.18  -3202  -3195  -3192  -3190  -3185     46   1.07
```

Model Assumptions and methods

This model produces the observed price of a product is a function of the base price of that product and multipliers associated with its the store brand where its sold and the neighborhood of the store. More precisely, the price is modeled as a sample from a normal distribution centered at the value of the product among the base price and the multipliers.

The half-Cauchy distribution was selected as the generative before the base price for the following reasons. First, the prior is set to be broad, inclusive of a variety of currencies and stores. Second, the base price is a positive real number larger than zero. Thus, having the Cauchy distribution with $x = 1.5$ ensures that the values drawn from the distribution are real positive numbers. Third, we specified $y = 2$ to ensure that the model has a broad range that captures the variety of base prices. Finally, the Cauchy distribution has heavy tails, which serve the broad range of the base price we assumed.

For the store brand and the neighborhood multipliers, we used the lognormal distribution to sample the priors for the following reasons. First, by setting the parameters of the lognormal to be 0.5 and 1, we are able to generate a prior distribution centered around 1. Second, the lognormal distribution has a positive and continuous support that also provides an interpretation for multiplicative effects.

For the likelihood function

$$\text{Normal}\left(y_i | x_{f_1(i)} * \lambda^{sb}_{f_2(i)} * \lambda^{nh}_{f_4(i)}, \sigma\right) \tag{1}$$

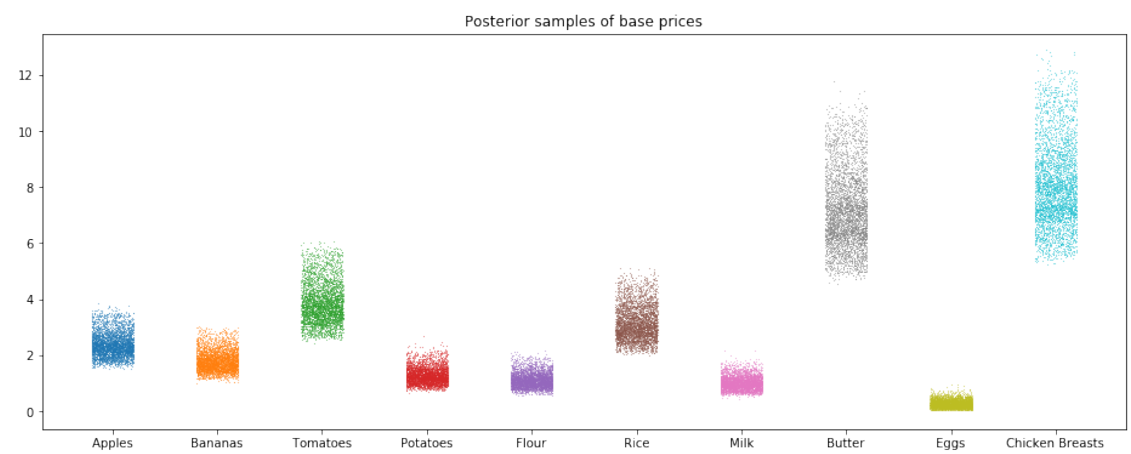
for i in $i \ldots n$ where n is the number of data points. λ^{nh} is the multiplier for the neighborhood and λ^{sb} is the multiplier for the store brand. σ is the error term for the normal distribution. $f(i)$ is the mapping function the matches the encoded values to their original numbers.

We chose the normal likelihood function for the following reasons. First, it captures the fluctuation of produce prices that can be caused by other factors we didn't account for- demand and supply in each area. Second, since the lowest price in the original data set was 1.19, there are fewer chances of having the normal distribution generating negative values.

