Data Cleaning

Features

source_id = producer code —> only got 1 —> useless

hs_code = list of numbers used by customs to classify a product

commodity_desc = product name

geography_code = location code

geography_desc = location name (paired with location code)

attribute_desc = export quantity or export value

unit_desc = KG, $ or L

year_id = year

timeperiod_id = month

amount = amount transacted

In [1]:

```
import pandas as pd
import numpy as np
import regex as re
import seaborn as sns
import matplotlib.pyplot as plt
```

In [2]:

```
df_product = pd.read_csv('catfish_trout.csv')
```

In [3]:
```
def catfish_filter(val):
    catfish_stat = re.search(r'C*c*at',val)
    if catfish_stat:
        return True
    else:
        return False

#df_filtered = df[df['col'].apply(regex_filter)]
```

In [4]:
```
df_filtered = df_product[df_product['commodity_desc'].apply(catfish_filter)]
```

In [5]:
```
df_filtered = df_filtered[df_filtered['geography_desc'] == 'United States of America']
```

In [6]:
```
df_filtered.drop(columns = ['source_id', 'geography_desc', 'geography_code'], inplace=True)
df_filtered = df_filtered[df_filtered['attribute_desc'].isin(['Farm Sales to Processors', 'Farm Price', 'Producer
```

In [7]:
```
df_sales = df_filtered[df_filtered['attribute_desc'].isin(['Farm Sales to Processors'])]
df_price = df_filtered[df_filtered['attribute_desc'].isin(['Farm Price'])]
df_inventory = df_filtered[df_filtered['attribute_desc'].isin(['Producer Inventories'])]
```

In [8]:
```
df_price.rename(columns={'amount': 'price'}, inplace = True)
```

```
/Users/salimwid/opt/anaconda3/lib/python3.8/site-packages/pandas/core/frame.py:4441: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#return
ing-a-view-versus-a-copy
  return super().rename(
```

In [9]:
```
df_price.drop(columns=['attribute_desc','unit_desc'], inplace= True)
```

```
/Users/salimwid/opt/anaconda3/lib/python3.8/site-packages/pandas/core/frame.py:4308: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#return
ing-a-view-versus-a-copy
  return super().drop(
```

In [10]:
```
df_sales.drop(columns=['attribute_desc', 'unit_desc'], inplace=True)
```

In [11]:
```
df_sales_his = df_sales
df_sales_his = df_sales_his.merge(df_price, on = ['year_id', 'timeperiod_id', 'commodity_desc'])
df_sales = df_sales.merge(df_price, on = ['year_id', 'timeperiod_id', 'commodity_desc'])
df_sales = df_sales[df_sales['year_id'] != 2013]
```

In [12]:
```
year_value_list = pd.pivot_table(df_sales, values=['amount', 'price'], index=['year_id'], aggfunc={'amount': [np.s
```

In [13]:
```
year_dict = {'amount_agg':[], 'amount_mean': [],'price_mean': [], 'price_std': []}
```

In [14]:
```
for i in year_value_list:
    year_dict['amount_mean'].append(i[0])
    year_dict['amount_agg'].append(i[1])
    year_dict['price_mean'].append(i[2])
    year_dict['price_std'].append(i[3])
```

In [15]:
```
df_plot = pd.DataFrame.from_dict(year_dict)
```

In [16]:
```
df_plot['year'] = df_sales['year_id'].unique()
df_plot.set_index('year', inplace=True)
```

In [17]:
```
df_plot1 = df_plot[['price_mean', 'price_std']]
df_plot2 = df_plot[['amount_agg', 'amount_mean']]
```

In [18]:
```
sns.set(rc={'figure.figsize':(11.7,8.27)})
sns.lineplot(data = df_plot1, legend=False)
plt.vlines(x=2004, color='black', linestyle='--', ymin = 0, ymax = 125)
plt.legend(title='Metrics', loc='upper right', labels=['Catfish Price Mean', 'Catfish Price Standard Deviation'])
```

Out[18]: <matplotlib.legend.Legend at 0x7fef97f264f0>

In [19]:

```
sns.set(rc={'figure.figsize':(11.7,8.27)})
sns.lineplot(data = df_plot2, legend=False)
plt.vlines(x=2004, color='black', linestyle='--', ymin = 0, ymax = 600000)
plt.legend(title='Metrics', loc='upper right', labels=['Total Catfish Sold by U.S Producer (Locally)', 'Mean of Ca
```

Out[19]: <matplotlib.legend.Legend at 0x7fef98a75f40>

In [20]:

```
pd.pivot_table(df_sales, values=['amount', 'price'], index=['year_id'], aggfunc={'amount': [np.sum,np.mean], 'pric
```

Out[20]:

amount	price

	mean	sum	mean	std
year_id				
1986	17813.000000	213756.0	66.833333	4.745013
1987	23374.666667	280496.0	61.833333	3.214550
1988	24592.416667	295109.0	76.416667	3.604501
1989	28491.666667	341900.0	71.666667	5.613836
1990	30036.250000	360435.0	75.750000	2.179449
1991	32572.500000	390870.0	63.083333	5.583390
1992	38113.916667	457367.0	59.833333	3.069893
1993	38251.083333	459013.0	71.000000	3.074824
1994	36605.750000	439269.0	78.416667	1.975225
1995	37240.500000	446886.0	78.583333	0.514929
1996	39343.583333	472123.0	77.250000	1.815339
1997	43745.750000	524949.0	71.166667	1.850471
1998	47029.583333	564355.0	74.166667	3.737606
1999	49719.000000	596628.0	73.675000	2.637190
2000	49466.916667	593603.0	74.975000	3.918749
2001	49759.000000	597108.0	64.516667	5.313761
2002	52550.083333	630601.0	56.766667	1.479148
2003	55125.333333	661504.0	58.116667	3.410634
2004	52537.500000	630450.0	69.608333	1.866186
2005	50055.833333	600670.0	72.516667	0.348590

2006	47177.583333	566131.0	79.625000	4.146658
2007	41353.833333	496246.0	76.658333	7.723689
2008	42466.416667	509597.0	77.975000	5.789823
2009	38841.666667	466100.0	77.075000	1.296236
2010	39306.916667	471683.0	80.233333	2.990085
2011	27845.250000	334143.0	117.700000	11.669774
2012	25012.583333	300151.0	97.550000	18.793785

In [21]:
```
df_sales.set_index(['year_id', 'timeperiod_id'], inplace=True)
df_sales.reset_index(inplace=True)
```

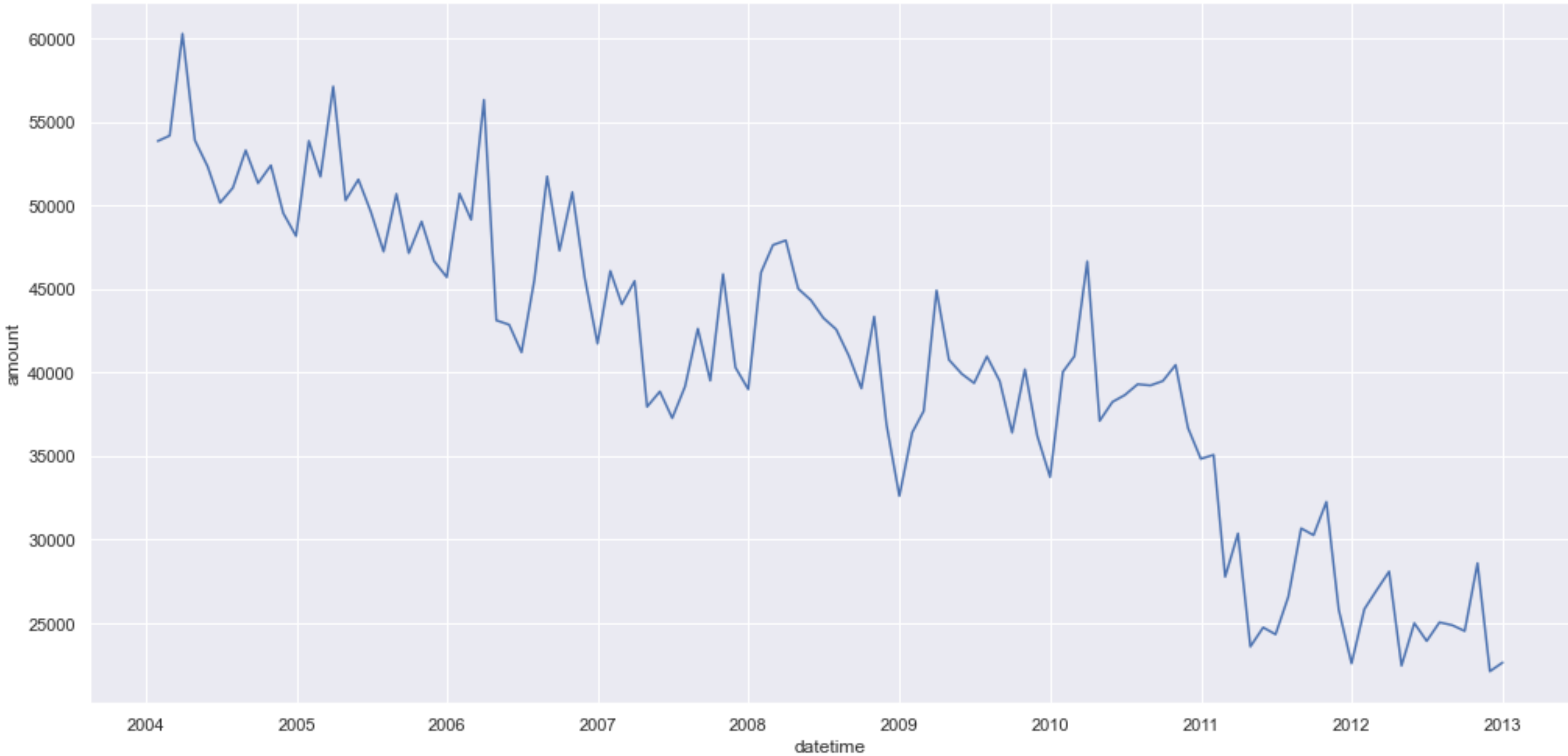
In [22]:
```
df_sales = df_sales[df_sales['year_id'] >= 2004]
```

In [23]:
```
df_sales['datetime'] = pd.date_range(start='1/1/2004', periods=108, freq='M')
```

In [24]:
```
df_sales.reset_index(inplace=True)
df_sales.set_index('datetime', inplace=True)
```

In [25]:
```
sns.set(rc={'figure.figsize':(17,8.27)})
sns.lineplot(data=df_sales, x='datetime', y='amount')
```

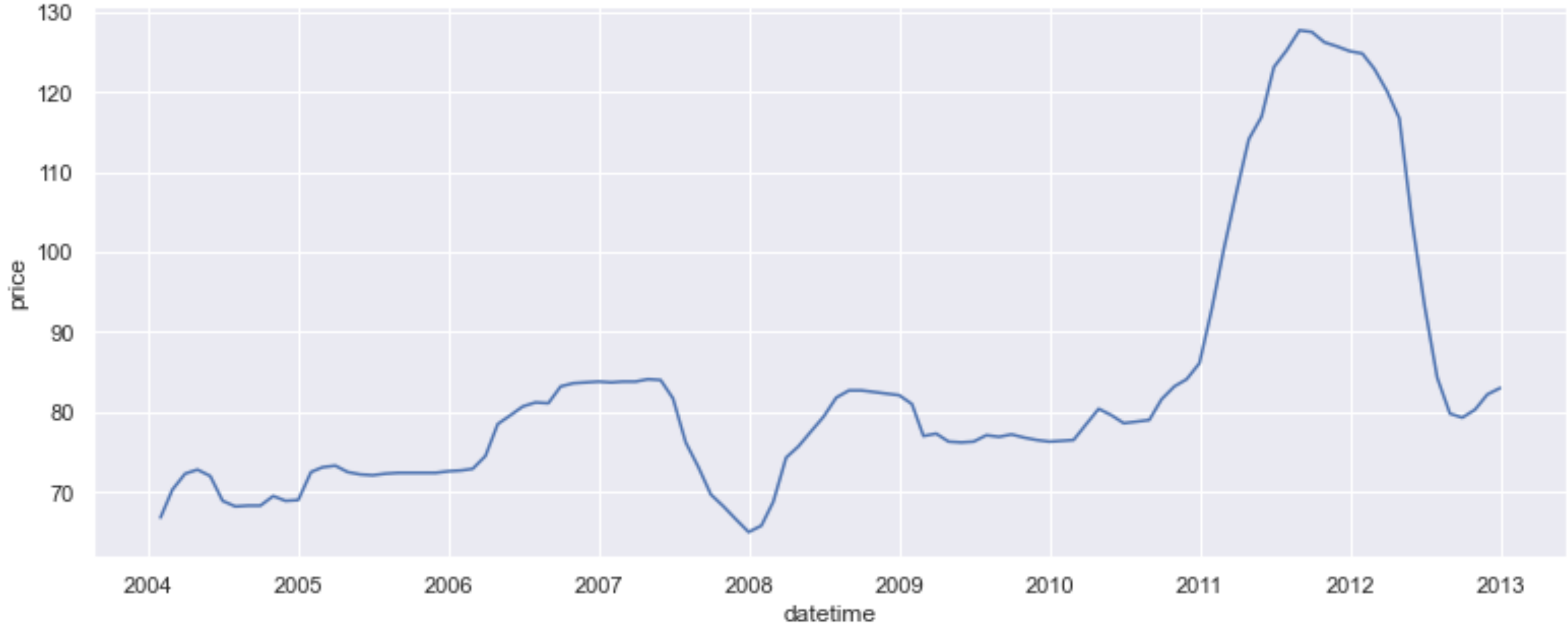
Out[25]: <AxesSubplot:xlabel='datetime', ylabel='amount'>



In [26]:

```
sns.set(rc={'figure.figsize':(13,5)})
sns.lineplot(data=df_sales, x='datetime', y='price')
```

Out[26]: <AxesSubplot:xlabel='datetime', ylabel='price'>



In [27]:

```
min_val = df_sales[['year_id', 'price']].groupby('year_id').min().values
max_val = df_sales[['year_id', 'price']].groupby('year_id').max().values
```

In [28]:

```
def calculate_benefit_val(df, min_max, colname, year, col_selected):
    for i in min_max:
        year+= 1
        df_sales.loc[df_sales['year_id'] == year, colname] = df_sales[col_selected] - i[0]
    return None

calculate_benefit_val(df_sales, min_val, 'benefit_criterion', 2003, 'price')
```

In [29]:
```
def calculate_criterion_val(df, min_max, colname, year, col_selected):
    for i in min_max:
        year+= 1
        df_sales.loc[df_sales['year_id'] == year, colname] = i[0] - df_sales[col_selected]
    return None

calculate_criterion_val(df_sales, max_val, 'regret_criterion', 2003, 'price')
```

