Question 1

In [ ]:

```
import pandas as pd
import numpy as np

dat = pd.read_excel('ConcessionSalesData_ForClass.xlsx')
dat.head(5)
```

Out[ ]:

	food_game	UserID	UseCount	revenue	game_week	special_discount	special_item	F
0	BAG PEANUTS_Game 1	3304107	1	4.726207	Game 1	STH Discount Only	Yes	
1	BAG PEANUTS_Game 1	3405989	1	4.730000	Game 1	STH Discount Only	Yes	
2	BAG PEANUTS_Game 1	3302989	1	4.730000	Game 1	STH Discount Only	Yes	
3	BAG PEANUTS_Game 1	3253641	1	4.567500	Game 1	STH Discount Only	Yes	
4	BAG PEANUTS_Game 1	3315665	1	4.726615	Game 1	STH Discount Only	Yes	

5 rows × 25 columns

Assumptions

- Customer spending habits differ by game, so we must control for it
- Assume that the average pricepoint for an item in each game is the weighted average of all actual prices, weighted by the demand.
- Assume the occurence of discounts on 1 item does not depend on occurence of discounts of another item

Peanuts

In [ ]:

```
peanuts = dat.loc[dat['MENUITEMNAME'] == 'BAG PEANUTS', :]
peanuts.head(5)
```

Out[ ]:

	food_game	UserID	UseCount	revenue	game_week	special_discount	special_item	F
0	BAG PEANUTS_Game 1	3304107	1	4.726207	Game 1	STH Discount Only	Yes	
1	BAG PEANUTS_Game 1	3405989	1	4.730000	Game 1	STH Discount Only	Yes	
2	BAG PEANUTS_Game 1	3302989	1	4.730000	Game 1	STH Discount Only	Yes	
3	BAG PEANUTS_Game 1	3253641	1	4.567500	Game 1	STH Discount Only	Yes	
4	BAG PEANUTS_Game 1	3315665	1	4.726615	Game 1	STH Discount Only	Yes	

5 rows × 25 columns

In [ ]:

```
peanuts.columns
```

Out[ ]:

```
Index(['food_game', 'UserID', 'UseCount', 'revenue', 'game_week',
       'special_discount', 'special_item', 'FAMILYGROUPNAME', 'Master_Item
',
       'MENUITEMNAME', 'PRICES', 'actual_discount', 'actual_price',
       'Discount Type', 'Discount Percentage', 'first_week_discount',
       'Discount_HotDog', 'Discount_SouvCup', 'Discount_BtlWater',
       'Discount_Peanuts', 'Discount_Nachos', 'Discount_Pretzel',
       'Discount_Popcorn', 'sth_rev_game', 'total_product_rev_nonSTH'],
      dtype='object')
```

In [ ]:

```
# Weight prices according to their demand.
# Prices for CL are siginificantly lower than GA / STH
# However the number of CL is also significantly lower than GA / STH
# We want to weight each actual price by the demand of item at that price po
# This negates the class imbalance issues

weights = peanuts.groupby(by = ['game_week', 'Discount Type'])['UseCount', '
weights['weighted_sums'] = weights['UseCount'] * weights['revenue']
weights['uc2'] = weights['UseCount'] ** 2
weights = weights.groupby(by = ['game_week'])['weighted_sums', 'uc2'].sum().
weights['weighted_actual_price'] = weights['weighted_sums'] / weights['uc2']
demand = peanuts.groupby(by = ['game_week'])['UseCount'].sum(numeric_only=Tr
peanut_demand_price = pd.merge(left = weights, right = demand, on = 'game_we
peanut_demand_price.drop(labels=['weighted_sums', 'uc2'], axis = 1, inplace=
/tmp/ipykernel_415796/2273690423.py:7: FutureWarning: Indexing with multipl
e keys (implicitly converted to a tuple of keys) will be deprecated, use a
list instead.
  weights = peanuts.groupby(by = ['game_week', 'Discount Type'])['UseCount
', 'revenue'].sum(numeric_only=True).reset_index()
/tmp/ipykernel_415796/2273690423.py:10: FutureWarning: Indexing with multip
le keys (implicitly converted to a tuple of keys) will be deprecated, use a
list instead.
  weights = weights.groupby(by = ['game_week'])['weighted_sums', 'uc2'].sum
().reset_index()
```

In [ ]:

```
peanut_demand_price_controlled = pd.merge(left=peanut_demand_price, right=pe
        'Discount_Nachos', 'Discount_Pretzel',
       'Discount_Popcorn']], on = 'game_week', how = 'left').drop_duplicates
```

In [ ]:

```
peanut_demand_price_controlled
```

Out[ ]:

	game_week	weighted_actual_price	UseCount	Discount_HotDog	Discount_SouvCup	Discou
0	Game 1	4.639045	105	Yes	No	
105	Game 2	2.629261	176	No	No	
267	Game 3	4.640434	94	No	Yes	
361	Game 4	4.649035	105	Yes	No	
466	Game 5	4.646859	66	No	Yes	
532	Game 6	4.630899	73	No	No	
605	Game 7	4.651225	41	Yes	Yes	
646	Game 8	4.553264	58	Yes	Yes	