Questions to answer

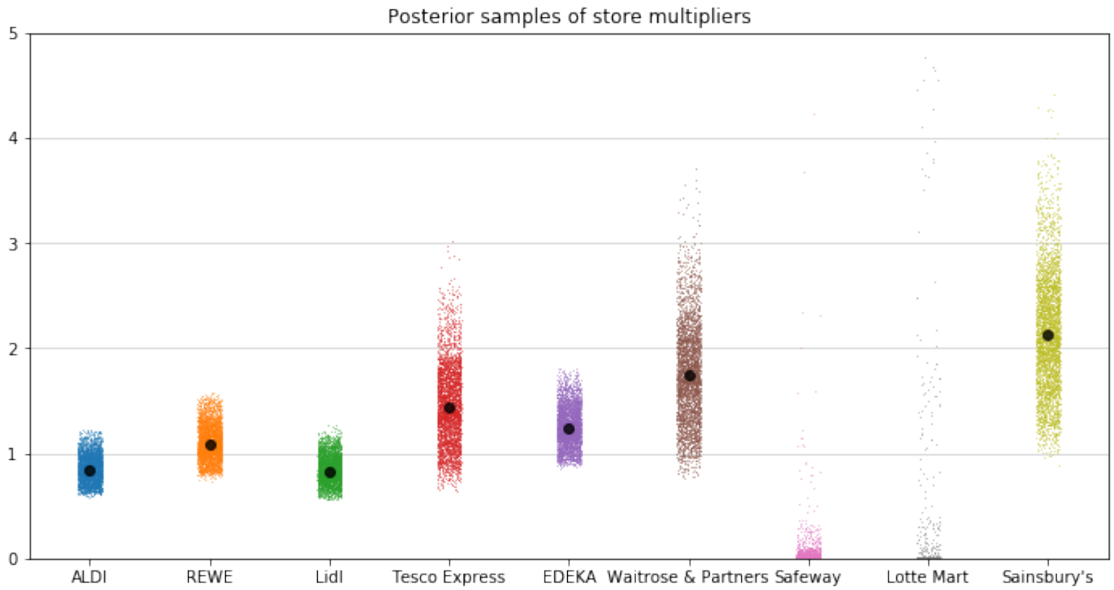
Base price for each product

	Product	Average Price
0	Apples	2.34
1	Bananas	1.77
2	Tomatoes	3.82
3	Potatoes	1.24
4	Flour	1.07
5	Rice	3.10
6	Milk	0.98
7	Butter	7.12
8	Eggs	0.23
9	Chicken Breasts	8.03



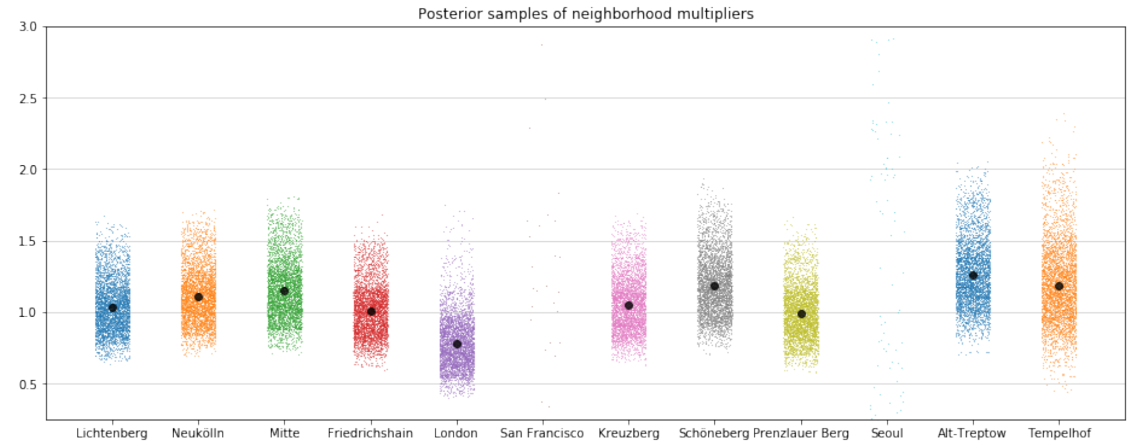
As we can see from the figures above, the base price for each product is the mean of the samples of the posterior we generated. We can see that dairy products, like milk and eggs, are the cheapest-except for the butter. We also can see a wide range of prices for chicken breasts and butter which reflects the wide range of prices. The wide range of base prices is also a result of selecting the Cauchy distribution as our prior.

Store Brand Multiplier



From the observations above, the Lidl has the lowest multiplier.- slightly less than Aldi. Sainsbury's has the highest multiplication factor. We can see that Lotte Mart and Safeway- Seoul and San Francisco entries- have fewer dots and narrower range of their multiplier due to the fact that there are are two data points for both of them.