In [30]:
```
df_sales['benefit_criterion'].values.reshape(9,12)
```

Out[30]:
```
array([[ 0. ,  3.5,  5.5,  6. ,  5.2,  2.1,  1.4,  1.5,  1.5,  2.7,  2.1,
         2.2],
       [ 0.4,  1. ,  1.2,  0.4,  0.1,  0. ,  0.2,  0.3,  0.3,  0.3,  0.3,
         0.5],
       [ 0. ,  0.2,  1.8,  5.8,  6.9,  8. ,  8.5,  8.4, 10.5, 10.9, 11. ,
        11.1],
       [18.7, 18.8, 18.8, 19.1, 19. , 16.7, 11.2,  8.1,  4.7,  3.2,  1.6,
         0. ],
       [ 0. ,  3. ,  8.5,  9.9, 11.8, 13.6, 16. , 16.9, 16.9, 16.7, 16.5,
        16.3],
       [ 4.8,  0.8,  1.1,  0.1,  0. ,  0.1,  0.9,  0.7,  1. ,  0.6,  0.3,
         0.1],
       [ 0. ,  0.1,  2.1,  4. ,  3.2,  2.2,  2.4,  2.6,  5.2,  6.8,  7.7,
         9.7],
       [ 0. ,  7.2, 14.4, 21. , 23.8, 30. , 32.1, 34.6, 34.4, 33.1, 32.6,
        32. ],
       [45.5, 43.6, 40.8, 37.4, 24.5, 14.1,  5. ,  0.5,  0. ,  1. ,  2.9,
         3.7]])
```

In [31]:
```
df_sales.describe()
```

Out[31]:

	index	year_id	timeperiod_id	amount	price	benefit_criterion	regret_criterion
count	108.00000	108.000000	108.000000	108.000000	108.000000	108.000000	108.000000
mean	269.50000	2008.000000	6.500000	40510.842593	83.215741	9.060185	7.484259
std	31.32092	2.594026	3.468146	9410.871487	16.361741	11.196887	10.887781
min	216.00000	2004.000000	1.000000	22124.000000	65.000000	0.000000	0.000000
25%	242.75000	2006.000000	3.750000	35927.250000	72.675000	0.775000	0.875000
50%	269.50000	2008.000000	6.500000	40867.000000	78.700000	4.350000	3.900000
75%	296.25000	2010.000000	9.250000	47380.500000	83.700000	14.175000	7.950000
max	323.00000	2012.000000	12.000000	60272.000000	127.700000	45.500000	45.500000

In [32]:

```
df_sales['price'].values.reshape(9,12)
```

Out[32]:

```
array([[ 66.8,  70.3,  72.3,  72.8,  72. ,  68.9,  68.2,  68.3,  68.3,
         69.5,  68.9,  69. ],
       [ 72.5,  73.1,  73.3,  72.5,  72.2,  72.1,  72.3,  72.4,  72.4,
         72.4,  72.4,  72.6],
       [ 72.7,  72.9,  74.5,  78.5,  79.6,  80.7,  81.2,  81.1,  83.2,
         83.6,  83.7,  83.8],
       [ 83.7,  83.8,  83.8,  84.1,  84. ,  81.7,  76.2,  73.1,  69.7,
         68.2,  66.6,  65. ],
       [ 65.8,  68.8,  74.3,  75.7,  77.6,  79.4,  81.8,  82.7,  82.7,
         82.5,  82.3,  82.1],
       [ 81. ,  77. ,  77.3,  76.3,  76.2,  76.3,  77.1,  76.9,  77.2,
         76.8,  76.5,  76.3],
       [ 76.4,  76.5,  78.5,  80.4,  79.6,  78.6,  78.8,  79. ,  81.6,
         83.2,  84.1,  86.1],
       [ 93.1, 100.3, 107.5, 114.1, 116.9, 123.1, 125.2, 127.7, 127.5,
        126.2, 125.7, 125.1],
       [124.8, 122.9, 120.1, 116.7, 103.8,  93.4,  84.3,  79.8,  79.3,
         80.3,  82.2,  83. ]])
```

In [33]:
```
df_sales['regret_criterion'].values.reshape(9,12)
```

Out[33]:
```
array([[ 6. ,  2.5,  0.5,  0. ,  0.8,  3.9,  4.6,  4.5,  4.5,  3.3,  3.9,
         3.8],
       [ 0.8,  0.2,  0. ,  0.8,  1.1,  1.2,  1. ,  0.9,  0.9,  0.9,  0.9,
         0.7],
       [11.1, 10.9,  9.3,  5.3,  4.2,  3.1,  2.6,  2.7,  0.6,  0.2,  0.1,
         0. ],
       [ 0.4,  0.3,  0.3,  0. ,  0.1,  2.4,  7.9, 11. , 14.4, 15.9, 17.5,
        19.1],
       [16.9, 13.9,  8.4,  7. ,  5.1,  3.3,  0.9,  0. ,  0. ,  0.2,  0.4,
         0.6],
       [ 0. ,  4. ,  3.7,  4.7,  4.8,  4.7,  3.9,  4.1,  3.8,  4.2,  4.5,
         4.7],
       [ 9.7,  9.6,  7.6,  5.7,  6.5,  7.5,  7.3,  7.1,  4.5,  2.9,  2. ,
         0. ],
       [34.6, 27.4, 20.2, 13.6, 10.8,  4.6,  2.5,  0. ,  0.2,  1.5,  2. ,
         2.6],
       [ 0. ,  1.9,  4.7,  8.1, 21. , 31.4, 40.5, 45. , 45.5, 44.5, 42.6,
        41.8]])
```

In [34]:
```
df_sales.to_csv('df_final_first_3.csv')
```

In [35]:
```
str = '2000 1.101 1.127 1.354 1.407 1.399 1.321 1.378 1.400 1.336 1.308 1.217 1.245 2001 1.140 1.098 1.018 1.002 1
```

In [36]:
```
split_list = str.split(' ')
```

In [37]:

```
year_dict = {}
current_year = 2000
for val in split_list:
    int_val = float(val)
    if int_val >= current_year:
        year_dict[int_val] = []
        current_year = int_val
    else:
        year_dict[current_year].append(int_val)
```

In [38]:

```
float_list = []
for i in split_list:
    float_val = float(i)
    if float_val < 1999:
        float_list.append(float(i))
```

In [39]:

```
np.array(float_list).reshape(8,12)
```

Out[39]:

```
array([[1.101, 1.127, 1.354, 1.407, 1.399, 1.321, 1.378, 1.4  , 1.336,
        1.308, 1.217, 1.245],
       [1.14 , 1.098, 1.018, 1.002, 1.028, 1.108, 1.062, 0.997, 0.924,
        0.87 , 0.953, 0.98 ],
       [1.04 , 1.118, 1.245, 1.293, 1.284, 1.164, 1.032, 1.097, 1.041,
        1.002, 1.041, 1.077],
       [0.912, 0.945, 1.018, 1.325, 1.623, 1.71 , 1.512, 1.538, 1.507,
        1.369, 1.344, 1.182],
       [1.189, 1.279, 1.602, 1.733, 1.565, 1.24 , 1.391, 1.915, 1.632,
        1.542, 1.374, 1.126],
       [1.313, 1.234, 1.429, 1.484, 1.632, 1.751, 2.105, 2.108, 1.742,
        1.394, 1.084, 1.385],
       [1.164, 1.418, 1.776, 1.867, 1.349, 1.132, 1.138, 1.473, 1.595,
        1.517, 1.49 , 1.366],
       [1.483, 1.962, 1.995, 2.327, 2.574, 2.49 , 2.49 , 2.455, 2.618,
        2.449, 1.857, 1.601]])
```

In [40]:
```
df_inventory[df_inventory['year_id'] > 2003]
```

Out[40]:

	commodity_desc	attribute_desc	unit_desc	year_id	timeperiod_id	amount
1966	Catfish-Broodfish	Producer Inventories	1,000 EA	2004	17	1113.0
1967	Catfish-Broodfish	Producer Inventories	1,000 EA	2005	17	1053.0
1968	Catfish-Broodfish	Producer Inventories	1,000 EA	2006	17	1091.0
1969	Catfish-Broodfish	Producer Inventories	1,000 EA	2007	17	886.0
1970	Catfish-Broodfish	Producer Inventories	1,000 EA	2008	17	801.0
...	...	...	...	...	...	...
2099	Catfish-Large Food-size	Producer Inventories	1,000 EA	2012	17	3595.0
2100	Catfish-Large Food-size	Producer Inventories	1,000 EA	2013	17	5155.0
2101	Catfish-Large Food-size	Producer Inventories	1,000 EA	2014	17	4500.0
2102	Catfish-Large Food-size	Producer Inventories	1,000 EA	2015	17	5090.0
2103	Catfish-Large Food-size	Producer Inventories	1,000 EA	2016	17	3520.0

78 rows × 6 columns

In [41]:
```
mean_year = df_sales_his[['year_id','price']].groupby('year_id').mean().values
```

In [42]:
```
df_sales_his['datetime'] = pd.date_range(start='1/1/1986', periods=326, freq='M')
```

In [43]:
```
initial_year = 1986
for i in mean_year:
    df_sales_his.loc[df_sales_his['year_id'] == initial_year, 'percent_diff'] = (df_sales_his['price'] - i[0])/i[0
    initial_year+=1
```

In [61]:
```
min_val_his = df_sales_his[['timeperiod_id', 'percent_diff']].groupby('timeperiod_id').min().values
max_val_his = df_sales_his[['timeperiod_id', 'percent_diff']].groupby('timeperiod_id').max().values
```

In [45]:
```
for i in range(12):
    df_sales.loc[df_sales['timeperiod_id'] == i+1, 'lower_bound'] = df_sales['price'] * (1 + min_val_his[i])
    df_sales.loc[df_sales['timeperiod_id'] == i+1, 'upper_bound'] = df_sales['price'] * (1 + max_val_his[i])
```

In [46]:
```
min_val_lower = df_sales[['year_id', 'lower_bound']].groupby('year_id').min().values
max_val_lower = df_sales[['year_id', 'lower_bound']].groupby('year_id').max().values

min_val_upper = df_sales[['year_id', 'upper_bound']].groupby('year_id').min().values
max_val_upper = df_sales[['year_id', 'upper_bound']].groupby('year_id').max().values
```

In [47]:
```
calculate_benefit_val(df_sales, min_val_lower, 'benefit_criterion_lower', 2003, 'lower_bound')
calculate_benefit_val(df_sales, min_val_upper, 'benefit_criterion_upper', 2003, 'upper_bound')

calculate_criterion_val(df_sales, max_val_lower, 'regret_criterion_lower', 2003, 'lower_bound')
calculate_criterion_val(df_sales, max_val_upper, 'regret_criterion_upper', 2003, 'upper_bound')
```

In [48]:
```
df_sales
```

Out[48]:

	index	year_id	timeperiod_id	commodity_desc	amount	price	benefit_criterion	regret_criterion	lower_bound	upper_bound	b
datetime											
2004-01-31	216	2004	1	Catfish	53849.0	66.8	0.0	6.0	52.838403	85.460174	
2004-02-29	217	2004	2	Catfish	54173.0	70.3	3.5	2.5	59.907307	88.568631	
2004-03-31	218	2004	3	Catfish	60272.0	72.3	5.5	0.5	66.034410	89.013121	
2004-04-30	219	2004	4	Catfish	53896.0	72.8	6.0	0.0	70.573322	87.091338	
2004-05-31	220	2004	5	Catfish	52324.0	72.0	5.2	0.8	70.665213	80.345013	
...	...	...	...	...	...	...	...	...	...	...	
2012-08-31	319	2012	8	Catfish	24886.0	79.8	0.5	45.0	65.279754	86.579949	
2012-09-30	320	2012	9	Catfish	24535.0	79.3	0.0	45.5	64.464275	85.902719	
2012-10-31	321	2012	10	Catfish	28596.0	80.3	1.0	44.5	66.100359	86.099065	
2012-11-30	322	2012	11	Catfish	22124.0	82.2	2.9	42.6	69.265402	87.787086	
2012-12-31	323	2012	12	Catfish	22653.0	83.0	3.7	41.8	69.733157	89.831374	

108 rows × 14 columns

In [49]:
```
lower_matrix = df_sales['lower_bound'].values.reshape(9,12)
benefit_lower_matrix = df_sales['benefit_criterion_lower'].values.reshape(9,12)
regret_lower_matrix = df_sales['regret_criterion_lower'].values.reshape(9,12)
```

In [50]:
```
upper_matrix = df_sales['upper_bound'].values.reshape(9,12)
benefit_upper_matrix = df_sales['benefit_criterion_upper'].values.reshape(9,12)
regret_upper_matrix = df_sales['regret_criterion_upper'].values.reshape(9,12)
```

In [ ]:
```

```

In [52]:
```
df_sales.to_csv('df_model_matrix.csv')
```

In [62]:
```
print(min_val_his)
print(max_val_his)
```

```
[[-0.20900595]
 [-0.14783347]
 [-0.086661  ]
 [-0.03058624]
 [-0.01853871]
 [-0.04254229]
 [-0.13582778]
 [-0.18195797]
 [-0.18708355]
 [-0.17683239]
 [-0.1573552 ]
 [-0.15984148]]
[[0.27934393]
 [0.25986674]
 [0.23116351]
 [0.19630958]
 [0.11590296]
 [0.06576802]
 [0.06372133]
 [0.08496177]
 [0.08326253]
 [0.0722175 ]
 [0.06796941]
 [0.08230571]]
```

In [ ]:

Game Theory Models:

Wald (Pessimistic), Laplace, Hurwicz, Benefit, Wald (Optimistic)

In [1]:

```
from gurobipy import *
import numpy as np
import pandas as pd
```

In [2]:

```
# Read dataset
df = pd.read_csv('df_final_first_3.csv')
data = pd.read_csv('/Users/hpone/Desktop/NUS MSBA/DBA5103/Term project/Mansi code/dba5103_gp-widya/df_model_matrix
data.head()
```

Out[2]:

	datetime	index	year_id	timeperiod_id	commodity_desc	amount	price	benefit_criterion	regret_criterion	lower_bound	upper_bound
0	2004-01-31	216	2004	1	Catfish	53849.0	66.8	0.0	6.0	52.838403	85.460174
1	2004-02-29	217	2004	2	Catfish	54173.0	70.3	3.5	2.5	59.907307	88.568631
2	2004-03-31	218	2004	3	Catfish	60272.0	72.3	5.5	0.5	66.034410	89.013121
3	2004-04-30	219	2004	4	Catfish	53896.0	72.8	6.0	0.0	70.573322	87.091338
4	2004-05-31	220	2004	5	Catfish	52324.0	72.0	5.2	0.8	70.665213	80.345013