In [ ]:

```
from sklearn.preprocessing import LabelEncoder

le = LabelEncoder()
df_peanut = pd.get_dummies(peanut_demand_price_controlled, columns=['game_we

df_peanut.loc[:, ['Discount_HotDog', 'Discount_SouvCup', 'Discount_BtlWater'
        'Discount_Nachos', 'Discount_Pretzel',
       'Discount_Popcorn']] = df_peanut.loc[:, ['Discount_HotDog', 'Discount
        'Discount_Nachos', 'Discount_Pretzel',
       'Discount_Popcorn']].apply(le.fit_transform)

df_peanut['weighted_actual_price'] = np.log(df_peanut['weighted_actual_price
df_peanut['UseCount'] = np.log(df_peanut['UseCount'])
df_peanut.drop(labels='game_week_Game 1', axis = 1, inplace=True)
```

```
/tmp/ipykernel_415796/877056312.py:6: FutureWarning: In a future version, `
df.iloc[:, i] = newvals` will attempt to set the values inplace instead of
always setting a new array. To retain the old behavior, use either `df[df.c
olumns[i]] = newvals` or, if columns are non-unique, `df.isetitem(i, newval
s)`
  df_peanut.loc[:, ['Discount_HotDog', 'Discount_SouvCup', 'Discount_BtlWat
er',
```

In [ ]:

```
df_peanut
```

Out[ ]:

	weighted_actual_price	UseCount	Discount_HotDog	Discount_SouvCup	Discount_BtlWater
0	1.534509	4.653960	1	0	1
105	0.966703	5.170484	0	0	0
267	1.534808	4.543295	0	1	0
361	1.536660	4.653960	1	0	1
466	1.536191	4.189655	0	1	0
532	1.532751	4.290459	0	0	1
605	1.537131	3.713572	1	1	0
646	1.515844	4.060443	1	1	1

Modeling

In [ ]:

```
import statsmodels.api as sm

X = df_peanut.drop(labels='UseCount', axis = 1)
X = sm.add_constant(X)
y = df_peanut['UseCount']

m_peanut = sm.OLS(y, X).fit()
print('Price elasticity for peanuts is', m_peanut.params[1])
```

```
Price elasticity for peanuts is 1.7426767806748145
```

BAVARIAN PRETZEL

In [ ]:

```
# Extrat item
bav_pret = dat.loc[dat['MENUITEMNAME'] == 'BAVARIAN PRETZEL', :]

# Sum demand and revenue grouped by game_week and discount type
weights = bav_pret.groupby(by = ['game_week', 'Discount Type'])['UseCount',

# Weighted average of price, weighted on demand
weights['weighted_sums'] = weights['UseCount'] * weights['revenue']
weights['uc2'] = weights['UseCount'] ** 2
weights = weights.groupby(by = ['game_week'])['weighted_sums', 'uc2'].sum().
weights['weighted_actual_price'] = weights['weighted_sums'] / weights['uc2']

# Obtain total demand by game
demand = bav_pret.groupby(by = ['game_week'])['UseCount'].sum(numeric_only=T
bav_demand_price = pd.merge(left = weights, right = demand, on = 'game_week'
bav_demand_price.drop(labels=['weighted_sums', 'uc2'], axis = 1, inplace=Tru
/tmp/ipykernel_415796/2360950994.py:5: FutureWarning: Indexing with multipl
e keys (implicitly converted to a tuple of keys) will be deprecated, use a
list instead.
  weights = bav_pret.groupby(by = ['game_week', 'Discount Type'])['UseCount
', 'revenue'].sum(numeric_only=True).reset_index()
/tmp/ipykernel_415796/2360950994.py:10: FutureWarning: Indexing with multip
le keys (implicitly converted to a tuple of keys) will be deprecated, use a
list instead.
  weights = weights.groupby(by = ['game_week'])['weighted_sums', 'uc2'].sum
().reset_index()
```

In [ ]:

```
# Add covariates to df
bav_demand_price_controlled = pd.merge(left=bav_demand_price, right=bav_pret
       'Discount_Peanuts', 'Discount_Nachos',
       'Discount_Popcorn']], on = 'game_week', how = 'left').drop_duplicates

le = LabelEncoder()
df_bav = pd.get_dummies(bav_demand_price_controlled, columns=['game_week'])

# Label encoding
df_bav.loc[:, ['Discount_HotDog', 'Discount_SouvCup', 'Discount_BtlWater',
       'Discount_Peanuts', 'Discount_Nachos',
       'Discount_Popcorn']] = df_bav.loc[:, ['Discount_HotDog', 'Discount_So
       'Discount_Peanuts', 'Discount_Nachos',
       'Discount_Popcorn']].apply(le.fit_transform)

# Take ln
df_bav['weighted_actual_price'] = np.log(df_bav['weighted_actual_price'])
df_bav['UseCount'] = np.log(df_bav['UseCount'])
df_bav.drop(labels='game_week_Game 1', axis = 1, inplace=True)
```

```
/tmp/ipykernel_415796/1266464013.py:10: FutureWarning: In a future version,
`df.iloc[:, i] = newvals` will attempt to set the values inplace instead of
always setting a new array. To retain the old behavior, use either `df[df.c
olumns[i]] = newvals` or, if columns are non-unique, `df.isetitem(i, newval
s)`
  df_bav.loc[:, ['Discount_HotDog', 'Discount_SouvCup', 'Discount_BtlWater
',
```