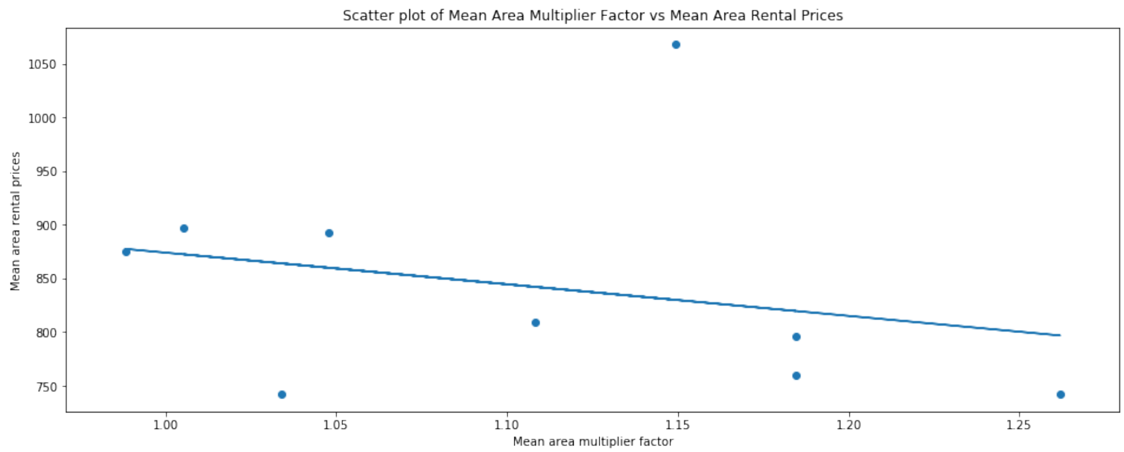
Location Multiplier



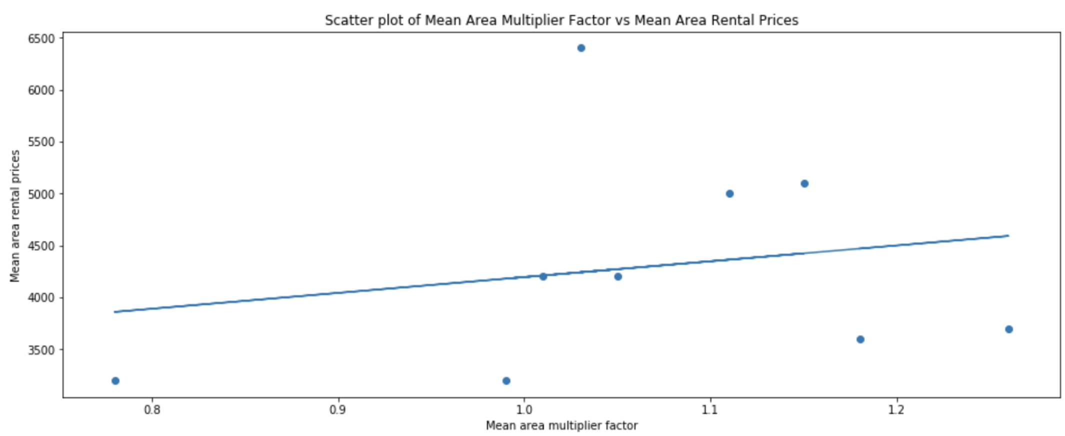
	Neighborhood	Mean Multiplier
0	Lichtenberg	1.03
1	Neukölln	1.11
2	Mitte	1.15
3	Friedrichshain	1.01
4	London	0.78
5	Kreuzberg	1.05
6	Prenzlauer Berg	0.99
7	Alt-Treptow	1.26
8	Tempelhof	1.18

We can see from the figures above, the highest location multiplier is Mitte and the lowest in London. We didn't include San Francisco or Seoul because the multiplier was too small, which is a result of having only 2 data points for both. In the first graph, the multipliers for Seoul and San Francisco were too faint in the graph, which also the result of having two data points.

Correlation



From the above figure, we can see that the rent price and for London and Berlin neighborhood is negatively correlated to each other. The coefficient of this correlation is -0.27 which indicates there is a weak correlation between the multipliers and the rent prices. One explanation to the weak relation is that the presence of other factors that influences the multipliers that we didn't take into account, whether the neighborhood is rural or urban.



When I used the property prices rather than the rent prices, the correlation coefficient became 0.25 which implies there is a weak positive relationship between the property prices and the neighborhood multiplier.

Appendix

I visited each store at 8 and 9 PM on November first.