In [3]:
```
# initial all criterion matrices

# game or neutral matrix
wald_matrix = df['price'].values.reshape(9,12)
# pessistic/optimistic matrix ~ lower/upper bounds respectively
pessimistic_matrix = data['lower_bound'].values.reshape(9,12)
optimistc_matrix = data['upper_bound'].values.reshape(9,12)
benefit_matrix = df['benefit_criterion'].values.reshape(9,12)
regret_matrix = df['regret_criterion'].values.reshape(9,12)

alpha = 0.80
hurwicz_matrix = optimistc_matrix*alpha + (1-alpha)*pessimistic_matrix
laplace_matrix = 0.5*optimistc_matrix + 0.5*pessimistic_matrix

# Number of years: M; No. of months: N = 12
M, N = wald_matrix.shape

month_dict = {0:"Jan", 1:"Feb", 2:"Mar", 3:"Apr", 4:"May", 5:"Jun", 6:"Jul",7:"Aug",8:"Sept",9:"Oct",10:"Nov",11:"

# monthly mean for all years combined
monthly_mean = [np.mean(wald_matrix[:,i]) for i in range(N)]
monthly_mean
```

Out[3]:
```
[81.86666666666666,
 82.84444444444445,
 84.62222222222222,
 85.67777777777778,
 84.65555555555554,
 83.8,
 82.78888888888888,
 82.33333333333333,
 82.43333333333334,
 82.52222222222223,
 82.4888888888889,
 82.55555555555556]
```

In [4]:

```
# Setup Criterion Based Linear Programming Optimization Model
def model_setup(name, matrix):
    # initialize criterion model
    model = Model(f"{name} Criterion")

    # Decision Variables for percentage of catfish sells every month
    p = model.addVars(N)
    # Decision Variable for Price per unit of catfish ~ cents/pounds
    Z = model.addVar(name = 'Z')

    # Set objective to maximize Price per unit
    model.setObjective(Z, GRB.MAXIMIZE)

    for i in range(M):
        # Contraints for Sells every year to be greater than the optimized result
        model.addConstr(quicksum(matrix[i, j]*p[j] for j in range(N)) >= Z, 'Contraints')
        # percentages for every year add up to 1
        model.addConstr (quicksum(p[j] for j in range(N)) == 1)

    model.optimize()

    return model
```

Pessimistic Wald Criterion Model Optimization

In [5]:
```
# Pessimistic Matrix with Wald Criterion Model Optimization
model_name = "Pessimistic Wald"
pessimistic_wald_model =  model_setup(model_name, pessimistic_matrix)

#  Print optimal sells for every month
print("\n Optimal solution:")
price = 0
for i, v in enumerate(pessimistic_wald_model.getVars()[:N]):
    print(v.VarName, v.x)

# Optimal Price given by model
pessimistic_price = round(pessimistic_wald_model.objVal, 3)
print('{} Criterion Z objective => Price: {} cents/pound'.format(model_name, round(pessimistic_wald_model.objVal,
```

```
Academic license - for non-commercial use only - expires 2021-12-22
Using license file /Users/hpone/gurobi.lic
Gurobi Optimizer version 9.1.2 build v9.1.2rc0 (mac64)
Thread count: 2 physical cores, 4 logical processors, using up to 4 threads
Optimize a model with 18 rows, 13 columns and 225 nonzeros
Model fingerprint: 0x3656bf2e
Coefficient statistics:
  Matrix range     [1e+00, 1e+02]
  Objective range  [1e+00, 1e+00]
  Bounds range     [0e+00, 0e+00]
  RHS range        [1e+00, 1e+00]
Presolve removed 8 rows and 0 columns
Presolve time: 0.01s
Presolved: 10 rows, 13 columns, 129 nonzeros

Iteration    Objective       Primal Inf.    Dual Inf.      Time
       0    8.0698122e+02   8.350033e+02   0.000000e+00      0s
       3    7.0665213e+01   0.000000e+00   0.000000e+00      0s

Solved in 3 iterations and 0.03 seconds
Optimal objective  7.066521265e+01

 Optimal solution:
C0 0.0
C1 0.0
C2 0.0
C3 0.0
C4 1.0
C5 0.0
C6 0.0
C7 0.0
C8 0.0
C9 0.0
C10 0.0
C11 0.0
Pessimistic Wald Criterion Z objective => Price: 70.665 cents/pound
```

Pessimistic Wald Criterion Optimal Solution:

p4 = 1 and Maximum Z = 70.665

The solution indicates that out of the total catfish supplied to middlemen, 100% should be sold in the month of May. Thus the guaranteed average price received by the catfish producers(farmers) will be 70.665 cents/pound

Laplace Criterion

In [6]:

```
# Laplace Criterion Model Optimization
model_name = "Laplace"
laplace_model =  model_setup(model_name, laplace_matrix)

#  Print optimal sells for every month
print("\n Optimal solution:")
for i, v in enumerate(laplace_model.getVars()[:N]):
    print(v.VarName, v.x)

# Optimal Price given by model
laplace_price = round(laplace_model.objVal, 3)
print('{} Criterion Z objective => Price : {} cents/pound'.format(model_name, round(laplace_model.objVal, 3)))
```

```
Gurobi Optimizer version 9.1.2 build v9.1.2rc0 (mac64)
Thread count: 2 physical cores, 4 logical processors, using up to 4 threads
Optimize a model with 18 rows, 13 columns and 225 nonzeros
Model fingerprint: 0xb96521ad
Coefficient statistics:
  Matrix range     [1e+00, 1e+02]
  Objective range  [1e+00, 1e+00]
  Bounds range     [0e+00, 0e+00]
  RHS range        [1e+00, 1e+00]
Presolve removed 8 rows and 0 columns
Presolve time: 0.02s
Presolved: 10 rows, 13 columns, 129 nonzeros

Iteration    Objective       Primal Inf.    Dual Inf.      Time
       0    9.2858272e+02   8.993544e+02   0.000000e+00      0s
       4    7.8528060e+01   0.000000e+00   0.000000e+00      0s

Solved in 4 iterations and 0.03 seconds
Optimal objective  7.852806012e+01

 Optimal solution:
C0 0.0
C1 0.0
C2 0.23252181968706165
C3 0.7674781803129384
C4 0.0
C5 0.0
C6 0.0
C7 0.0
C8 0.0
C9 0.0
C10 0.0
C11 0.0
Laplace Criterion Z objective => Price : 78.528 cents/pound
```

Laplace Criterion Optimal Solution:

**p2 = 0.023 and p3=0.77

0.023 *Monthly_mean_for_March + 0.767* Monthly_mean_for_April = 78.528 cents/pound

The optimal solution indicates that out of the total catfish supplied to middlemen, 2.3 and 76.7 percent of catfish should be sold in the months of March and April respectively. Thus the guaranteed average price received by the catfish producers(farmers) will be 78.528 cents/pound

Hurwicz

In [7]:

```
# Hurwicz Criterion Model Optimization
model_name = "Hurwicz"
hurwicz_model =  model_setup(model_name, hurwicz_matrix)

#  Print optimal sells for every month
print("\n Optimal solution:")
for i, v in enumerate(hurwicz_model.getVars()[:N]):
    print(v.VarName, v.x)

# Optimal Price given by model
hurwicz_price = round(hurwicz_model.objVal, 3)
print('{} Criterion Z objective => Price : {} cents/pound'.format(model_name, round(hurwicz_model.objVal, 3)))
```

```
Gurobi Optimizer version 9.1.2 build v9.1.2rc0 (mac64)
Thread count: 2 physical cores, 4 logical processors, using up to 4 threads
Optimize a model with 18 rows, 13 columns and 225 nonzeros
Model fingerprint: 0x9da72ce5
Coefficient statistics:
  Matrix range     [1e+00, 1e+02]
  Objective range  [1e+00, 1e+00]
  Bounds range     [0e+00, 0e+00]
  RHS range        [1e+00, 1e+00]
Presolve removed 8 rows and 0 columns
Presolve time: 0.04s
Presolved: 10 rows, 13 columns, 129 nonzeros

Iteration    Objective       Primal Inf.    Dual Inf.      Time
       0    1.0015436e+03   9.088044e+02   0.000000e+00      0s
       6    8.4417379e+01   0.000000e+00   0.000000e+00      0s

Solved in 6 iterations and 0.05 seconds
Optimal objective  8.441737908e+01

 Optimal solution:
C0 0.0
C1 0.0
C2 1.0
C3 0.0
C4 0.0
C5 0.0
C6 0.0
C7 0.0
C8 0.0
C9 0.0
C10 0.0
C11 0.0
Hurwicz Criterion Z objective => Price : 84.417 cents/pound
```

Hurwicz Criterion Optimal Solution:

p2 = 1.0

1 * Monthly_mean_for_March = 84.417 cents/pound

The optimal solution indicates that out of the total catfish supplied to middlemen, 100 percent of catfish should be sold in the month of March. Thus the guaranteed average price received by the catfish producers(farmers) will be 84.417 cents/pound

Benefit Criterion

In [8]:

```
# Benefit Criterion Model Optimization
model_name = 'Benefit'
benefit_model =  model_setup("Benefit", benefit_matrix)

#  Print optimal sells for every month
benefit_price = 0
for i, v in enumerate(benefit_model.getVars()[:N]):
    if v.x > 0:
        benefit_price = benefit_price + monthly_mean[i]*v.x
    print(v.VarName, v.x)

# Optimal Price given by model
print('{} Criterion Z objective : {}'.format(model_name, round(benefit_model.objVal, 3)))
print("Price given by Benefit Criterion : : {} cents/pound".format(round(benefit_price,3)))
```

```
Gurobi Optimizer version 9.1.2 build v9.1.2rc0 (mac64)
Thread count: 2 physical cores, 4 logical processors, using up to 4 threads
Optimize a model with 18 rows, 13 columns and 216 nonzeros
Model fingerprint: 0x36faad9b
Coefficient statistics:
  Matrix range     [1e-01, 5e+01]
  Objective range  [1e+00, 1e+00]
  Bounds range     [0e+00, 0e+00]
  RHS range        [1e+00, 1e+00]
Presolve removed 8 rows and 0 columns
Presolve time: 0.02s
Presolved: 10 rows, 13 columns, 120 nonzeros

Iteration    Objective       Primal Inf.    Dual Inf.      Time
       0    1.0511500e+01   1.446725e+01   0.000000e+00      0s
       4    1.1822222e+00   0.000000e+00   0.000000e+00      0s

Solved in 4 iterations and 0.02 seconds
Optimal objective  1.182222222e+00
C0 0.022222222222224117
C1 0.0
C2 0.9777777777777759
C3 0.0
C4 0.0
C5 0.0
C6 0.0
C7 0.0
C8 0.0
C9 0.0
C10 0.0
C11 0.0
Benefit Criterion Z objective : 1.182
Price given by Benefit Criterion : : 84.561 cents/pound
```

Benefit Wald Criterion Optimal Solution:

p0 = 0.022 and p2=0.978 Maximum Z = 1.182

0.022 *Monthly_mean_for_January + 0.978* Monthly_mean_for_March = 84.561 cents/pound

The optimal solution indicates that out of the total catfish supplied to middlemen, 2.2 and 97.8 percent of catfish should be sold in the months of January and March respectively. Thus the guaranteed average price received by the catfish producers(farmers) will be 84.561 cents/pound

Optimistic Wald

In [9]:

```
# Optimistic Matrix with Wald Criterion Model Optimization
model_name = "Optimistic Wald"
optimistc_wald_model =  model_setup(model_name, optimistc_matrix)

#  Print optimal sells for every month
print("\n Optimal solution:")
for i, v in enumerate(optimistc_wald_model.getVars()[:N]):
    print(v.VarName, v.x)

# Optimal Price given by model
optimistic_price = round(optimistc_wald_model.objVal, 3)
print('{} Criterion Z objective => Price : {} cents/pound'.format(model_name, round(optimistc_wald_model.objVal, 3
```

```
Gurobi Optimizer version 9.1.2 build v9.1.2rc0 (mac64)
Thread count: 2 physical cores, 4 logical processors, using up to 4 threads
Optimize a model with 18 rows, 13 columns and 225 nonzeros
Model fingerprint: 0x1d346ea2
Coefficient statistics:
  Matrix range     [1e+00, 2e+02]
  Objective range  [1e+00, 1e+00]
  Bounds range     [0e+00, 0e+00]
  RHS range        [1e+00, 1e+00]
Presolve removed 8 rows and 0 columns
Presolve time: 0.01s
Presolved: 10 rows, 13 columns, 129 nonzeros

Iteration    Objective       Primal Inf.    Dual Inf.      Time
       0    1.0501842e+03   9.497797e+02   0.000000e+00      0s
       3    8.9013121e+01   0.000000e+00   0.000000e+00      0s

Solved in 3 iterations and 0.02 seconds
Optimal objective  8.901312148e+01

 Optimal solution:
C0 0.0
C1 0.0
C2 1.0
C3 0.0
C4 0.0
C5 0.0
C6 0.0
C7 0.0
C8 0.0
C9 0.0
C10 0.0
C11 0.0
Optimistic Wald Criterion Z objective => Price : 89.013 cents/pound
```

Optimistic Wald Criterion Optimal Solution:

p2 = 1 and Maximum Z = 89.013

The solution indicates that out of the total catfish supplied to middlemen, 100% should be sold in the month of March. Thus the guaranteed averaGe price received by the catfish producers(farmers) will be 89.013 cents/pound

Consolidating Results

In [10]:

```
results_dict = {
    "pessimistic": pessimistic_price,
    "laplace":laplace_price,
    "hurwicz":hurwicz_price,
    "benefit":benefit_price,
    "optimistic":optimistic_price,
}
```

In [11]:

```
results_dict
```

Out[11]:

```
{'pessimistic': 70.665,
 'laplace': 78.528,
 'hurwicz': 84.417,
 'benefit': 84.56098765432097,
 'optimistic': 89.013}
```

In [12]:

```
difference = {}
for key, value in results_dict.items():
    difference[key] = (1 + (results_dict[key] - pessimistic_price) / results_dict[key])*100-100
difference
```

Out[12]:
```
{'pessimistic': 0.0,
 'laplace': 10.012988997554999,
 'hurwicz': 16.290557589111202,
 'benefit': 16.433095260342427,
 'optimistic': 20.612719490411507}
```

In [13]:
```
results_df = pd.DataFrame(index=["Price (\xa2 per lb)", 'Improvement %'], data=[results_dict, difference])
results_df
```

Out[13]:

	pessimistic	laplace	hurwicz	benefit	optimistic
Price (¢ per lb)	70.665	78.528000	84.417000	84.560988	89.013000
Improvement %	0.000	10.012989	16.290558	16.433095	20.612719

Wald’s Pessimistic

In [1]:
```
###To import the necessary libaries

import pandas as pd
import numpy as np
from gurobipy import *
```