In [ ]:

```
# Fit linear regression
X = df_bav.drop(labels='UseCount', axis = 1)
X = sm.add_constant(X)
y = df_bav['UseCount']

m_bav = sm.OLS(y, X).fit()
print('Price elasticity for bavarian pretzels is', m_bav.params[1])
```

```
Price elasticity for bavarian pretzels is 1.8180249676100928
```

Nachos

In [ ]:

```
# Extract item
nacho = dat.loc[dat['MENUITEMNAME'] == 'NACHOS', :]

# Sum demand and revenue grouped by game_week and discount type
weights = nacho.groupby(by = ['game_week', 'Discount Type'])['UseCount', 're

# Weighted average of price, weighted on demand
weights['weighted_sums'] = weights['UseCount'] * weights['revenue']
weights['uc2'] = weights['UseCount'] ** 2
weights = weights.groupby(by = ['game_week'])['weighted_sums', 'uc2'].sum().
weights['weighted_actual_price'] = weights['weighted_sums'] / weights['uc2']

# Obtain total demand by game
demand = nacho.groupby(by = ['game_week'])['UseCount'].sum(numeric_only=True
nacho_demand_price = pd.merge(left = weights, right = demand, on = 'game_wee
nacho_demand_price.drop(labels=['weighted_sums', 'uc2'], axis = 1, inplace=T
```

```
/tmp/ipykernel_415796/4031717643.py:5: FutureWarning: Indexing with multipl
e keys (implicitly converted to a tuple of keys) will be deprecated, use a
list instead.
  weights = nacho.groupby(by = ['game_week', 'Discount Type'])['UseCount',
'revenue'].sum(numeric_only=True).reset_index()
/tmp/ipykernel_415796/4031717643.py:10: FutureWarning: Indexing with multip
le keys (implicitly converted to a tuple of keys) will be deprecated, use a
list instead.
  weights = weights.groupby(by = ['game_week'])['weighted_sums', 'uc2'].sum
().reset_index()
```

In [ ]:

```
# Add covariates to df
nacho_demand_price_controlled = pd.merge(left=nacho_demand_price, right=nach
       'Discount_Peanuts',  'Discount_Pretzel',
       'Discount_Popcorn']], on = 'game_week', how = 'left').drop_duplicates

# Label encoding
le = LabelEncoder()
df_nacho = pd.get_dummies(nacho_demand_price_controlled, columns=['game_week

df_nacho.loc[:, ['Discount_HotDog', 'Discount_SouvCup', 'Discount_BtlWater',
       'Discount_Peanuts', 'Discount_Pretzel',
       'Discount_Popcorn']] = df_nacho.loc[:, ['Discount_HotDog', 'Discount_
       'Discount_Peanuts', 'Discount_Pretzel',
       'Discount_Popcorn']].apply(le.fit_transform)

# Take ln
df_nacho['weighted_actual_price'] = np.log(df_nacho['weighted_actual_price']
df_nacho['UseCount'] = np.log(df_nacho['UseCount'])
df_nacho.drop(labels='game_week_Game 1', axis = 1, inplace=True)
```

```
/tmp/ipykernel_415796/607551027.py:10: FutureWarning: In a future version,
`df.iloc[:, i] = newvals` will attempt to set the values inplace instead of
always setting a new array. To retain the old behavior, use either `df[df.c
olumns[i]] = newvals` or, if columns are non-unique, `df.isetitem(i, newval
s)`
  df_nacho.loc[:, ['Discount_HotDog', 'Discount_SouvCup', 'Discount_BtlWate
r',
```

In [ ]:

```
# Fit linear regression
X = df_nacho.drop(labels='UseCount', axis = 1)
X = sm.add_constant(X)
y = df_nacho['UseCount']

m_nacho = sm.OLS(y, X).fit()
print('Price elasticity for nachos is', m_nacho.params[1])
```

```
Price elasticity for nachos is 1.489932164132497
```

Souv Pop

In [ ]:

```
# Extract item
souv_pop = dat.loc[dat['MENUITEMNAME'] == 'SOUV POPCORN', :]

# Sum demand and revenue grouped by game_week and discount type
weights = souv_pop.groupby(by = ['game_week', 'Discount Type'])['UseCount',

# Weighted average of price, weighted on demand
weights['weighted_sums'] = weights['UseCount'] * weights['revenue']
weights['uc2'] = weights['UseCount'] ** 2
weights = weights.groupby(by = ['game_week'])['weighted_sums', 'uc2'].sum().
weights['weighted_actual_price'] = weights['weighted_sums'] / weights['uc2']

# Obtain total demand by game
demand = souv_pop.groupby(by = ['game_week'])['UseCount'].sum(numeric_only=T
souv_pop_demand_price = pd.merge(left = weights, right = demand, on = 'game_
souv_pop_demand_price.drop(labels=['weighted_sums', 'uc2'], axis = 1, inplac
```

```
/tmp/ipykernel_415796/1977475684.py:5: FutureWarning: Indexing with multipl
e keys (implicitly converted to a tuple of keys) will be deprecated, use a
list instead.
  weights = souv_pop.groupby(by = ['game_week', 'Discount Type'])['UseCount
', 'revenue'].sum(numeric_only=True).reset_index()
/tmp/ipykernel_415796/1977475684.py:10: FutureWarning: Indexing with multip
le keys (implicitly converted to a tuple of keys) will be deprecated, use a
list instead.
  weights = weights.groupby(by = ['game_week'])['weighted_sums', 'uc2'].sum
().reset_index()
```