In [2]:
```
#########Parameters Set-up############

# Production, budget and others
max_production = 96000
min_production = [0]+[60000]*12
max_production_budget = 673719
fcr = 2.2

# Cost
h = [0.036]*13
var_cost = [0]+ [0.2051444222]*12
f = 13260.66667

# Prices for the different feed
pe_one = [0, 0.105, 0.105, 0.105, 0.105, 0.105, 0.105, 0.105, 0.105, 0.105, 0.105, 0.105, 0.105]
pe_two = [0, 0.115, 0.115, 0.115, 0.115, 0.115, 0.115, 0.115, 0.115, 0.115, 0.115, 0.115, 0.115]

# Price for the catfishes
p_one = [0, 0.70665, 0.70665, 0.70665, 0.70665, 0.70665, 0.70665, 0.70665, 0.70665, 0.70665, 0.70665, 0.70665, 0.70665]
#Wald (Pessimistic)
p_two =[0, 0.70665, 0.70665, 0.70665, 0.70665, 0.70665, 0.70665, 0.70665, 0.70665, 0.70665, 0.70665, 0.70665, 0.70665]
#Laplace
#p_two = [0, 0.70665, 0.7852, 0.7852, 0.70665, 0.70665, 0.70665, 0.70665, 0.70665, 0.70665, 0.70665, 0.70665, 0.70665]
#Max regret
#p_two = [0, 0.70665, 0.70665, 0.70665, 0.70665, 0.70665, 0.8380, 0.70665, 0.70665, 0.70665, 0.70665, 0.70665, 0.70665]
#Hurwicz
#p_two = [0, 0.70665, 0.70665, 0.8441, 0.70665, 0.70665, 0.70665, 0.70665, 0.70665, 0.70665, 0.70665, 0.70665, 0.70665]
#Benefit
#p_two = [0, 0.8456, 0.70665, 0.8456, 0.70665, 0.70665, 0.70665, 0.70665, 0.70665, 0.70665, 0.70665, 0.70665, 0.70665]
#Wald (Optimistic)
#p_two = [0, 0.70665, 0.70665, 0.8901, 0.70665, 0.70665, 0.70665, 0.70665, 0.70665, 0.70665, 0.70665, 0.70665, 0.70665]

#survival rate
sr = [0.8490,0.8490]
#sr = [0.8490, 0.9052]

t= len(min_production)

print("min_production:", len(min_production))
print("h:", len(h))
print("var_cost:", len(var_cost))
print("pe_one:", len(pe_one))
print("pe_two:", len(pe_two))
print("p_one:", len(p_one))
```

```
print("p_two:", len(p_two))
print("t:", t)
```

```
min_production: 13
h: 13
var_cost: 13
pe_one: 13
pe_two: 13
p_one: 13
p_two: 13
t: 13
```

In [3]:

```
###Without uncertainty to the survival rate for the feed
```

In [4]:

```
#########Model Set-up###############

m = Model("production")

### Decision Variables:
# q = quantity of catfish produced (in pound) in each month t
q_one = m.addVars(t, name = "quantity_produced_feedone")
q_two = m.addVars(t, name = "quantity_produced_feedtwo")
# x = quantity of catfish (in pound) that will be kept as inventory in each month t
x = m.addVars(t, name = "inventory_kept")
# dt = quantity of catfish (in pound) that will be sold to the intermediaries
d = m.addVars(t, name = "quantity_sold")

# e_one = quantity of feed 1 used in each month t (pound/month) (except month 0 for the planning month)
e_one = m.addVars(t, name = "quantity_feed_one")
# e_two = quantity of feed 2 used in each month t (pound/month) (except month 0 for the planning month)
e_two = m.addVars(t, name = "quantity_feed_two")
```

```
Academic license - for non-commercial use only - expires 2021-12-11
Using license file C:\Users\Sophil\gurobi.lic
```

In [5]:

```
#to set the objective function
m.setObjective( ( quicksum(p_one[i]*min_production[i] for i in range(t))
                 + quicksum(np.max((d[i]- min_production[i]),0) * p_two[i] for i in range(t))
                 -12*f
                 - quicksum(var_cost[i]*(q_one[i] + q_two[i]) for i in range(t))
                - quicksum(pe_one[i]*e_one[i] for i in range(t))
                - quicksum(pe_two[i]*e_two[i] for i in range(t)) - quicksum(h[i]*x[i] for i in range(t)) ), GRB.MAXIMIZE)
```

In [6]:

```
#Add the constraints for start, t=0 and t=12
m.addConstr(q_one[0] == 0, "quantity produced at the start, t=0")
m.addConstr(q_two[0] == 0, "quantity produced at the start, t=0")
m.addConstr(x[0] == 60000, "inventory at the start, t=0")
m.addConstr(d[0] == 0 , "quantity sold at the start, t=0")
m.addConstr(e_one[0] == 0, "quantity of feed 1 at the start, t=0")
m.addConstr(e_two[0] == 0, "quantity of feed 2 at the start, t=0")
```

```python
m.addConstr(x[12] >= 60000, "inventory at the end, t=12")

###Add the inventory constraint
m.addConstr( ( (x[1] == sr[0]*q_one[1] + sr[1]*q_two[1] + x[0] - d[1]) ) , "inventory1")
m.addConstr( ( (x[2] == sr[0]*q_one[2] + sr[1]*q_two[2] + x[1] - d[2]) ) , "inventory2")
m.addConstr( ( (x[3] == sr[0]*q_one[3] + sr[1]*q_two[3] + x[2] - d[3]) ) , "inventory3")
m.addConstr( ( (x[4] == sr[0]*q_one[4] + sr[1]*q_two[4] + x[3] - d[4]) ) , "inventory4")
m.addConstr( ( (x[5] == sr[0]*q_one[5] + sr[1]*q_two[5] + x[4] - d[5]) ) , "inventory5")
m.addConstr( ( (x[6] == sr[0]*q_one[6] + sr[1]*q_two[6] + x[5] - d[6]) ) , "inventory6")
m.addConstr( ( (x[7] == sr[0]*q_one[7] + sr[1]*q_two[7] + x[6] - d[7]) ) , "inventory7")
m.addConstr( ( (x[8] == sr[0]*q_one[8] + sr[1]*q_two[8] + x[7] - d[8]) ) , "inventory8")
m.addConstr( ( (x[9] == sr[0]*q_one[9] + sr[1]*q_two[9] + x[8] - d[9]) ) , "inventory9")
m.addConstr( ( (x[10] == sr[0]*q_one[10] + sr[1]*q_two[10] + x[9] - d[10]) ) , "inventory10")
m.addConstr( ( (x[11] == sr[0]*q_one[11] + sr[1]*q_two[11] + x[10] - d[11]) ) , "inventory11")
m.addConstr( ( (x[12] == sr[0]*q_one[12] + sr[1]*q_two[12] + x[11] - d[12]) ) , "inventory12")

# Add selling quantity constraint (dt >= mt)
m.addConstrs( ( d[i]  >= min_production[i] for i in range(t) ) ,"Selling_quantity")

#Add production capacity production
m.addConstrs( ( q_one[i] + q_two[i] <= max_production for i in range(t) ) ,"production_capacity")

#Add budget for feed constraint ((∑ pe_onet * e_onet  + ∑ pe_twot * e_twot)
# <= max_production_budget - (12 * ft) - (∑ vt*qt)) - (∑ ht*xt)
m.addConstr( ( quicksum(pe_one[i]*e_one[i] + pe_two[i]*e_two[i] for i in range(t))
              <= max_production_budget - 12*f -quicksum(var_cost[i]*(q_one[i] + q_two[i]) for i in range(t))
                  - quicksum(x[i]*h[i] for i in range(t)) ), "feed_budget")

#Add feed constraint (e_onet  + e_twot >= FCR * qt)
m.addConstrs( ( e_one[i]/ 2.2 >= q_one[i] for i in range(t) ), "feed_constraint1")
m.addConstrs( ( e_two[i]/ 2.2 >= q_two[i] for i in range(t) ), "feed_constraint2")
#m.addConstrs( ( (e_one[i] + e_two[i]) >= 2.2*q[i] for i in range(t) ) ,"feed_constraint")

#Add survival constraint (0.8490 e_onet +  0.97052 e_twot - 0.8(e_onet  + e_twot) >= 0)
m.addConstrs( ( sr[0]*e_one[i] + sr[1]*e_two[i]  >= 0.8*(e_one[i] + e_two[i]) for i in range(t) ), 'Survival')
```

Out[6]:
```
{0: <gurobi.Constr *Awaiting Model Update*>,
 1: <gurobi.Constr *Awaiting Model Update*>,
 2: <gurobi.Constr *Awaiting Model Update*>,
 3: <gurobi.Constr *Awaiting Model Update*>,
 4: <gurobi.Constr *Awaiting Model Update*>,
 5: <gurobi.Constr *Awaiting Model Update*>,
 6: <gurobi.Constr *Awaiting Model Update*>,
 7: <gurobi.Constr *Awaiting Model Update*>,
 8: <gurobi.Constr *Awaiting Model Update*>,
 9: <gurobi.Constr *Awaiting Model Update*>,
 10: <gurobi.Constr *Awaiting Model Update*>,
 11: <gurobi.Constr *Awaiting Model Update*>,
 12: <gurobi.Constr *Awaiting Model Update*>}
```

In [7]:
```python
# Solving the model
m.optimize()
```

```python
#  Print optimal solutions and optimal value
print("\n Optimal Solution :\n")
for i, v in enumerate(m.getVars()):
    print(v.VarName, v.x)

print('Obj:', m.objVal)
```

```
Gurobi Optimizer version 9.1.2 build v9.1.2rc0 (win64)
Thread count: 4 physical cores, 8 logical processors, using up to 8 threads
Optimize a model with 85 rows, 78 columns and 245 nonzeros
Model fingerprint: 0x247cd328
Coefficient statistics:
  Matrix range     [4e-02, 1e+00]
  Objective range  [4e-02, 7e-01]
  Bounds range     [0e+00, 0e+00]
  RHS range        [6e+04, 5e+05]
Presolve removed 60 rows and 31 columns
Presolve time: 0.01s
Presolved: 25 rows, 47 columns, 117 nonzeros

Iteration    Objective       Primal Inf.    Dual Inf.      Time
       0    6.7838400e+31   1.200000e+31   6.783840e+01      0s
      32    2.3253677e+04   0.000000e+00   0.000000e+00      0s

Solved in 32 iterations and 0.01 seconds
Optimal objective  2.325367679e+04

 Optimal Solution :

quantity_produced_feedone[0] 0.0
quantity_produced_feedone[1] 96000.0
quantity_produced_feedone[2] 96000.0
quantity_produced_feedone[3] 96000.0
quantity_produced_feedone[4] 96000.0
quantity_produced_feedone[5] 96000.0
quantity_produced_feedone[6] 96000.0
quantity_produced_feedone[7] 96000.0
quantity_produced_feedone[8] 96000.0
quantity_produced_feedone[9] 96000.0
quantity_produced_feedone[10] 96000.0
quantity_produced_feedone[11] 96000.0
quantity_produced_feedone[12] 96000.0
quantity_produced_feedtwo[0] 0.0
quantity_produced_feedtwo[1] 0.0
quantity_produced_feedtwo[2] 0.0
quantity_produced_feedtwo[3] 0.0
quantity_produced_feedtwo[4] 0.0
quantity_produced_feedtwo[5] 0.0
quantity_produced_feedtwo[6] 0.0
quantity_produced_feedtwo[7] 0.0
quantity_produced_feedtwo[8] 0.0
quantity_produced_feedtwo[9] 0.0
quantity_produced_feedtwo[10] 0.0
quantity_produced_feedtwo[11] 0.0
quantity_produced_feedtwo[12] 0.0
inventory_kept[0] 60000.0
```

```
inventory_kept[1] 0.0
inventory_kept[2] 0.0
inventory_kept[3] 0.0
inventory_kept[4] 0.0
inventory_kept[5] 0.0
inventory_kept[6] 0.0
inventory_kept[7] 0.0
inventory_kept[8] 0.0
inventory_kept[9] 0.0
inventory_kept[10] 16992.0
inventory_kept[11] 38496.0
inventory_kept[12] 60000.0
quantity_sold[0] 0.0
quantity_sold[1] 141504.0
quantity_sold[2] 81504.0
quantity_sold[3] 81504.0
quantity_sold[4] 81504.0
quantity_sold[5] 81504.0
quantity_sold[6] 81504.0
quantity_sold[7] 81504.0
quantity_sold[8] 81504.0
quantity_sold[9] 81504.0
quantity_sold[10] 64512.0
quantity_sold[11] 60000.0
quantity_sold[12] 60000.0
quantity_feed_one[0] 0.0
quantity_feed_one[1] 211200.0
quantity_feed_one[2] 211200.0
quantity_feed_one[3] 211200.0
quantity_feed_one[4] 211200.0
quantity_feed_one[5] 211200.0
quantity_feed_one[6] 211200.0
quantity_feed_one[7] 211200.0
quantity_feed_one[8] 211200.0
quantity_feed_one[9] 211200.0
quantity_feed_one[10] 211200.0
quantity_feed_one[11] 211200.0
quantity_feed_one[12] 211200.0
quantity_feed_two[0] 0.0
quantity_feed_two[1] 0.0
quantity_feed_two[2] 0.0
quantity_feed_two[3] 0.0
quantity_feed_two[4] 0.0
quantity_feed_two[5] 0.0
quantity_feed_two[6] 0.0
quantity_feed_two[7] 0.0
quantity_feed_two[8] 0.0
quantity_feed_two[9] 0.0
quantity_feed_two[10] 0.0
quantity_feed_two[11] 0.0
quantity_feed_two[12] 0.0
Obj: 23253.676785600022
```

In [8]:

```
solution = np.array(m.x)
len(solution)
solution = solution.reshape(6,13)
df = pd.DataFrame({'q_one':solution[0],
```

```python
                 'q_two':solution[1],
                 'x':solution[2],
                 'd':solution[3],
                 'e_one':solution[4],
                 'e_two':solution[5]})
df
```

Out[8]:

	q_one	q_two	x	d	e_one	e_two
0	0.0	0.0	60000.0	0.0	0.0	0.0
1	96000.0	0.0	0.0	141504.0	211200.0	0.0
2	96000.0	0.0	0.0	81504.0	211200.0	0.0
3	96000.0	0.0	0.0	81504.0	211200.0	0.0
4	96000.0	0.0	0.0	81504.0	211200.0	0.0
5	96000.0	0.0	0.0	81504.0	211200.0	0.0
6	96000.0	0.0	0.0	81504.0	211200.0	0.0
7	96000.0	0.0	0.0	81504.0	211200.0	0.0
8	96000.0	0.0	0.0	81504.0	211200.0	0.0
9	96000.0	0.0	0.0	81504.0	211200.0	0.0
10	96000.0	0.0	16992.0	64512.0	211200.0	0.0
11	96000.0	0.0	38496.0	60000.0	211200.0	0.0
12	96000.0	0.0	60000.0	60000.0	211200.0	0.0

In [ ]:

Laplace

In [1]:

```
###To import the necessary libaries

import pandas as pd
import numpy as np
from gurobipy import *
```