In [ ]:

```
# Add covariates to df
souv_pop_demand_price_controlled = pd.merge(left=souv_pop_demand_price, righ
       'Discount_Peanuts', 'Discount_Nachos', 'Discount_Pretzel']], on = 'ga

# Label Encoding
le = LabelEncoder()
df_souv_pop = pd.get_dummies(souv_pop_demand_price_controlled, columns=['gam

df_souv_pop.loc[:, ['Discount_HotDog', 'Discount_SouvCup', 'Discount_BtlWate
       'Discount_Peanuts', 'Discount_Nachos', 'Discount_Pretzel']] = df_souv
       'Discount_Peanuts', 'Discount_Nachos', 'Discount_Pretzel']].apply(le.

# Take ln
df_souv_pop['weighted_actual_price'] = np.log(df_souv_pop['weighted_actual_p
df_souv_pop['UseCount'] = np.log(df_souv_pop['UseCount'])
df_souv_pop.drop(labels='game_week_Game 1', axis = 1, inplace=True)
```

```
/tmp/ipykernel_415796/1415636204.py:9: FutureWarning: In a future version,
`df.iloc[:, i] = newvals` will attempt to set the values inplace instead of
always setting a new array. To retain the old behavior, use either `df[df.c
olumns[i]] = newvals` or, if columns are non-unique, `df.isetitem(i, newval
s)`
  df_souv_pop.loc[:, ['Discount_HotDog', 'Discount_SouvCup', 'Discount_BtlW
ater',
```

In [ ]:

```
# Fit linear regression
X = df_souv_pop.drop(labels='UseCount', axis = 1)
X = sm.add_constant(X)
y = df_souv_pop['UseCount']

m_souv_pop = sm.OLS(y, X).fit()
print('Price elasticity for souvenir popcorn is', m_souv_pop.params[1])
```

```
Price elasticity for souvenir popcorn is 0.7648721684375243
```

Hot Dog

In [ ]:

```
# Extract Item
hotdog = dat.loc[dat['MENUITEMNAME'] == 'HOT DOG', :]

# Sum demand and revenue grouped by game_week and discount type
weights = hotdog.groupby(by = ['game_week', 'Discount Type'])['UseCount', 'r

# Weighted average of price, weighted on demand
weights['weighted_sums'] = weights['UseCount'] * weights['revenue']
weights['uc2'] = weights['UseCount'] ** 2
weights = weights.groupby(by = ['game_week'])['weighted_sums', 'uc2'].sum().
weights['weighted_actual_price'] = weights['weighted_sums'] / weights['uc2']

# Obtain demand by week
demand = hotdog.groupby(by = ['game_week'])['UseCount'].sum(numeric_only=Tru
hotdog_demand_price = pd.merge(left = weights, right = demand, on = 'game_we
hotdog_demand_price.drop(labels=['weighted_sums', 'uc2'], axis = 1, inplace=
/tmp/ipykernel_415796/3905450397.py:5: FutureWarning: Indexing with multipl
e keys (implicitly converted to a tuple of keys) will be deprecated, use a
list instead.
  weights = hotdog.groupby(by = ['game_week', 'Discount Type'])['UseCount',
'revenue'].sum(numeric_only=True).reset_index()
/tmp/ipykernel_415796/3905450397.py:10: FutureWarning: Indexing with multip
le keys (implicitly converted to a tuple of keys) will be deprecated, use a
list instead.
  weights = weights.groupby(by = ['game_week'])['weighted_sums', 'uc2'].sum
().reset_index()
```

In [ ]:

```
# Add covariates to df
hotdog_demand_price_controlled = pd.merge(left=hotdog_demand_price, right=ho
       'Discount_Peanuts', 'Discount_Nachos', 'Discount_Pretzel',
       'Discount_Popcorn']], on = 'game_week', how = 'left').drop_duplicates

# Label encoding
le = LabelEncoder()
df_hotdog = pd.get_dummies(hotdog_demand_price_controlled, columns=['game_we

df_hotdog.loc[:, [ 'Discount_SouvCup', 'Discount_BtlWater',
       'Discount_Peanuts', 'Discount_Nachos', 'Discount_Pretzel',
       'Discount_Popcorn']] = df_hotdog.loc[:, ['Discount_SouvCup', 'Discoun
       'Discount_Peanuts', 'Discount_Nachos', 'Discount_Pretzel',
       'Discount_Popcorn']].apply(le.fit_transform)
# Take ln
df_hotdog['weighted_actual_price'] = np.log(df_hotdog['weighted_actual_price
df_hotdog['UseCount'] = np.log(df_hotdog['UseCount'])
df_hotdog.drop(labels='game_week_Game 1', axis = 1, inplace=True)
```

```
/tmp/ipykernel_415796/278722298.py:10: FutureWarning: In a future version,
`df.iloc[:, i] = newvals` will attempt to set the values inplace instead of
always setting a new array. To retain the old behavior, use either `df[df.c
olumns[i]] = newvals` or, if columns are non-unique, `df.isetitem(i, newval
s)`
  df_hotdog.loc[:, [ 'Discount_SouvCup', 'Discount_BtlWater',
```

In [ ]:

```
# Fit linear regression
X = df_hotdog.drop(labels='UseCount', axis = 1)
X = sm.add_constant(X)
y = df_hotdog['UseCount']

m_hotdog = sm.OLS(y, X).fit()
print('Price elasticity for hot dog is', m_hotdog.params[1])
```

```
Price elasticity for hot dog is 1.9188957353279497
```

Bottled Water (non 1L)

In [ ]:

```
# Extract item
btlwater = dat.loc[dat['MENUITEMNAME'] == 'BTL DEJA BLUE', :]

# Sum of demand and revenue grouped by game week and discount type
weights = btlwater.groupby(by = ['game_week', 'Discount Type'])['UseCount',

# Weighted average of price, weighted on demand
weights['weighted_sums'] = weights['UseCount'] * weights['revenue']
weights['uc2'] = weights['UseCount'] ** 2
weights = weights.groupby(by = ['game_week'])['weighted_sums', 'uc2'].sum().
weights['weighted_actual_price'] = weights['weighted_sums'] / weights['uc2']

# Obtain total demand for item by week
demand = btlwater.groupby(by = ['game_week'])['UseCount'].sum(numeric_only=T
btlwater_demand_price = pd.merge(left = weights, right = demand, on = 'game_
btlwater_demand_price.drop(labels=['weighted_sums', 'uc2'], axis = 1, inplac
```

```
/tmp/ipykernel_415796/1872982700.py:5: FutureWarning: Indexing with multipl
e keys (implicitly converted to a tuple of keys) will be deprecated, use a
list instead.
  weights = btlwater.groupby(by = ['game_week', 'Discount Type'])['UseCount
', 'revenue'].sum(numeric_only=True).reset_index()
/tmp/ipykernel_415796/1872982700.py:10: FutureWarning: Indexing with multip
le keys (implicitly converted to a tuple of keys) will be deprecated, use a
list instead.
  weights = weights.groupby(by = ['game_week'])['weighted_sums', 'uc2'].sum
().reset_index()
```

In [ ]:

```
# Add covariates to df
btlwater_demand_price_controlled = pd.merge(left=btlwater_demand_price, righ
       'Discount_Peanuts', 'Discount_Nachos', 'Discount_Pretzel',
       'Discount_Popcorn']], on = 'game_week', how = 'left').drop_duplicates

# Label encoding
le = LabelEncoder()
df_btlwater = pd.get_dummies(btlwater_demand_price_controlled, columns=['gam

df_btlwater.loc[:, ['Discount_HotDog', 'Discount_SouvCup',
       'Discount_Peanuts', 'Discount_Nachos', 'Discount_Pretzel',
       'Discount_Popcorn']] = df_btlwater.loc[:, ['Discount_HotDog', 'Discou
       'Discount_Peanuts', 'Discount_Nachos', 'Discount_Pretzel',
       'Discount_Popcorn']].apply(le.fit_transform)

# Take ln
df_btlwater['weighted_actual_price'] = np.log(df_btlwater['weighted_actual_p
df_btlwater['UseCount'] = np.log(df_btlwater['UseCount'])
df_btlwater.drop(labels='game_week_Game 1', axis = 1, inplace=True)
```

```
/tmp/ipykernel_415796/1239130973.py:10: FutureWarning: In a future version,
`df.iloc[:, i] = newvals` will attempt to set the values inplace instead of
always setting a new array. To retain the old behavior, use either `df[df.c
olumns[i]] = newvals` or, if columns are non-unique, `df.isetitem(i, newval
s)`
  df_btlwater.loc[:, ['Discount_HotDog', 'Discount_SouvCup',
```

In [ ]:

```
# Fit linear regression
X = df_btlwater.drop(labels='UseCount', axis = 1)
X = sm.add_constant(X)
y = df_btlwater['UseCount']

m_btlwater = sm.OLS(y, X).fit()
print('Price elasticity for bottled water is', m_btlwater.params[1])
```

```
Price elasticity for bottled water is 2.025598624588172
```

Souvenir Soda (32 oz)