In [2]:

```
#########Parameters Set-up############

# Production, budget and others
max_production = 96000
min_production = [0]+[60000]*12
max_production_budget = 673719
fcr = 2.2

# Cost
h = [0.036]*13
var_cost = [0]+ [0.2051444222]*12
f = 13260.66667

# Prices for the different feed
pe_one = [0, 0.105, 0.105, 0.105, 0.105, 0.105, 0.105, 0.105, 0.105, 0.105, 0.105, 0.105, 0.105]
pe_two = [0, 0.115, 0.115, 0.115, 0.115, 0.115, 0.115, 0.115, 0.115, 0.115, 0.115, 0.115, 0.115]

# Price for the catfishes
p_one = [0, 0.70665, 0.70665, 0.70665, 0.70665, 0.70665, 0.70665, 0.70665, 0.70665, 0.70665, 0.70665, 0.70665, 0.70665]
#Wald (Pessimistic)
#p_two =[0, 0.70665, 0.70665, 0.70665, 0.70665, 0.70665, 0.70665, 0.70665, 0.70665, 0.70665, 0.70665, 0.70665, 0.70665]
#Laplace
p_two = [0, 0.70665, 0.7852, 0.7852, 0.70665, 0.70665, 0.70665, 0.70665, 0.70665, 0.70665, 0.70665, 0.70665, 0.70665]
#Max regret
#p_two = [0, 0.70665, 0.70665, 0.70665, 0.70665, 0.70665, 0.8380, 0.70665, 0.70665, 0.70665, 0.70665, 0.70665, 0.70665]
#Hurwicz
#p_two = [0, 0.70665, 0.70665, 0.8441, 0.70665, 0.70665, 0.70665, 0.70665, 0.70665, 0.70665, 0.70665, 0.70665, 0.70665]
#Benefit
#p_two = [0, 0.8456, 0.70665, 0.8456, 0.70665, 0.70665, 0.70665, 0.70665, 0.70665, 0.70665, 0.70665, 0.70665, 0.70665]
#Wald (Optimistic)
#p_two = [0, 0.70665, 0.70665, 0.8901, 0.70665, 0.70665, 0.70665, 0.70665, 0.70665, 0.70665, 0.70665, 0.70665, 0.70665]

#survival rate
sr = [0.8490,0.8490]
#sr = [0.8490, 0.9052]

t= len(min_production)

print("min_production:", len(min_production))
print("h:", len(h))
print("var_cost:", len(var_cost))
print("pe_one:", len(pe_one))
print("pe_two:", len(pe_two))
print("p_one:", len(p_one))
```

```
print("p_two:", len(p_two))
print("t:", t)
```

```
min_production: 13
h: 13
var_cost: 13
pe_one: 13
pe_two: 13
p_one: 13
p_two: 13
t: 13
```

In [3]:

```
###Without uncertainty to the survival rate for the feed
```

In [4]:

```
#########Model Set-up###############

m = Model("production")

### Decision Variables:
# q = quantity of catfish produced (in pound) in each month t
q_one = m.addVars(t, name = "quantity_produced_feedone")
q_two = m.addVars(t, name = "quantity_produced_feedtwo")
# x = quantity of catfish (in pound) that will be kept as inventory in each month t
x = m.addVars(t, name = "inventory_kept")
# dt = quantity of catfish (in pound) that will be sold to the intermediaries
d = m.addVars(t, name = "quantity_sold")

# e_one = quantity of feed 1 used in each month t (pound/month) (except month 0 for the planning month)
e_one = m.addVars(t, name = "quantity_feed_one")
# e_two = quantity of feed 2 used in each month t (pound/month) (except month 0 for the planning month)
e_two = m.addVars(t, name = "quantity_feed_two")
```

```
Academic license - for non-commercial use only - expires 2021-12-11
Using license file C:\Users\Sophil\gurobi.lic
```

In [5]:

```
#to set the objective function
m.setObjective( ( quicksum(p_one[i]*min_production[i] for i in range(t))
                 + quicksum(np.max((d[i]- min_production[i]),0) * p_two[i] for i in range(t))
                 -12*f
                 - quicksum(var_cost[i]*(q_one[i] + q_two[i]) for i in range(t))
                - quicksum(pe_one[i]*e_one[i] for i in range(t))
                - quicksum(pe_two[i]*e_two[i] for i in range(t)) - quicksum(h[i]*x[i] for i in range(t)) ), GRB.MAXIMIZE)
```

In [6]:

```
#Add the constraints for start, t=0 and t=12
m.addConstr(q_one[0] == 0, "quantity produced at the start, t=0")
m.addConstr(q_two[0] == 0, "quantity produced at the start, t=0")
m.addConstr(x[0] == 60000, "inventory at the start, t=0")
m.addConstr(d[0] == 0 , "quantity sold at the start, t=0")
m.addConstr(e_one[0] == 0, "quantity of feed 1 at the start, t=0")
m.addConstr(e_two[0] == 0, "quantity of feed 2 at the start, t=0")
```

```python
m.addConstr(x[12] >= 60000, "inventory at the end, t=12")

###Add the inventory constraint
m.addConstr( ( (x[1] == sr[0]*q_one[1] + sr[1]*q_two[1] + x[0] - d[1]) ) , "inventory1")
m.addConstr( ( (x[2] == sr[0]*q_one[2] + sr[1]*q_two[2] + x[1] - d[2]) ) , "inventory2")
m.addConstr( ( (x[3] == sr[0]*q_one[3] + sr[1]*q_two[3] + x[2] - d[3]) ) , "inventory3")
m.addConstr( ( (x[4] == sr[0]*q_one[4] + sr[1]*q_two[4] + x[3] - d[4]) ) , "inventory4")
m.addConstr( ( (x[5] == sr[0]*q_one[5] + sr[1]*q_two[5] + x[4] - d[5]) ) , "inventory5")
m.addConstr( ( (x[6] == sr[0]*q_one[6] + sr[1]*q_two[6] + x[5] - d[6]) ) , "inventory6")
m.addConstr( ( (x[7] == sr[0]*q_one[7] + sr[1]*q_two[7] + x[6] - d[7]) ) , "inventory7")
m.addConstr( ( (x[8] == sr[0]*q_one[8] + sr[1]*q_two[8] + x[7] - d[8]) ) , "inventory8")
m.addConstr( ( (x[9] == sr[0]*q_one[9] + sr[1]*q_two[9] + x[8] - d[9]) ) , "inventory9")
m.addConstr( ( (x[10] == sr[0]*q_one[10] + sr[1]*q_two[10] + x[9] - d[10]) ) , "inventory10")
m.addConstr( ( (x[11] == sr[0]*q_one[11] + sr[1]*q_two[11] + x[10] - d[11]) ) , "inventory11")
m.addConstr( ( (x[12] == sr[0]*q_one[12] + sr[1]*q_two[12] + x[11] - d[12]) ) , "inventory12")

# Add selling quantity constraint (dt >= mt)
m.addConstrs( ( d[i]  >= min_production[i] for i in range(t) ) ,"Selling_quantity")

#Add production capacity production
m.addConstrs( ( q_one[i] + q_two[i] <= max_production for i in range(t) ) ,"production_capacity")

#Add budget for feed constraint ((∑ pe_onet * e_onet  + ∑ pe_twot * e_twot)
# <= max_production_budget – (12 * ft) – (∑ vt*qt)) – (∑ ht*xt)
m.addConstr( ( quicksum(pe_one[i]*e_one[i] + pe_two[i]*e_two[i] for i in range(t))
              <= max_production_budget - 12*f -quicksum(var_cost[i]*(q_one[i] + q_two[i]) for i in range(t))
                 - quicksum(x[i]*h[i] for i in range(t)) ), "feed_budget")

#Add feed constraint (e_onet  + e_twot >= FCR * qt)
m.addConstrs( ( e_one[i]/ 2.2 >= q_one[i] for i in range(t) ), "feed_constraint1")
m.addConstrs( ( e_two[i]/ 2.2 >= q_two[i] for i in range(t) ), "feed_constraint2")
#m.addConstrs( ( (e_one[i] + e_two[i]) >= 2.2*q[i] for i in range(t) ) ,"feed_constraint")

#Add survival constraint (0.8490 e_onet +  0.97052 e_twot – 0.8(e_onet  + e_twot) >= 0)
m.addConstrs( ( sr[0]*e_one[i] + sr[1]*e_two[i]  >= 0.8*(e_one[i] + e_two[i]) for i in range(t) ), 'Survival')
```

Out[6]:

```
{0: <gurobi.Constr *Awaiting Model Update*>,
 1: <gurobi.Constr *Awaiting Model Update*>,
 2: <gurobi.Constr *Awaiting Model Update*>,
 3: <gurobi.Constr *Awaiting Model Update*>,
 4: <gurobi.Constr *Awaiting Model Update*>,
 5: <gurobi.Constr *Awaiting Model Update*>,
 6: <gurobi.Constr *Awaiting Model Update*>,
 7: <gurobi.Constr *Awaiting Model Update*>,
 8: <gurobi.Constr *Awaiting Model Update*>,
 9: <gurobi.Constr *Awaiting Model Update*>,
 10: <gurobi.Constr *Awaiting Model Update*>,
 11: <gurobi.Constr *Awaiting Model Update*>,
 12: <gurobi.Constr *Awaiting Model Update*>}
```

In [7]:

```python
# Solving the model
m.optimize()
```

```python
#  Print optimal solutions and optimal value
print("\n Optimal Solution :\n")
for i, v in enumerate(m.getVars()):
    print(v.VarName, v.x)

print('Obj:', m.objVal)
```

```
Gurobi Optimizer version 9.1.2 build v9.1.2rc0 (win64)
Thread count: 4 physical cores, 8 logical processors, using up to 8 threads
Optimize a model with 85 rows, 78 columns and 245 nonzeros
Model fingerprint: 0x03a791a1
Coefficient statistics:
  Matrix range     [4e-02, 1e+00]
  Objective range  [4e-02, 8e-01]
  Bounds range     [0e+00, 0e+00]
  RHS range        [6e+04, 5e+05]
Presolve removed 60 rows and 31 columns
Presolve time: 0.00s
Presolved: 25 rows, 47 columns, 117 nonzeros

Iteration    Objective       Primal Inf.    Dual Inf.      Time
       0    6.9095200e+31   1.200000e+31   6.909520e+01      0s
      40    3.0099950e+04   0.000000e+00   0.000000e+00      0s

Solved in 40 iterations and 0.01 seconds
Optimal objective  3.009995039e+04

 Optimal Solution :

quantity_produced_feedone[0] 0.0
quantity_produced_feedone[1] 96000.0
quantity_produced_feedone[2] 96000.0
quantity_produced_feedone[3] 96000.0
quantity_produced_feedone[4] 96000.0
quantity_produced_feedone[5] 96000.0
quantity_produced_feedone[6] 96000.0
quantity_produced_feedone[7] 96000.0
quantity_produced_feedone[8] 96000.0
quantity_produced_feedone[9] 96000.0
quantity_produced_feedone[10] 96000.0
quantity_produced_feedone[11] 96000.0
quantity_produced_feedone[12] 96000.0
quantity_produced_feedtwo[0] 0.0
quantity_produced_feedtwo[1] 0.0
quantity_produced_feedtwo[2] 0.0
quantity_produced_feedtwo[3] 0.0
quantity_produced_feedtwo[4] 0.0
quantity_produced_feedtwo[5] 0.0
quantity_produced_feedtwo[6] 0.0
quantity_produced_feedtwo[7] 0.0
quantity_produced_feedtwo[8] 0.0
quantity_produced_feedtwo[9] 0.0
quantity_produced_feedtwo[10] 0.0
quantity_produced_feedtwo[11] 0.0
quantity_produced_feedtwo[12] 0.0
inventory_kept[0] 60000.0
```

```
inventory_kept[1] 81504.0
inventory_kept[2] 0.0
inventory_kept[3] 0.0
inventory_kept[4] 0.0
inventory_kept[5] 0.0
inventory_kept[6] 0.0
inventory_kept[7] 0.0
inventory_kept[8] 0.0
inventory_kept[9] 0.0
inventory_kept[10] 16992.0
inventory_kept[11] 38496.0
inventory_kept[12] 60000.0
quantity_sold[0] 0.0
quantity_sold[1] 60000.0
quantity_sold[2] 163008.0
quantity_sold[3] 81504.0
quantity_sold[4] 81504.0
quantity_sold[5] 81504.0
quantity_sold[6] 81504.0
quantity_sold[7] 81504.0
quantity_sold[8] 81504.0
quantity_sold[9] 81504.0
quantity_sold[10] 64512.0
quantity_sold[11] 60000.0
quantity_sold[12] 60000.0
quantity_feed_one[0] 0.0
quantity_feed_one[1] 211200.0
quantity_feed_one[2] 211200.0
quantity_feed_one[3] 211200.0
quantity_feed_one[4] 211200.0
quantity_feed_one[5] 211200.0
quantity_feed_one[6] 211200.0
quantity_feed_one[7] 211200.0
quantity_feed_one[8] 211200.0
quantity_feed_one[9] 211200.0
quantity_feed_one[10] 211200.0
quantity_feed_one[11] 211200.0
quantity_feed_one[12] 211200.0
quantity_feed_two[0] 0.0
quantity_feed_two[1] 0.0
quantity_feed_two[2] 0.0
quantity_feed_two[3] 0.0
quantity_feed_two[4] 0.0
quantity_feed_two[5] 0.0
quantity_feed_two[6] 0.0
quantity_feed_two[7] 0.0
quantity_feed_two[8] 0.0
quantity_feed_two[9] 0.0
quantity_feed_two[10] 0.0
quantity_feed_two[11] 0.0
quantity_feed_two[12] 0.0
Obj: 30099.950385600037
```

In [8]:

```
solution = np.array(m.x)
len(solution)
solution = solution.reshape(6,13)
df = pd.DataFrame({'q_one':solution[0],
```

```
                  'q_two':solution[1],
                  'x':solution[2],
                  'd':solution[3],
                  'e_one':solution[4],
                  'e_two':solution[5]})
df
```

Out[8]:

	q_one	q_two	x	d	e_one	e_two
0	0.0	0.0	60000.0	0.0	0.0	0.0
1	96000.0	0.0	81504.0	60000.0	211200.0	0.0
2	96000.0	0.0	0.0	163008.0	211200.0	0.0
3	96000.0	0.0	0.0	81504.0	211200.0	0.0
4	96000.0	0.0	0.0	81504.0	211200.0	0.0
5	96000.0	0.0	0.0	81504.0	211200.0	0.0
6	96000.0	0.0	0.0	81504.0	211200.0	0.0
7	96000.0	0.0	0.0	81504.0	211200.0	0.0
8	96000.0	0.0	0.0	81504.0	211200.0	0.0
9	96000.0	0.0	0.0	81504.0	211200.0	0.0
10	96000.0	0.0	16992.0	64512.0	211200.0	0.0
11	96000.0	0.0	38496.0	60000.0	211200.0	0.0
12	96000.0	0.0	60000.0	60000.0	211200.0	0.0

In [ ]:

Hurwicz

In [1]:

```python
###To import the necessary libaries

import pandas as pd
import numpy as np
from gurobipy import *
```