In [ ]:

```
# Extract item
souv_soda = dat.loc[dat['MENUITEMNAME'] == 'SOUV CUP 32', :]

# Obtain total revenue and demand by game week and discount type
weights = souv_soda.groupby(by = ['game_week', 'Discount Type'])['UseCount',

# Weighted average of price, weighted on demand
weights['weighted_sums'] = weights['UseCount'] * weights['revenue']
weights['uc2'] = weights['UseCount'] ** 2
weights = weights.groupby(by = ['game_week'])['weighted_sums', 'uc2'].sum().
weights['weighted_actual_price'] = weights['weighted_sums'] / weights['uc2']

# Obtain total demand of item by game_week
demand = souv_soda.groupby(by = ['game_week'])['UseCount'].sum(numeric_only=
souv_soda_demand_price = pd.merge(left = weights, right = demand, on = 'game
souv_soda_demand_price.drop(labels=['weighted_sums', 'uc2'], axis = 1, inpla
```

```
/tmp/ipykernel_415796/1155200150.py:5: FutureWarning: Indexing with multipl
e keys (implicitly converted to a tuple of keys) will be deprecated, use a
list instead.
  weights = souv_soda.groupby(by = ['game_week', 'Discount Type'])['UseCoun
t', 'revenue'].sum(numeric_only=True).reset_index()
/tmp/ipykernel_415796/1155200150.py:10: FutureWarning: Indexing with multip
le keys (implicitly converted to a tuple of keys) will be deprecated, use a
list instead.
  weights = weights.groupby(by = ['game_week'])['weighted_sums', 'uc2'].sum
().reset_index()
```

In [ ]:

```
# Add covariates to df
souv_soda_demand_price_controlled = pd.merge(left=souv_soda_demand_price, ri
       'Discount_Peanuts', 'Discount_Nachos', 'Discount_Pretzel',
       'Discount_Popcorn']], on = 'game_week', how = 'left').drop_duplicates

# Label encoding
le = LabelEncoder()
df_souv_soda = pd.get_dummies(souv_soda_demand_price_controlled, columns=['g

df_souv_soda.loc[:, ['Discount_HotDog',  'Discount_BtlWater',
       'Discount_Peanuts', 'Discount_Nachos', 'Discount_Pretzel',
       'Discount_Popcorn']] = df_souv_soda.loc[:, ['Discount_HotDog', 'Disco
       'Discount_Peanuts', 'Discount_Nachos', 'Discount_Pretzel',
       'Discount_Popcorn']].apply(le.fit_transform)

# Take ln
df_souv_soda['weighted_actual_price'] = np.log(df_souv_soda['weighted_actual
df_souv_soda['UseCount'] = np.log(df_souv_soda['UseCount'])
df_souv_soda.drop(labels='game_week_Game 1', axis = 1, inplace=True)
```

```
/tmp/ipykernel_415796/3384735821.py:10: FutureWarning: In a future version,
`df.iloc[:, i] = newvals` will attempt to set the values inplace instead of
always setting a new array. To retain the old behavior, use either `df[df.c
olumns[i]] = newvals` or, if columns are non-unique, `df.isetitem(i, newval
s)`
  df_souv_soda.loc[:, ['Discount_HotDog',  'Discount_BtlWater',
```

In [ ]:

```
# Fit linear regression
X = df_souv_soda.drop(labels='UseCount', axis = 1)
X = sm.add_constant(X)
y = df_souv_soda['UseCount']

m_souv_soda = sm.OLS(y, X).fit()
print('Price elasticity for soda is', m_souv_soda.params[1])
```

```
Price elasticity for soda is 1.9469140350087892
```

Question 2

Note

Here, we fit a new model for every item to find relationship the between demand of that item and game_week, price, and discount on other items. For every dataframe, the column variable is the variable in question, and the effect column is the effect of that variable (raw unit changes), if true, on demand.

Nachos

In [ ]:

```
import statsmodels.api as sm
import statsmodels.formula.api as smf

formula = 'UseCount ~ C(game_week) + weighted_actual_price + C(Discount_HotD
m_nacho2 = smf.ols(formula = formula, data = nacho_demand_price_controlled).
effects_nacho = pd.DataFrame(m_nacho2.params[1:len(m_nacho2.params)-1]).rese
effects_nacho.columns = ['Variable', 'Effect on Demand']
effects_nacho
```

Out[ ]:

	Variable	Effect on Demand
0	C(game_week)[T.Game 2]	74.519596
1	C(game_week)[T.Game 3]	9.626436
2	C(game_week)[T.Game 4]	-22.095237
3	C(game_week)[T.Game 5]	-14.515390
4	C(game_week)[T.Game 6]	-25.000582
5	C(game_week)[T.Game 7]	-33.279772
6	C(game_week)[T.Game 8]	9.467838
7	C(Discount_HotDog)[T.Yes]	-10.442865
8	C(Discount_SouvCup)[T.Yes]	-28.700888
9	C(Discount_BtlWater)[T.Yes]	-2.163676
10	C(Discount_Popcorn)[T.Yes]	-5.047552
11	C(Discount_Pretzel)[T.Yes]	-15.374146
12	C(Discount_Peanuts)[T.Yes]	74.519596