In [2]:

```python
#########Parameters Set-up############

# Production, budget and others
max_production = 96000
min_production = [0]+[60000]*12
max_production_budget = 673719
fcr = 2.2

# Cost
h = [0.036]*13
var_cost = [0]+ [0.2051444222]*12
f = 13260.66667

# Prices for the different feed
pe_one = [0, 0.105, 0.105, 0.105, 0.105, 0.105, 0.105, 0.105, 0.105, 0.105, 0.105, 0.105, 0.105]
pe_two = [0, 0.115, 0.115, 0.115, 0.115, 0.115, 0.115, 0.115, 0.115, 0.115, 0.115, 0.115, 0.115]

# Price for the catfishes
p_one = [0, 0.70665, 0.70665, 0.70665, 0.70665, 0.70665, 0.70665, 0.70665, 0.70665, 0.70665, 0.70665, 0.70665, 0.70665]
#Wald (Pessimistic)
#p_two =[0, 0.70665, 0.70665, 0.70665, 0.70665, 0.70665, 0.70665, 0.70665, 0.70665, 0.70665, 0.70665, 0.70665, 0.70665]
#Laplace
#p_two = [0, 0.70665, 0.7852, 0.7852, 0.70665, 0.70665, 0.70665, 0.70665, 0.70665, 0.70665, 0.70665, 0.70665, 0.70665]
#Max regret
#p_two = [0, 0.70665, 0.70665, 0.70665, 0.70665, 0.70665, 0.8380, 0.70665, 0.70665, 0.70665, 0.70665, 0.70665, 0.70665]
#Hurwicz
p_two = [0, 0.70665, 0.70665, 0.8441, 0.70665, 0.70665, 0.70665, 0.70665, 0.70665, 0.70665, 0.70665, 0.70665, 0.70665]
#Benefit
#p_two = [0, 0.8456, 0.70665, 0.8456, 0.70665, 0.70665, 0.70665, 0.70665, 0.70665, 0.70665, 0.70665, 0.70665, 0.70665]
#Wald (Optimistic)
#p_two = [0, 0.70665, 0.70665, 0.8901, 0.70665, 0.70665, 0.70665, 0.70665, 0.70665, 0.70665, 0.70665, 0.70665, 0.70665]

#survival rate
sr = [0.8490,0.8490]
#sr = [0.8490, 0.9052]

t= len(min_production)

print("min_production:", len(min_production))
print("h:", len(h))
print("var_cost:", len(var_cost))
print("pe_one:", len(pe_one))
print("pe_two:", len(pe_two))
print("p_one:", len(p_one))
```

```python
print("p_two:", len(p_two))
print("t:", t)
```

```
min_production: 13
h: 13
var_cost: 13
pe_one: 13
pe_two: 13
p_one: 13
p_two: 13
t: 13
```

In [3]:

```python
###Without uncertainty to the survival rate for the feed
```

In [4]:

```python
#########Model Set-up###############

m = Model("production")

### Decision Variables:
# q = quantity of catfish produced (in pound) in each month t
q_one = m.addVars(t, name = "quantity_produced_feedone")
q_two = m.addVars(t, name = "quantity_produced_feedtwo")
# x = quantity of catfish (in pound) that will be kept as inventory in each month t
x = m.addVars(t, name = "inventory_kept")
# dt = quantity of catfish (in pound) that will be sold to the intermediaries
d = m.addVars(t, name = "quantity_sold")

# e_one = quantity of feed 1 used in each month t (pound/month) (except month 0 for the planning month)
e_one = m.addVars(t, name = "quantity_feed_one")
# e_two = quantity of feed 2 used in each month t (pound/month) (except month 0 for the planning month)
e_two = m.addVars(t, name = "quantity_feed_two")
```

```
Academic license - for non-commercial use only - expires 2021-12-11
Using license file C:\Users\Sophil\gurobi.lic
```

In [5]:

```python
#to set the objective function
m.setObjective( ( quicksum(p_one[i]*min_production[i] for i in range(t))
                 + quicksum(np.max((d[i]- min_production[i]),0) * p_two[i] for i in range(t))
                 -12*f
                 - quicksum(var_cost[i]*(q_one[i] + q_two[i]) for i in range(t))
                - quicksum(pe_one[i]*e_one[i] for i in range(t))
                - quicksum(pe_two[i]*e_two[i] for i in range(t)) - quicksum(h[i]*x[i] for i in range(t)) ), GRB.MAXIMIZE)
```

In [6]:

```python
#Add the constraints for start, t=0 and t=12
m.addConstr(q_one[0] == 0, "quantity produced at the start, t=0")
m.addConstr(q_two[0] == 0, "quantity produced at the start, t=0")
m.addConstr(x[0] == 60000, "inventory at the start, t=0")
m.addConstr(d[0] == 0 , "quantity sold at the start, t=0")
m.addConstr(e_one[0] == 0, "quantity of feed 1 at the start, t=0")
m.addConstr(e_two[0] == 0, "quantity of feed 2 at the start, t=0")
```

```
m.addConstr(x[12] >= 60000, "inventory at the end, t=12")

###Add the inventory constraint
m.addConstr( ( (x[1] == sr[0]*q_one[1] + sr[1]*q_two[1] + x[0] - d[1]) ) , "inventory1")
m.addConstr( ( (x[2] == sr[0]*q_one[2] + sr[1]*q_two[2] + x[1] - d[2]) ) , "inventory2")
m.addConstr( ( (x[3] == sr[0]*q_one[3] + sr[1]*q_two[3] + x[2] - d[3]) ) , "inventory3")
m.addConstr( ( (x[4] == sr[0]*q_one[4] + sr[1]*q_two[4] + x[3] - d[4]) ) , "inventory4")
m.addConstr( ( (x[5] == sr[0]*q_one[5] + sr[1]*q_two[5] + x[4] - d[5]) ) , "inventory5")
m.addConstr( ( (x[6] == sr[0]*q_one[6] + sr[1]*q_two[6] + x[5] - d[6]) ) , "inventory6")
m.addConstr( ( (x[7] == sr[0]*q_one[7] + sr[1]*q_two[7] + x[6] - d[7]) ) , "inventory7")
m.addConstr( ( (x[8] == sr[0]*q_one[8] + sr[1]*q_two[8] + x[7] - d[8]) ) , "inventory8")
m.addConstr( ( (x[9] == sr[0]*q_one[9] + sr[1]*q_two[9] + x[8] - d[9]) ) , "inventory9")
m.addConstr( ( (x[10] == sr[0]*q_one[10] + sr[1]*q_two[10] + x[9] - d[10]) ) , "inventory10")
m.addConstr( ( (x[11] == sr[0]*q_one[11] + sr[1]*q_two[11] + x[10] - d[11]) ) , "inventory11")
m.addConstr( ( (x[12] == sr[0]*q_one[12] + sr[1]*q_two[12] + x[11] - d[12]) ) , "inventory12")

# Add selling quantity constraint (dt >= mt)
m.addConstrs( ( d[i]  >= min_production[i] for i in range(t) ) ,"Selling_quantity")

#Add production capacity production
m.addConstrs( ( q_one[i] + q_two[i] <= max_production for i in range(t) ) ,"production_capacity")

#Add budget for feed constraint ((∑ pe_onet * e_onet  + ∑ pe_twot * e_twot)
# <= max_production_budget - (12 * ft) - (∑ vt*qt)) - (∑ ht*xt)
m.addConstr( ( quicksum(pe_one[i]*e_one[i] + pe_two[i]*e_two[i] for i in range(t))
              <= max_production_budget - 12*f -quicksum(var_cost[i]*(q_one[i] + q_two[i]) for i in range(t))
                 - quicksum(x[i]*h[i] for i in range(t)) ), "feed_budget")

#Add feed constraint (e_onet  + e_twot >= FCR * qt)
m.addConstrs( ( e_one[i]/ 2.2 >= q_one[i] for i in range(t) ), "feed_constraint1")
m.addConstrs( ( e_two[i]/ 2.2 >= q_two[i] for i in range(t) ), "feed_constraint2")
#m.addConstrs( ( (e_one[i] + e_two[i]) >= 2.2*q[i] for i in range(t) ) ,"feed_constraint")

#Add survival constraint (0.8490 e_onet +  0.97052 e_twot - 0.8(e_onet  + e_twot) >= 0)
m.addConstrs( ( sr[0]*e_one[i] + sr[1]*e_two[i]  >= 0.8*(e_one[i] + e_two[i]) for i in range(t) ), 'Survival')
```

Out[6]:
```
{0: <gurobi.Constr *Awaiting Model Update*>,
 1: <gurobi.Constr *Awaiting Model Update*>,
 2: <gurobi.Constr *Awaiting Model Update*>,
 3: <gurobi.Constr *Awaiting Model Update*>,
 4: <gurobi.Constr *Awaiting Model Update*>,
 5: <gurobi.Constr *Awaiting Model Update*>,
 6: <gurobi.Constr *Awaiting Model Update*>,
 7: <gurobi.Constr *Awaiting Model Update*>,
 8: <gurobi.Constr *Awaiting Model Update*>,
 9: <gurobi.Constr *Awaiting Model Update*>,
 10: <gurobi.Constr *Awaiting Model Update*>,
 11: <gurobi.Constr *Awaiting Model Update*>,
 12: <gurobi.Constr *Awaiting Model Update*>}
```

In [7]:
```
# Solving the model
m.optimize()
```

```python
#  Print optimal solutions and optimal value
print("\n Optimal Solution :\n")
for i, v in enumerate(m.getVars()):
    print(v.VarName, v.x)

print('Obj:', m.objVal)
```

```
Gurobi Optimizer version 9.1.2 build v9.1.2rc0 (win64)
Thread count: 4 physical cores, 8 logical processors, using up to 8 threads
Optimize a model with 85 rows, 78 columns and 245 nonzeros
Model fingerprint: 0xebc94273
Coefficient statistics:
  Matrix range     [4e-02, 1e+00]
  Objective range  [4e-02, 8e-01]
  Bounds range     [0e+00, 0e+00]
  RHS range        [6e+04, 5e+05]
Presolve removed 60 rows and 31 columns
Presolve time: 0.00s
Presolved: 25 rows, 47 columns, 117 nonzeros

Iteration    Objective       Primal Inf.    Dual Inf.      Time
       0    6.8938000e+31   1.200000e+31   6.893800e+01      0s
      36    3.3422196e+04   0.000000e+00   0.000000e+00      0s

Solved in 36 iterations and 0.01 seconds
Optimal objective  3.342219604e+04

 Optimal Solution :

quantity_produced_feedone[0] 0.0
quantity_produced_feedone[1] 96000.0
quantity_produced_feedone[2] 96000.0
quantity_produced_feedone[3] 96000.0
quantity_produced_feedone[4] 96000.0
quantity_produced_feedone[5] 96000.0
quantity_produced_feedone[6] 96000.0
quantity_produced_feedone[7] 96000.0
quantity_produced_feedone[8] 96000.0
quantity_produced_feedone[9] 94148.8370060367
quantity_produced_feedone[10] 96000.0
quantity_produced_feedone[11] 96000.0
quantity_produced_feedone[12] 96000.0
quantity_produced_feedtwo[0] 0.0
quantity_produced_feedtwo[1] 0.0
quantity_produced_feedtwo[2] 0.0
quantity_produced_feedtwo[3] 0.0
quantity_produced_feedtwo[4] 0.0
quantity_produced_feedtwo[5] 0.0
quantity_produced_feedtwo[6] 0.0
quantity_produced_feedtwo[7] 0.0
quantity_produced_feedtwo[8] 0.0
quantity_produced_feedtwo[9] 0.0
quantity_produced_feedtwo[10] 0.0
quantity_produced_feedtwo[11] 0.0
quantity_produced_feedtwo[12] 0.0
inventory_kept[0] 60000.0
```

```
inventory_kept[1] 81504.0
inventory_kept[2] 103008.0
inventory_kept[3] 0.0
inventory_kept[4] 0.0
inventory_kept[5] 0.0
inventory_kept[6] 0.0
inventory_kept[7] 0.0
inventory_kept[8] 0.0
inventory_kept[9] 0.0
inventory_kept[10] 16992.0
inventory_kept[11] 38496.0
inventory_kept[12] 60000.0
quantity_sold[0] 0.0
quantity_sold[1] 60000.0
quantity_sold[2] 60000.0
quantity_sold[3] 184512.0
quantity_sold[4] 81504.0
quantity_sold[5] 81504.0
quantity_sold[6] 81504.0
quantity_sold[7] 81504.0
quantity_sold[8] 81504.0
quantity_sold[9] 79932.36261812516
quantity_sold[10] 64512.0
quantity_sold[11] 60000.0
quantity_sold[12] 60000.0
quantity_feed_one[0] 0.0
quantity_feed_one[1] 211200.0
quantity_feed_one[2] 211200.0
quantity_feed_one[3] 211200.0
quantity_feed_one[4] 211200.0
quantity_feed_one[5] 211200.0
quantity_feed_one[6] 211200.0
quantity_feed_one[7] 211200.0
quantity_feed_one[8] 211200.0
quantity_feed_one[9] 207127.44141328076
quantity_feed_one[10] 211200.0
quantity_feed_one[11] 211200.0
quantity_feed_one[12] 211200.0
quantity_feed_two[0] 0.0
quantity_feed_two[1] 0.0
quantity_feed_two[2] 0.0
quantity_feed_two[3] 0.0
quantity_feed_two[4] 0.0
quantity_feed_two[5] 0.0
quantity_feed_two[6] 0.0
quantity_feed_two[7] 0.0
quantity_feed_two[8] 0.0
quantity_feed_two[9] 0.0
quantity_feed_two[10] 0.0
quantity_feed_two[11] 0.0
quantity_feed_two[12] 0.0
Obj: 33422.19604409824
```

In [8]:

```
solution = np.array(m.x)
len(solution)
solution = solution.reshape(6,13)
df = pd.DataFrame({'q_one':solution[0],
```

```python
                  'q_two':solution[1],
                  'x':solution[2],
                  'd':solution[3],
                  'e_one':solution[4],
                  'e_two':solution[5]})
df
```

Out[8]:

	q_one	q_two	x	d	e_one	e_two
0	0.000000	0.0	60000.0	0.000000	0.000000	0.0
1	96000.000000	0.0	81504.0	60000.000000	211200.000000	0.0
2	96000.000000	0.0	103008.0	60000.000000	211200.000000	0.0
3	96000.000000	0.0	0.0	184512.000000	211200.000000	0.0
4	96000.000000	0.0	0.0	81504.000000	211200.000000	0.0
5	96000.000000	0.0	0.0	81504.000000	211200.000000	0.0
6	96000.000000	0.0	0.0	81504.000000	211200.000000	0.0
7	96000.000000	0.0	0.0	81504.000000	211200.000000	0.0
8	96000.000000	0.0	0.0	81504.000000	211200.000000	0.0
9	94148.837006	0.0	0.0	79932.362618	207127.441413	0.0
10	96000.000000	0.0	16992.0	64512.000000	211200.000000	0.0
11	96000.000000	0.0	38496.0	60000.000000	211200.000000	0.0
12	96000.000000	0.0	60000.0	60000.000000	211200.000000	0.0

In [ ]:

Benefit

In [1]:
```python
###To import the necessary libaries

import pandas as pd
import numpy as np
from gurobipy import *
```