Souvenir Popcorn

In [ ]:

```
formula = 'UseCount ~ C(game_week) + weighted_actual_price + C(Discount_HotD
m_souv_pop2 = smf.ols(formula = formula, data = souv_pop_demand_price_contro
effects_souv_pop = pd.DataFrame(m_souv_pop2.params[1:len(m_souv_pop2.params)
effects_souv_pop.columns = ['Variable', 'Effect on Demand']
effects_souv_pop
```

Out[ ]:

	Variable	Effect on Demand
0	C(game_week)[T.Game 2]	-4.792903
1	C(game_week)[T.Game 3]	-30.691604
2	C(game_week)[T.Game 4]	5.813987
3	C(game_week)[T.Game 5]	93.910922
4	C(game_week)[T.Game 6]	7.990612
5	C(game_week)[T.Game 7]	-30.021992
6	C(game_week)[T.Game 8]	13.824035
7	C(Discount_HotDog)[T.Yes]	-23.075755
8	C(Discount_SouvCup)[T.Yes]	47.021362
9	C(Discount_BtlWater)[T.Yes]	14.936849
10	C(Discount_Nachos)[T.Yes]	-4.792903
11	C(Discount_Pretzel)[T.Yes]	-22.700991
12	C(Discount_Peanuts)[T.Yes]	-4.792903

Hot Dog

In [ ]:

```
formula = 'UseCount ~ C(game_week) + weighted_actual_price + C(Discount_Popc
m_hotdog2 = smf.ols(formula = formula, data = hotdog_demand_price_controlled
effects_hotdog = pd.DataFrame(m_hotdog2.params[1:len(m_hotdog2.params)-1]).r
effects_hotdog.columns = ['Variable', 'Effect on Demand']
effects_hotdog
```

Out[ ]:

	Variable	Effect on Demand
0	C(game_week)[T.Game 2]	-128.392499
1	C(game_week)[T.Game 3]	110.044617
2	C(game_week)[T.Game 4]	-202.003958
3	C(game_week)[T.Game 5]	-133.671479
4	C(game_week)[T.Game 6]	-519.778052
5	C(game_week)[T.Game 7]	38.204405
6	C(game_week)[T.Game 8]	-46.186503
7	C(Discount_Popcorn)[T.Yes]	-179.857983
8	C(Discount_SouvCup)[T.Yes]	-31.608961
9	C(Discount_BtlWater)[T.Yes]	570.597788
10	C(Discount_Nachos)[T.Yes]	-128.392499
11	C(Discount_Pretzel)[T.Yes]	-409.733435
12	C(Discount_Peanuts)[T.Yes]	-128.392499

Peanuts

In [ ]:
```
formula = 'UseCount ~ C(game_week) + weighted_actual_price + C(Discount_HotD
m_peanut2 = smf.ols(formula = formula, data = peanut_demand_price_controlled
effects_peanut = pd.DataFrame(m_peanut2.params[1:len(m_peanut2.params)-1]).r
effects_peanut.columns = ['Variable', 'Effect on Demand']
effects_peanut
```

Out[ ]:

	Variable	Effect on Demand
0	C(game_week)[T.Game 2]	50.309474
1	C(game_week)[T.Game 3]	19.392785
2	C(game_week)[T.Game 4]	-0.188302
3	C(game_week)[T.Game 5]	-6.922128
4	C(game_week)[T.Game 6]	-29.400296
5	C(game_week)[T.Game 7]	-36.256773
6	C(game_week)[T.Game 8]	-4.891581
7	C(Discount_HotDog)[T.Yes]	-7.561361
8	C(Discount_SouvCup)[T.Yes]	-28.677696
9	C(Discount_BtlWater)[T.Yes]	-0.704883
10	C(Discount_Nachos)[T.Yes]	50.309474
11	C(Discount_Pretzel)[T.Yes]	-10.007511
12	C(Discount_Popcorn)[T.Yes]	-11.813708

Pretzel

In [ ]:
```
formula = 'UseCount ~ C(game_week) + weighted_actual_price + C(Discount_HotD
m_bav2 = smf.ols(formula = formula, data = bav_demand_price_controlled).fit(
effects_bav = pd.DataFrame(m_bav2.params[1:len(m_bav2.params)-1]).reset_inde
effects_bav.columns = ['Variable', 'Effect on Demand']
effects_bav
```

Out[ ]:

	Variable	Effect on Demand
0	C(game_week)[T.Game 2]	-53.578025
1	C(game_week)[T.Game 3]	531.762066
2	C(game_week)[T.Game 4]	4.976135
3	C(game_week)[T.Game 5]	-134.999177
4	C(game_week)[T.Game 6]	169.758036
5	C(game_week)[T.Game 7]	-169.992466
6	C(game_week)[T.Game 8]	-48.236774
7	C(Discount_HotDog)[T.Yes]	-212.212683
8	C(Discount_SouvCup)[T.Yes]	178.533649
9	C(Discount_BtlWater)[T.Yes]	127.537819
10	C(Discount_Nachos)[T.Yes]	-53.578025
11	C(Discount_Peanuts)[T.Yes]	-53.578025
12	C(Discount_Popcorn)[T.Yes]	-183.235951