In [2]:
```python
#########Parameters Set-up############

# Production, budget and others
max_production = 96000
min_production = [0]+[60000]*12
max_production_budget = 673719
fcr = 2.2

# Cost
h = [0.036]*13
var_cost = [0]+ [0.2051444222]*12
f = 13260.66667

# Prices for the different feed
pe_one = [0, 0.105, 0.105, 0.105, 0.105, 0.105, 0.105, 0.105, 0.105, 0.105, 0.105, 0.105, 0.105]
pe_two = [0, 0.115, 0.115, 0.115, 0.115, 0.115, 0.115, 0.115, 0.115, 0.115, 0.115, 0.115, 0.115]

# Price for the catfishes
p_one = [0, 0.70665, 0.70665, 0.70665, 0.70665, 0.70665, 0.70665, 0.70665, 0.70665, 0.70665, 0.70665, 0.70665, 0.70665]
#Wald (Pessimistic)
#p_two =[0, 0.70665, 0.70665, 0.70665, 0.70665, 0.70665, 0.70665, 0.70665, 0.70665, 0.70665, 0.70665, 0.70665, 0.70665]
#Laplace
#p_two = [0, 0.70665, 0.7852, 0.7852, 0.70665, 0.70665, 0.70665, 0.70665, 0.70665, 0.70665, 0.70665, 0.70665, 0.70665]
#Max regret
#p_two = [0, 0.70665, 0.70665, 0.70665, 0.70665, 0.70665, 0.8380, 0.70665, 0.70665, 0.70665, 0.70665, 0.70665, 0.70665]
#Hurwicz
#p_two = [0, 0.70665, 0.70665, 0.8441, 0.70665, 0.70665, 0.70665, 0.70665, 0.70665, 0.70665, 0.70665, 0.70665, 0.70665]
#Benefit
p_two = [0, 0.8456, 0.70665, 0.8456, 0.70665, 0.70665, 0.70665, 0.70665, 0.70665, 0.70665, 0.70665, 0.70665, 0.70665]
#Wald (Optimistic)
#p_two = [0, 0.70665, 0.70665, 0.8901, 0.70665, 0.70665, 0.70665, 0.70665, 0.70665, 0.70665, 0.70665, 0.70665, 0.70665]

#survival rate
sr = [0.8490,0.8490]
#sr = [0.8490, 0.9052]

t= len(min_production)

print("min_production:", len(min_production))
print("h:", len(h))
print("var_cost:", len(var_cost))
print("pe_one:", len(pe_one))
print("pe_two:", len(pe_two))
print("p_one:", len(p_one))
```

```
print("p_two:", len(p_two))
print("t:", t)
```

```
min_production: 13
h: 13
var_cost: 13
pe_one: 13
pe_two: 13
p_one: 13
p_two: 13
t: 13
```

In [3]:

```
###Without uncertainty to the survival rate for the feed
```

In [4]:

```
#########Model Set-up###############

m = Model("production")

### Decision Variables:
# q = quantity of catfish produced (in pound) in each month t
q_one = m.addVars(t, name = "quantity_produced_feedone")
q_two = m.addVars(t, name = "quantity_produced_feedtwo")
# x = quantity of catfish (in pound) that will be kept as inventory in each month t
x = m.addVars(t, name = "inventory_kept")
# dt = quantity of catfish (in pound) that will be sold to the intermediaries
d = m.addVars(t, name = "quantity_sold")

# e_one = quantity of feed 1 used in each month t (pound/month) (except month 0 for the planning month)
e_one = m.addVars(t, name = "quantity_feed_one")
# e_two = quantity of feed 2 used in each month t (pound/month) (except month 0 for the planning month)
e_two = m.addVars(t, name = "quantity_feed_two")
```

```
Academic license - for non-commercial use only - expires 2021-12-11
Using license file C:\Users\Sophil\gurobi.lic
```

In [5]:

```
#to set the objective function
m.setObjective( ( quicksum(p_one[i]*min_production[i] for i in range(t))
                 + quicksum(np.max((d[i]- min_production[i]),0) * p_two[i] for i in range(t))
                 -12*f
                 - quicksum(var_cost[i]*(q_one[i] + q_two[i]) for i in range(t))
                - quicksum(pe_one[i]*e_one[i] for i in range(t))
                - quicksum(pe_two[i]*e_two[i] for i in range(t)) - quicksum(h[i]*x[i] for i in range(t)) ), GRB.MAXIMIZE)
```

In [6]:

```
#Add the constraints for start, t=0 and t=12
m.addConstr(q_one[0] == 0, "quantity produced at the start, t=0")
m.addConstr(q_two[0] == 0, "quantity produced at the start, t=0")
m.addConstr(x[0] == 60000, "inventory at the start, t=0")
m.addConstr(d[0] == 0 , "quantity sold at the start, t=0")
m.addConstr(e_one[0] == 0, "quantity of feed 1 at the start, t=0")
m.addConstr(e_two[0] == 0, "quantity of feed 2 at the start, t=0")
```

```python
m.addConstr(x[12] >= 60000, "inventory at the end, t=12")

###Add the inventory constraint
m.addConstr( ( (x[1] == sr[0]*q_one[1] + sr[1]*q_two[1] + x[0] - d[1]) ) , "inventory1")
m.addConstr( ( (x[2] == sr[0]*q_one[2] + sr[1]*q_two[2] + x[1] - d[2]) ) , "inventory2")
m.addConstr( ( (x[3] == sr[0]*q_one[3] + sr[1]*q_two[3] + x[2] - d[3]) ) , "inventory3")
m.addConstr( ( (x[4] == sr[0]*q_one[4] + sr[1]*q_two[4] + x[3] - d[4]) ) , "inventory4")
m.addConstr( ( (x[5] == sr[0]*q_one[5] + sr[1]*q_two[5] + x[4] - d[5]) ) , "inventory5")
m.addConstr( ( (x[6] == sr[0]*q_one[6] + sr[1]*q_two[6] + x[5] - d[6]) ) , "inventory6")
m.addConstr( ( (x[7] == sr[0]*q_one[7] + sr[1]*q_two[7] + x[6] - d[7]) ) , "inventory7")
m.addConstr( ( (x[8] == sr[0]*q_one[8] + sr[1]*q_two[8] + x[7] - d[8]) ) , "inventory8")
m.addConstr( ( (x[9] == sr[0]*q_one[9] + sr[1]*q_two[9] + x[8] - d[9]) ) , "inventory9")
m.addConstr( ( (x[10] == sr[0]*q_one[10] + sr[1]*q_two[10] + x[9] - d[10]) ) , "inventory10")
m.addConstr( ( (x[11] == sr[0]*q_one[11] + sr[1]*q_two[11] + x[10] - d[11]) ) , "inventory11")
m.addConstr( ( (x[12] == sr[0]*q_one[12] + sr[1]*q_two[12] + x[11] - d[12]) ) , "inventory12")

# Add selling quantity constraint (dt >= mt)
m.addConstrs( ( d[i]  >= min_production[i] for i in range(t) ) ,"Selling_quantity")

#Add production capacity production
m.addConstrs( ( q_one[i] + q_two[i] <= max_production for i in range(t) ) ,"production_capacity")

#Add budget for feed constraint ((∑ pe_onet * e_onet  + ∑ pe_twot * e_twot)
# <= max_production_budget - (12 * ft) - (∑ vt*qt)) - (∑ ht*xt)
m.addConstr( ( quicksum(pe_one[i]*e_one[i] + pe_two[i]*e_two[i] for i in range(t))
              <= max_production_budget - 12*f -quicksum(var_cost[i]*(q_one[i] + q_two[i]) for i in range(t))
                 - quicksum(x[i]*h[i] for i in range(t)) ), "feed_budget")

#Add feed constraint (e_onet  + e_twot >= FCR * qt)
m.addConstrs( ( e_one[i]/ 2.2 >= q_one[i] for i in range(t) ), "feed_constraint1")
m.addConstrs( ( e_two[i]/ 2.2 >= q_two[i] for i in range(t) ), "feed_constraint2")
#m.addConstrs( ( (e_one[i] + e_two[i]) >= 2.2*q[i] for i in range(t) ) ,"feed_constraint")

#Add survival constraint (0.8490 e_onet +  0.97052 e_twot - 0.8(e_onet  + e_twot) >= 0)
m.addConstrs( ( sr[0]*e_one[i] + sr[1]*e_two[i]  >= 0.8*(e_one[i] + e_two[i]) for i in range(t) ), 'Survival')
```

Out[6]:

```
{0: <gurobi.Constr *Awaiting Model Update*>,
 1: <gurobi.Constr *Awaiting Model Update*>,
 2: <gurobi.Constr *Awaiting Model Update*>,
 3: <gurobi.Constr *Awaiting Model Update*>,
 4: <gurobi.Constr *Awaiting Model Update*>,
 5: <gurobi.Constr *Awaiting Model Update*>,
 6: <gurobi.Constr *Awaiting Model Update*>,
 7: <gurobi.Constr *Awaiting Model Update*>,
 8: <gurobi.Constr *Awaiting Model Update*>,
 9: <gurobi.Constr *Awaiting Model Update*>,
 10: <gurobi.Constr *Awaiting Model Update*>,
 11: <gurobi.Constr *Awaiting Model Update*>,
 12: <gurobi.Constr *Awaiting Model Update*>}
```

In [7]:

```python
# Solving the model
m.optimize()
```

```python
#  Print optimal solutions and optimal value
print("\n Optimal Solution :\n")
for i, v in enumerate(m.getVars()):
    print(v.VarName, v.x)

print('Obj:', m.objVal)
```

```
Gurobi Optimizer version 9.1.2 build v9.1.2rc0 (win64)
Thread count: 4 physical cores, 8 logical processors, using up to 8 threads
Optimize a model with 85 rows, 78 columns and 245 nonzeros
Model fingerprint: 0x10255b72
Coefficient statistics:
  Matrix range     [4e-02, 1e+00]
  Objective range  [4e-02, 8e-01]
  Bounds range     [0e+00, 0e+00]
  RHS range        [6e+04, 5e+05]
Presolve removed 60 rows and 31 columns
Presolve time: 0.01s
Presolved: 25 rows, 47 columns, 117 nonzeros

Iteration    Objective       Primal Inf.    Dual Inf.      Time
       0    7.0061600e+31   1.200000e+31   7.006160e+01      0s
      37    3.9780475e+04   0.000000e+00   0.000000e+00      0s

Solved in 37 iterations and 0.01 seconds
Optimal objective  3.978047519e+04

 Optimal Solution :

quantity_produced_feedone[0] 0.0
quantity_produced_feedone[1] 96000.0
quantity_produced_feedone[2] 96000.0
quantity_produced_feedone[3] 96000.0
quantity_produced_feedone[4] 96000.0
quantity_produced_feedone[5] 96000.0
quantity_produced_feedone[6] 96000.0
quantity_produced_feedone[7] 96000.0
quantity_produced_feedone[8] 96000.0
quantity_produced_feedone[9] 96000.0
quantity_produced_feedone[10] 96000.0
quantity_produced_feedone[11] 96000.0
quantity_produced_feedone[12] 96000.0
quantity_produced_feedtwo[0] 0.0
quantity_produced_feedtwo[1] 0.0
quantity_produced_feedtwo[2] 0.0
quantity_produced_feedtwo[3] 0.0
quantity_produced_feedtwo[4] 0.0
quantity_produced_feedtwo[5] 0.0
quantity_produced_feedtwo[6] 0.0
quantity_produced_feedtwo[7] 0.0
quantity_produced_feedtwo[8] 0.0
quantity_produced_feedtwo[9] 0.0
quantity_produced_feedtwo[10] 0.0
quantity_produced_feedtwo[11] 0.0
quantity_produced_feedtwo[12] 0.0
inventory_kept[0] 60000.0
```

```
inventory_kept[1] 0.0
inventory_kept[2] 21504.0
inventory_kept[3] 0.0
inventory_kept[4] 0.0
inventory_kept[5] 0.0
inventory_kept[6] 0.0
inventory_kept[7] 0.0
inventory_kept[8] 0.0
inventory_kept[9] 0.0
inventory_kept[10] 16992.0
inventory_kept[11] 38496.0
inventory_kept[12] 60000.0
quantity_sold[0] 0.0
quantity_sold[1] 141504.0
quantity_sold[2] 60000.0
quantity_sold[3] 103008.0
quantity_sold[4] 81504.0
quantity_sold[5] 81504.0
quantity_sold[6] 81504.0
quantity_sold[7] 81504.0
quantity_sold[8] 81504.0
quantity_sold[9] 81504.0
quantity_sold[10] 64512.0
quantity_sold[11] 60000.0
quantity_sold[12] 60000.0
quantity_feed_one[0] 0.0
quantity_feed_one[1] 211200.0
quantity_feed_one[2] 211200.0
quantity_feed_one[3] 211200.0
quantity_feed_one[4] 211200.0
quantity_feed_one[5] 211200.0
quantity_feed_one[6] 211200.0
quantity_feed_one[7] 211200.0
quantity_feed_one[8] 211200.0
quantity_feed_one[9] 211200.0
quantity_feed_one[10] 211200.0
quantity_feed_one[11] 211200.0
quantity_feed_one[12] 211200.0
quantity_feed_two[0] 0.0
quantity_feed_two[1] 0.0
quantity_feed_two[2] 0.0
quantity_feed_two[3] 0.0
quantity_feed_two[4] 0.0
quantity_feed_two[5] 0.0
quantity_feed_two[6] 0.0
quantity_feed_two[7] 0.0
quantity_feed_two[8] 0.0
quantity_feed_two[9] 0.0
quantity_feed_two[10] 0.0
quantity_feed_two[11] 0.0
quantity_feed_two[12] 0.0
Obj: 39780.47518559999
```

In [8]:

```
solution = np.array(m.x)
len(solution)
solution = solution.reshape(6,13)
df = pd.DataFrame({'q_one':solution[0],
```

```
                  'q_two':solution[1],
                  'x':solution[2],
                  'd':solution[3],
                  'e_one':solution[4],
                  'e_two':solution[5]})
df
```

Out[8]:

	q_one	q_two	x	d	e_one	e_two
0	0.0	0.0	60000.0	0.0	0.0	0.0
1	96000.0	0.0	0.0	141504.0	211200.0	0.0
2	96000.0	0.0	21504.0	60000.0	211200.0	0.0
3	96000.0	0.0	0.0	103008.0	211200.0	0.0
4	96000.0	0.0	0.0	81504.0	211200.0	0.0
5	96000.0	0.0	0.0	81504.0	211200.0	0.0
6	96000.0	0.0	0.0	81504.0	211200.0	0.0
7	96000.0	0.0	0.0	81504.0	211200.0	0.0
8	96000.0	0.0	0.0	81504.0	211200.0	0.0
9	96000.0	0.0	0.0	81504.0	211200.0	0.0
10	96000.0	0.0	16992.0	64512.0	211200.0	0.0
11	96000.0	0.0	38496.0	60000.0	211200.0	0.0
12	96000.0	0.0	60000.0	60000.0	211200.0	0.0

In [ ]:

Wald’s Optimistic

In [1]:

```
###To import the necessary libaries

import pandas as pd
import numpy as np
from gurobipy import *
```