Bottled Water

In [ ]:

```
formula = 'UseCount ~ C(game_week) + weighted_actual_price + C(Discount_HotD
m_btlwater2 = smf.ols(formula = formula, data = btlwater_demand_price_contro
effects_btlwater = pd.DataFrame(m_btlwater2.params[1:len(m_btlwater2.params)
effects_btlwater.columns = ['Variable', 'Effect on Demand']
effects_btlwater
```

Out[ ]:

	Variable	Effect on Demand
0	C(game_week)[T.Game 2]	-140.525011
1	C(game_week)[T.Game 3]	30.201447
2	C(game_week)[T.Game 4]	-53.382367
3	C(game_week)[T.Game 5]	31.538811
4	C(game_week)[T.Game 6]	83.046860
5	C(game_week)[T.Game 7]	-492.767066
6	C(game_week)[T.Game 8]	-48.132004
7	C(Discount_HotDog)[T.Yes]	434.343530
8	C(Discount_SouvCup)[T.Yes]	-479.158811
9	C(Discount_Pretzel)[T.Yes]	113.248306
10	C(Discount_Nachos)[T.Yes]	-140.525011
11	C(Discount_Peanuts)[T.Yes]	-140.525011
12	C(Discount_Popcorn)[T.Yes]	-16.593193

Souvenir Soda

In [ ]:

```
formula = 'UseCount ~ C(game_week) + weighted_actual_price + C(Discount_HotD
m_souv_soda2 = smf.ols(formula = formula, data = souv_soda_demand_price_cont
effects_souv_soda = pd.DataFrame(m_souv_soda2.params[1:len(m_souv_soda2.para
effects_souv_soda.columns = ['Variable', 'Effect on Demand']
effects_souv_soda
```

Out[ ]:

	Variable	Effect on Demand
0	C(game_week)[T.Game 2]	-36.981968
1	C(game_week)[T.Game 3]	392.859795
2	C(game_week)[T.Game 4]	60.685310
3	C(game_week)[T.Game 5]	7.453560
4	C(game_week)[T.Game 6]	-165.846892
5	C(game_week)[T.Game 7]	28.803508
6	C(game_week)[T.Game 8]	182.965578
7	C(Discount_HotDog)[T.Yes]	64.069190
8	C(Discount_Pretzel)[T.Yes]	227.012903
9	C(Discount_BtlWater)[T.Yes]	-130.581209
10	C(Discount_Nachos)[T.Yes]	-36.981968
11	C(Discount_Peanuts)[T.Yes]	-36.981968
12	C(Discount_Popcorn)[T.Yes]	190.419138

Question 3

From question 1, we can obtain the following

In [ ]:

```
names = ['Nachos', 'Souv Popcorn', "Hot Dog", "Peanuts", "Pretzels", "Bottle
elastic = pd.DataFrame({'item': names, 'elasticity': [m_nacho.params[1], m_s
elastic
```

Out[ ]:

	item	elasticity
0	Nachos	1.489932
1	Souv Popcorn	0.764872
2	Hot Dog	1.918896
3	Peanuts	1.742677
4	Pretzels	1.818025
5	Bottled Water	2.025599
6	Souv Soda 32oz	1.946914

Souvenir popcorn has a price elasticity of demand lower than 1, which makes it an inelastic good. This means that, after controlling for the game week and existence of discounts of other items, souvenir popcorn seem to be "necessities" without substitutes to the consumer. However, we must note that this analysis is conducted by analyzing the price change vs demand change over the 8 game weeks. What this means is that, holding all else equal, souvenir popcorn is, on average, an inelastic goods; the price changes over the 8 weeks did not cause the demand to change significantly.

With this information, the Bears can be more flexible in their pricing of souvenir popcorn, while not increasing prices of the others so much. However, because we added game week as a control, the Bears can also look at how game week affects the elasticity of goods.

Also, with the information presented in question 2, the Bears can look at how the existence of discounts on other items affects the demand of one item, as well as how the game week affects the demand of an item. By combining these 2 pieces of information, the Bears can develop a better pricing strategy by optimizing the co-occurence of discounts to maximize demand. Another possible use of this information is that, the Bears can optimize inventory by looking at how the demand varies by game week.

Question 4

Weakness

- The demand and price is split by game week. By doing so, we are essentially looking at 1 single price point (albiet weighted) for every game week. This means that we are assuming the price varies by game, and thus price affects demand.
- Because of splitting by game week, we only have 8 data points to build a linear regression on and to find the price elasticity. With so few data points, it is hard to find a good estimate of the actual value of the coefficient itself.
- Another weakness is that there are too few sources of variation. For instance, only a tiny fraction of customers are club-level with 20% discount. It is difficult to gauge demand for that 20% discount price point.

Solution

- Since the discounts are already only redeemable on the app with QR code, the Bears could send out random discounts to the app holders of varying percentages to gather more data on how demand varies at different price points.
- The Bears should also include the sales of the non STH or CL customers to see if no discount changes the demand. This also adds more data by incorporating demands at original price point.
- For each game, the Bears should also include who they’re playing against. The attendance of the games can vary depending on the excitement of the game, which is likely dictated by the Bears’ opponent of a given game. Attendance can also heavily affect the demand for food items, and should thus be added as a control variable.