In [2]:

```
#########Parameters Set-up############

# Production, budget and others
max_production = 96000
min_production = [0]+[60000]*12
max_production_budget = 673719
fcr = 2.2

# Cost
h = [0.036]*13
var_cost = [0]+ [0.2051444222]*12
f = 13260.66667

# Prices for the different feed
pe_one = [0, 0.105, 0.105, 0.105, 0.105, 0.105, 0.105, 0.105, 0.105, 0.105, 0.105, 0.105, 0.105]
pe_two = [0, 0.115, 0.115, 0.115, 0.115, 0.115, 0.115, 0.115, 0.115, 0.115, 0.115, 0.115, 0.115]

# Price for the catfishes
p_one = [0, 0.70665, 0.70665, 0.70665, 0.70665, 0.70665, 0.70665, 0.70665, 0.70665, 0.70665, 0.70665, 0.70665, 0.70665]
#Wald (Pessimistic)
#p_two =[0, 0.70665, 0.70665, 0.70665, 0.70665, 0.70665, 0.70665, 0.70665, 0.70665, 0.70665, 0.70665, 0.70665, 0.70665]
#Laplace
#p_two = [0, 0.70665, 0.7852, 0.7852, 0.70665, 0.70665, 0.70665, 0.70665, 0.70665, 0.70665, 0.70665, 0.70665, 0.70665]
#Max regret
#p_two = [0, 0.70665, 0.70665, 0.70665, 0.70665, 0.70665, 0.8380, 0.70665, 0.70665, 0.70665, 0.70665, 0.70665, 0.70665]
#Hurwicz
#p_two = [0, 0.70665, 0.70665, 0.8441, 0.70665, 0.70665, 0.70665, 0.70665, 0.70665, 0.70665, 0.70665, 0.70665, 0.70665]
#Benefit
#p_two = [0, 0.8456, 0.70665, 0.8456, 0.70665, 0.70665, 0.70665, 0.70665, 0.70665, 0.70665, 0.70665, 0.70665, 0.70665]
#Wald (Optimistic)
p_two = [0, 0.70665, 0.70665, 0.8901, 0.70665, 0.70665, 0.70665, 0.70665, 0.70665, 0.70665, 0.70665, 0.70665, 0.70665]

#survival rate
sr = [0.8490,0.8490]
#sr = [0.8490, 0.9052]

t= len(min_production)

print("min_production:", len(min_production))
print("h:", len(h))
print("var_cost:", len(var_cost))
print("pe_one:", len(pe_one))
print("pe_two:", len(pe_two))
print("p_one:", len(p_one))
```

```
print("p_two:", len(p_two))
print("t:", t)
```

```
min_production: 13
h: 13
var_cost: 13
pe_one: 13
pe_two: 13
p_one: 13
p_two: 13
t: 13
```

In [3]:

```
###Without uncertainty to the survival rate for the feed
```

In [4]:

```
#########Model Set-up###############

m = Model("production")

### Decision Variables:
# q = quantity of catfish produced (in pound) in each month t
q_one = m.addVars(t, name = "quantity_produced_feedone")
q_two = m.addVars(t, name = "quantity_produced_feedtwo")
# x = quantity of catfish (in pound) that will be kept as inventory in each month t
x = m.addVars(t, name = "inventory_kept")
# dt = quantity of catfish (in pound) that will be sold to the intermediaries
d = m.addVars(t, name = "quantity_sold")

# e_one = quantity of feed 1 used in each month t (pound/month) (except month 0 for the planning month)
e_one = m.addVars(t, name = "quantity_feed_one")
# e_two = quantity of feed 2 used in each month t (pound/month) (except month 0 for the planning month)
e_two = m.addVars(t, name = "quantity_feed_two")
```

```
Academic license - for non-commercial use only - expires 2021-12-11
Using license file C:\Users\Sophil\gurobi.lic
```

In [5]:

```
#to set the objective function
m.setObjective( ( quicksum(p_one[i]*min_production[i] for i in range(t))
                 + quicksum(np.max((d[i]- min_production[i]),0) * p_two[i] for i in range(t))
                 -12*f
                 - quicksum(var_cost[i]*(q_one[i] + q_two[i]) for i in range(t))
                - quicksum(pe_one[i]*e_one[i] for i in range(t))
                - quicksum(pe_two[i]*e_two[i] for i in range(t)) - quicksum(h[i]*x[i] for i in range(t)) ), GRB.MAXIMIZE)
```

In [6]:

```
#Add the constraints for start, t=0 and t=12
m.addConstr(q_one[0] == 0, "quantity produced at the start, t=0")
m.addConstr(q_two[0] == 0, "quantity produced at the start, t=0")
m.addConstr(x[0] == 60000, "inventory at the start, t=0")
m.addConstr(d[0] == 0 , "quantity sold at the start, t=0")
m.addConstr(e_one[0] == 0, "quantity of feed 1 at the start, t=0")
m.addConstr(e_two[0] == 0, "quantity of feed 2 at the start, t=0")
```

```python
m.addConstr(x[12] >= 60000, "inventory at the end, t=12")

###Add the inventory constraint
m.addConstr( ( (x[1] == sr[0]*q_one[1] + sr[1]*q_two[1] + x[0] - d[1]) ) , "inventory1")
m.addConstr( ( (x[2] == sr[0]*q_one[2] + sr[1]*q_two[2] + x[1] - d[2]) ) , "inventory2")
m.addConstr( ( (x[3] == sr[0]*q_one[3] + sr[1]*q_two[3] + x[2] - d[3]) ) , "inventory3")
m.addConstr( ( (x[4] == sr[0]*q_one[4] + sr[1]*q_two[4] + x[3] - d[4]) ) , "inventory4")
m.addConstr( ( (x[5] == sr[0]*q_one[5] + sr[1]*q_two[5] + x[4] - d[5]) ) , "inventory5")
m.addConstr( ( (x[6] == sr[0]*q_one[6] + sr[1]*q_two[6] + x[5] - d[6]) ) , "inventory6")
m.addConstr( ( (x[7] == sr[0]*q_one[7] + sr[1]*q_two[7] + x[6] - d[7]) ) , "inventory7")
m.addConstr( ( (x[8] == sr[0]*q_one[8] + sr[1]*q_two[8] + x[7] - d[8]) ) , "inventory8")
m.addConstr( ( (x[9] == sr[0]*q_one[9] + sr[1]*q_two[9] + x[8] - d[9]) ) , "inventory9")
m.addConstr( ( (x[10] == sr[0]*q_one[10] + sr[1]*q_two[10] + x[9] - d[10]) ) , "inventory10")
m.addConstr( ( (x[11] == sr[0]*q_one[11] + sr[1]*q_two[11] + x[10] - d[11]) ) , "inventory11")
m.addConstr( ( (x[12] == sr[0]*q_one[12] + sr[1]*q_two[12] + x[11] - d[12]) ) , "inventory12")

# Add selling quantity constraint (dt >= mt)
m.addConstrs( ( d[i]  >= min_production[i] for i in range(t) ) ,"Selling_quantity")

#Add production capacity production
m.addConstrs( ( q_one[i] + q_two[i] <= max_production for i in range(t) ) ,"production_capacity")

#Add budget for feed constraint ((∑ pe_onet * e_onet  + ∑ pe_twot * e_twot)
# <= max_production_budget - (12 * ft) - (∑ vt*qt)) - (∑ ht*xt)
m.addConstr( ( quicksum(pe_one[i]*e_one[i] + pe_two[i]*e_two[i] for i in range(t))
              <= max_production_budget - 12*f -quicksum(var_cost[i]*(q_one[i] + q_two[i]) for i in range(t))
                  - quicksum(x[i]*h[i] for i in range(t)) ), "feed_budget")

#Add feed constraint (e_onet  + e_twot >= FCR * qt)
m.addConstrs( ( e_one[i]/ 2.2 >= q_one[i] for i in range(t) ), "feed_constraint1")
m.addConstrs( ( e_two[i]/ 2.2 >= q_two[i] for i in range(t) ), "feed_constraint2")
#m.addConstrs( ( (e_one[i] + e_two[i]) >= 2.2*q[i] for i in range(t) ) ,"feed_constraint")

#Add survival constraint (0.8490 e_onet +  0.97052 e_twot - 0.8(e_onet  + e_twot) >= 0)
m.addConstrs( ( sr[0]*e_one[i] + sr[1]*e_two[i]  >= 0.8*(e_one[i] + e_two[i]) for i in range(t) ), 'Survival')
```

Out[6]:

```
{0: <gurobi.Constr *Awaiting Model Update*>,
 1: <gurobi.Constr *Awaiting Model Update*>,
 2: <gurobi.Constr *Awaiting Model Update*>,
 3: <gurobi.Constr *Awaiting Model Update*>,
 4: <gurobi.Constr *Awaiting Model Update*>,
 5: <gurobi.Constr *Awaiting Model Update*>,
 6: <gurobi.Constr *Awaiting Model Update*>,
 7: <gurobi.Constr *Awaiting Model Update*>,
 8: <gurobi.Constr *Awaiting Model Update*>,
 9: <gurobi.Constr *Awaiting Model Update*>,
 10: <gurobi.Constr *Awaiting Model Update*>,
 11: <gurobi.Constr *Awaiting Model Update*>,
 12: <gurobi.Constr *Awaiting Model Update*>}
```

In [7]:

```python
# Solving the model
m.optimize()
```

```python
#  Print optimal solutions and optimal value
print("\n Optimal Solution :\n")
for i, v in enumerate(m.getVars()):
    print(v.VarName, v.x)

print('Obj:', m.objVal)
```

```
Gurobi Optimizer version 9.1.2 build v9.1.2rc0 (win64)
Thread count: 4 physical cores, 8 logical processors, using up to 8 threads
Optimize a model with 85 rows, 78 columns and 245 nonzeros
Model fingerprint: 0x1aa73404
Coefficient statistics:
  Matrix range     [4e-02, 1e+00]
  Objective range  [4e-02, 9e-01]
  Bounds range     [0e+00, 0e+00]
  RHS range        [6e+04, 5e+05]
Presolve removed 60 rows and 31 columns
Presolve time: 0.01s
Presolved: 25 rows, 47 columns, 117 nonzeros

Iteration    Objective       Primal Inf.    Dual Inf.      Time
       0    6.9306000e+31   1.200000e+31   6.930600e+01      0s
      35    3.9149748e+04   0.000000e+00   0.000000e+00      0s

Solved in 35 iterations and 0.01 seconds
Optimal objective  3.914974804e+04

 Optimal Solution :

quantity_produced_feedone[0] 0.0
quantity_produced_feedone[1] 96000.0
quantity_produced_feedone[2] 96000.0
quantity_produced_feedone[3] 96000.0
quantity_produced_feedone[4] 96000.0
quantity_produced_feedone[5] 96000.0
quantity_produced_feedone[6] 96000.0
quantity_produced_feedone[7] 96000.0
quantity_produced_feedone[8] 96000.0
quantity_produced_feedone[9] 94148.8370060367
quantity_produced_feedone[10] 96000.0
quantity_produced_feedone[11] 96000.0
quantity_produced_feedone[12] 96000.0
quantity_produced_feedtwo[0] 0.0
quantity_produced_feedtwo[1] 0.0
quantity_produced_feedtwo[2] 0.0
quantity_produced_feedtwo[3] 0.0
quantity_produced_feedtwo[4] 0.0
quantity_produced_feedtwo[5] 0.0
quantity_produced_feedtwo[6] 0.0
quantity_produced_feedtwo[7] 0.0
quantity_produced_feedtwo[8] 0.0
quantity_produced_feedtwo[9] 0.0
quantity_produced_feedtwo[10] 0.0
quantity_produced_feedtwo[11] 0.0
quantity_produced_feedtwo[12] 0.0
inventory_kept[0] 60000.0
```

```
inventory_kept[1] 81504.0
inventory_kept[2] 103008.0
inventory_kept[3] 0.0
inventory_kept[4] 0.0
inventory_kept[5] 0.0
inventory_kept[6] 0.0
inventory_kept[7] 0.0
inventory_kept[8] 0.0
inventory_kept[9] 0.0
inventory_kept[10] 16992.0
inventory_kept[11] 38496.0
inventory_kept[12] 60000.0
quantity_sold[0] 0.0
quantity_sold[1] 60000.0
quantity_sold[2] 60000.0
quantity_sold[3] 184512.0
quantity_sold[4] 81504.0
quantity_sold[5] 81504.0
quantity_sold[6] 81504.0
quantity_sold[7] 81504.0
quantity_sold[8] 81504.0
quantity_sold[9] 79932.36261812516
quantity_sold[10] 64512.0
quantity_sold[11] 60000.0
quantity_sold[12] 60000.0
quantity_feed_one[0] 0.0
quantity_feed_one[1] 211200.0
quantity_feed_one[2] 211200.0
quantity_feed_one[3] 211200.0
quantity_feed_one[4] 211200.0
quantity_feed_one[5] 211200.0
quantity_feed_one[6] 211200.0
quantity_feed_one[7] 211200.0
quantity_feed_one[8] 211200.0
quantity_feed_one[9] 207127.44141328076
quantity_feed_one[10] 211200.0
quantity_feed_one[11] 211200.0
quantity_feed_one[12] 211200.0
quantity_feed_two[0] 0.0
quantity_feed_two[1] 0.0
quantity_feed_two[2] 0.0
quantity_feed_two[3] 0.0
quantity_feed_two[4] 0.0
quantity_feed_two[5] 0.0
quantity_feed_two[6] 0.0
quantity_feed_two[7] 0.0
quantity_feed_two[8] 0.0
quantity_feed_two[9] 0.0
quantity_feed_two[10] 0.0
quantity_feed_two[11] 0.0
quantity_feed_two[12] 0.0
Obj: 39149.74804409826
```

In [8]:

```
solution = np.array(m.x)
len(solution)
solution = solution.reshape(6,13)
df = pd.DataFrame({'q_one':solution[0],
```

```
                 'q_two':solution[1],
                 'x':solution[2],
                 'd':solution[3],
                 'e_one':solution[4],
                 'e_two':solution[5]})
df
```

Out[8]:

	q_one	q_two	x	d	e_one	e_two
0	0.000000	0.0	60000.0	0.000000	0.000000	0.0
1	96000.000000	0.0	81504.0	60000.000000	211200.000000	0.0
2	96000.000000	0.0	103008.0	60000.000000	211200.000000	0.0
3	96000.000000	0.0	0.0	184512.000000	211200.000000	0.0
4	96000.000000	0.0	0.0	81504.000000	211200.000000	0.0
5	96000.000000	0.0	0.0	81504.000000	211200.000000	0.0
6	96000.000000	0.0	0.0	81504.000000	211200.000000	0.0
7	96000.000000	0.0	0.0	81504.000000	211200.000000	0.0
8	96000.000000	0.0	0.0	81504.000000	211200.000000	0.0
9	94148.837006	0.0	0.0	79932.362618	207127.441413	0.0
10	96000.000000	0.0	16992.0	64512.000000	211200.000000	0.0
11	96000.000000	0.0	38496.0	60000.000000	211200.000000	0.0
12	96000.000000	0.0	60000.0	60000.000000	211200.000000	0.0

In [ ]: