DBA5103
Operations Research and Analytics
Group Project – Team 11

Optimizing Pricing & Sourcing Strategy for Catfish Producers in the United States

Team members
A0125002E / Hpone Myat Khine
A0218968J / Mansi Agarwal
A0231857Y / Widya Gani Salim
A0231875Y / Tian Shi Xin Sophil
A0231887U / Donghwan Kim
A0231999L / Cristian Bojaca

Table of Content

1. Introduction

1.1 Background & Context

Catfish is the most important species in the United States' aquaculture industry, accounting for 21% of the entire industry with a total sale of $371 million[1] and supporting 8,004 jobs per annum in 2016[2]. Due to its adaptability and yield, catfish has always been a popular species to cultivate and consume worldwide, especially in Southeast Asia. The US catfish farming began to flourish in the early 1960s starting in the Deep South regions as farmers found that catfish rearing was more profitable than growing cotton and soybeans[3]. The sector quickly expanded in the 1970s and in the 1980s because of changes in the construction method that led to an increase in the productivity (Perez)[4.] Catfish farming also spearheaded the development of the aquaculture industry in US. Today, it is one of the primary sources of economic activity and employment in 9 states, including Mississippi, Arkansas, Texas and Alabama[5].

1.2 Problem Statement

In recent years, however, U.S catfish producers have faced tremendous difficulties due to poor harvests, flooding, high variable costs, and most eminently, foreign competition. With the relaxation of regulation in seafood trading, foreign competitors such as Vietnamese catfish producers have flooded the US market with their cheap products. This resulted in U.S producers having to compete by adopting new technology to boost their production and protect themselves by lobbying for trade protection from the government. Some of their efforts were successful: 1) they won an anti-dumping case against Vietnamese producers; 2) a labelling law whereby foreign catfish can only be labelled under a different product name were imposed, and 3) import tariffs were introduced against foreign producers[5]. Nonetheless, foreign producers still remain a serious threat to the survival of US domestic producers. With high price volatility and rising variable costs, determining production without optimizing profit or selling time is risky.

1.3 Key Objectives

Given the economic significance and impact of catfish production, this report focuses on (i) optimising time of sale and then (ii) optimising the production under the identified selling months to maximise the producers' profit. Much attention has been given into creating the best catfish rearing conditions for a farm, such as finding the best feed combinations (Nahar et al.)[6] (Abdel-Aziz et al.)[7] water temperature (Vasal and Sundararaj)[8] and fish to pond density (Ghate et al.)[9]. On the contrary, little attention has been given to optimising U.S catfish producers' selling time and production**.** This discrepancy is understandable from a business standpoint, since catfish producers operate in a market with perfect competition characteristics. This implies that they are price takers with limited bargaining power that face fierce competitions and low profit margins. Henceforth, there are limited focus on increasing production yield and operational efficiency since such analysis will at most bring short-term gains.

Nonetheless, we argue that a focus on longer term issues might give a higher return. For example, while catfish producers are price takers, they could look to maximise their profit by understanding the optimal selling months to negotiate better contractual terms with the intermediaries and can reduce their production cost by planning their cycle based on the optimal selling months. Uncertainties in the selling price and production cost are the two biggest risks that undermine U.S catfish producers' profit margin, resolving these issues could increase their competitiveness in the market.

1.4 Model Introduction

To resolve the issues discussed above, two key objectives and models are proposed:

1.4.1 Optimize producers' selling time using Game Theory (Model 1)

Game Theory is a decision-making technique that solves the problems of competition where a

conflict of interest occurs among the decision-makers or where uncertainties are rampant. It is commonly adopted in many different fields, including in agriculture (Sexton)[10] (Brandenburger and Nalebuff)[11] that operate under perfect competition markets. For instance, Game Theory approach was used to analyse international trade of dairy products based on the utility functions of different countries (Muller)[12] and the price of rice trading price in the USA, Japan and South Korea (Lee and Kenedy)[13]. The purpose of adopting Game Theory in this paper is to find the optimum selling time for producers based on the market producers' selling price and their risk appetite. The different games adopted will be discussed in more detail in ***Section 4.1***.

1.4.2 Maximize profit by optimizing production through LP (Model 2)

A linear programming (LP) optimization model is often adopted by (Prišenk et al.)[14] to decrease production costs with different field types in Slovenia. Although one of the findings is that LP provides a solution for the problem, other techniques are explored, such as weighted goal programming. The purpose of adopting LP optimization in this report is to find the maximum profit that the producers can attain under different optimum selling times discovered from the game theory model, given their production capacity and budget constraints. The model will be discussed in more detail in ***Section 4.2***.

2. Catfish Producers' Supply Chain & Production Risks

The assumptions that are adopted for our models are derived from our research and understanding of the catfish farms setup and supply chain which are discussed in this section.

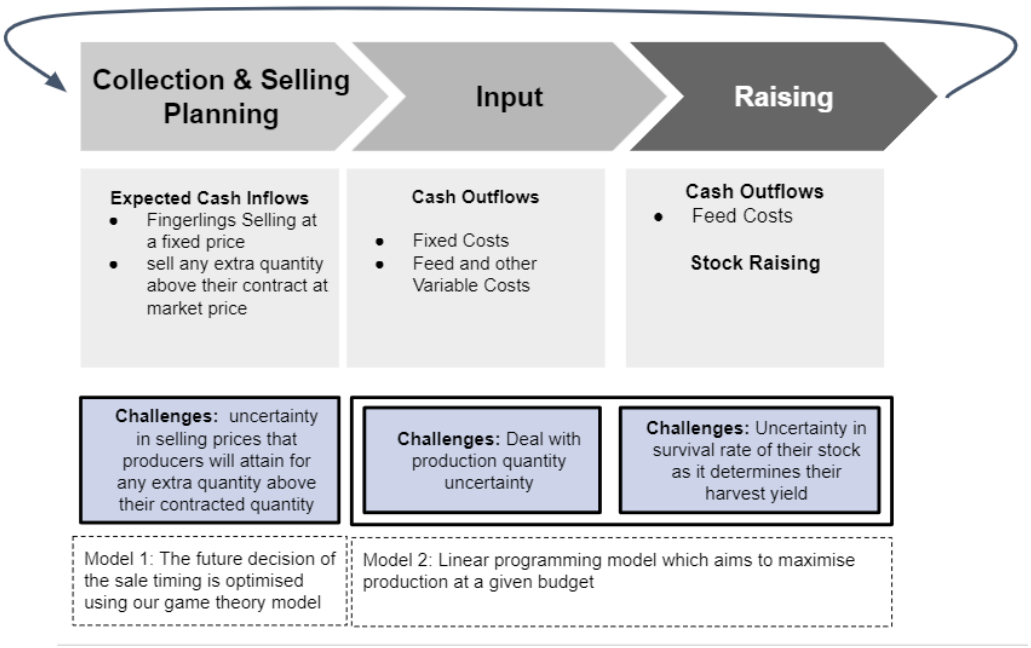
2.1. Producers' Supply Chain

Much like agriculture, aquaculture supply chains are complex and non-transparent; middlemen have the ability to impose significant markups to end consumers. However, this ability is non-transferrable to producers as they are only involved in the early, non-consumer facing phases

of the supply chain. As this paper solely focuses on the risks and uncertainties faced by producers, we will focus on the supply chain phases in which producers are involved.

Given that the producers are involved mainly in the upstream of aquaculture supply chain, we will focus on three main phases of producers (refer to ***Figure 1***).

Figure 1: Three phases that producers face



- **Input Phase** - In this phase, catfish producers invest in the necessary equipment and fish stock to produce, incurring all the fixed costs (e.g. building ponds, land lease) and some of the variable costs (e.g. fingerling stocks, seining & hauling). The main uncertainties faced by the producers during the Input Phase is the production quantity, as it affects the variable costs incurred. The scope of this paper covers only producers who have established farms that is 256 acres, and so, fixed costs are assumed to be constant.

- **Raising Phase** - Once fingerling stocks are determined in the input phase; producers tend to them in the raising phase. During the Raising Phase, the main uncertainties and risks that producers face are the variable costs needed to grow the catfish and the survival rate of their stock as it affects their harvest yield. The largest variable cost in this phase is feed costs; this

is taken into account in our Model 2 which aims to maximise production at a given budget.

- **Collection Phase** - Once fingerlings mature, they will be harvested and sold to traders and processors. Catfish producers regularly have contractual agreements with traders and processors (ie. intermediaries), who will purchase the contracted quantity at a fixed price. The contracted quantities however, are usually lower than the maximum quantity that the producers can supply. If the producers are able to understand the demand and market pricing prior to their negotiation of contract with the intermediaries, they can negotiate to include a contractual clause to charge the intermediaries a higher market price if they can produce more catfish to meet the surge in market demand above their contracted quantity (but the producers will not disclose this pricing strategy to the intermediaries to increase their profit).

  The main uncertainties and risks during the Collection Phase are the timing for the producers to produce more than the contracted quantity. The decision of the sale timing is optimised in our game theory model (Model 1). In parallel, in our LP model (Model 2), we considered the contracted quantities as a minimum quantity that a producer needs to produce each month.

2.2 Farming Setup

We discovered that in the U.S Catfish industry are dominated by medium-sized catfish farms. There are approximately 650 medium-sized catfish farms with an average of $673,719/year production budget, located primarily in the Southern states. To formulate our model, we took reference from Arkansas Catfish Production Budgets (Engle)[15]. The medium sized farms averaged around 256 acres, and all fixed and variable costs are derived accordingly. A medium-sized farm setup costs is presented in ***Figure 2***.

Figure 2: Cost of a medium-sized farm

Medium-sized (256 acres) Farm Setup			
Type of Cost	Constraints	Description	Costs
	Equipment	Tractors,feed bin, aerators, generator	$38,757
	Land & Construction	Cost of construction	$61,632
Fixed	Labour	2.5 hired help for medium farm*	$50,700
	Miscelleanous	Others such as insurance	$8,039
	Fixed Cost/Month		$13,260.67
			Costs per pound
	Chemical	Medicine, chemical to treat	$0.0032
	Repairs & Maintenance	Maintenance of ponds	$0.0229
Variable	Utilites	Electricity, diesel	$0.0657
	Seining & Hauling	See below **	$0.0500
	Fingerlings Stock	Cost of Fingerlings	$0.0632
	Variable Cost/Month		$0.205
	---	**Feed not considered in computation**	----
	Feeds	Cost of purchase	$0.105 - $0.115

*Average of 2.5 hired (paid) help for a medium sized farm (0.5 for part-time or seasonal helper); Additional 1 helper from farmer/farmer's family member. Labour was derived from a 10-hour work day for a total of 261 working days.

**Seining is the usage of a long net with weights on one end so that it hangs vertically in the water and is used to trap the catfishes. It is assumed here that the farms would hire crews to assist with the seining & hauling which is reflected accordingly in the cost.

Given the variability of feed prices and its role in total costs of the firm, it is not included in the set-up variable costs. Instead, the feed will be modelled as a separate variable cost to determine the optimal quantity and cost in our LP model.

2.3 Farming Risks and Uncertainties

Aggregating our understanding of the catfish supply chain and farm setup, we have summarized the major risks and uncertainties that producers face:

- **Pricing Seasonality** - Producers face high selling price volatility as demand is cyclical and have limited information about the market to negotiate with the intermediaries.
- **Fish Feed Costs** - Feed costs account for around 45% of total variable costs (Engle)[15] and is thus a significant source of risk and uncertainty. Different feed quality also affects the survivability of the catfish. Hence, producers need to determine the best feed source that balances between costs and survival rate.
- **Production Quantity** - Producers need to determine their production quantity as it will

determine the variable costs that they will incur and affect the profit that they will gain.

The consideration of these risks are accounted for in our models. The aim of our game theory model (Model 1) is to overcome the risks of pricing seasonality. Our LP model (Model 2) will address the risks caused by the uncertainties of production quantity and fish feed costs.

3. Dataset and Chosen Timeline

Our dataset contains catfish domestic sales and inventory in the U.S - both nationwide and within different states - from 1986 to 2013, according to the U.S Department of Agriculture Research Service. Based on our observation of the dataset and business objective, we chose to focus on an 8-year period from 2004 to 2012 in this paper. The period suits our business objectives since during this period, U.S catfish producers experienced high price and production volatility. At the beginning of the period, from 2004 to 2007, the producers were recovering from the entrance of foreign competitors. From 2007 to 2010, U.S catfish producers' production and selling price stabilized as they adjusted their strategy according to the market environment. At the end of the period, U.S catfish producers were able to fight back, as shown by the jump in selling price, by seeking government protection. A graph showing the price volatility in this period can be found in the appendix (Graph 1). Since 2012, U.S producers' catfish selling price has remained relatively stable, and thus, the results attained in this paper remains relevant. We also calculated the percentage deviation of catfish selling price for each month, based on its annual mean, to calculate the lower bound and the upper bound of catfish selling prices for different game theory scenarios. More details will be provided in ***Section 4.1***.

4. Methodology

4.1 Model 1 - Optimise Selling Time with Game Theory

The main objective of the Model 1 is to determine producers' optimum selling months to

maximize expected gains by taking into account the producers' selling price fluctuations and risk appetite. This model is based on Game Theory, which is utilised to solve problems on decision making under uncertainties.

Some key assumptions were made:

- Market is neutral and indifferent towards the producers' actions.
- Production patterns of producers will change according to their risk appetites.
- There are major external factors which are uncontrollable by the producers.

We employed five criteria of Game Theory which we believe sufficiently capture the different producers' behaviours and risk appetite: Wald's (Maximin) (Pessimistic) Criterion, Laplace Criterion, Hurwicz Criterion, Benefit Criterion and Wald's (Maximax) (Optimistic) Criterion. Each criterion uses different catfish price matrices to reflect the different risk appetites of the producers. The base mathematical formulation of our game theory models is represented in Figure 3 below.

Figure 3: Mathematical formulation base for our game theory models

Introduce an auxiliary variable Z, V:

$$\textbf{Max Z = V}$$

s.t.

$$\text{[Expected Gain Constraints]} \quad V \leqq \sum_{i=1,j=1}^{m,n} a_{ij} p_j$$

$$\text{[Total Proportion Constraint]} \quad \sum_{j=1}^{n} p_j = 1$$

$$p_j \geqq 0$$

$$i = 1,2,\ldots,m$$

$$j = 1,2,\ldots,n$$

a_{ij}= price per lb matrix

where i = years, j = months

p_j =proportion of sales in each month

4.1.1 Wald's (Maximin) Criterion

In Wald's criterion, producers adopt a pessimistic approach and believe that market conditions

will deteriorate and prices will decrease. Thus, they choose to avert risks and seek certainty. This translates to a maximin condition whereby producers take the best (max) price out of the worst (min) conditions where they expect the price to be lowest. By choosing the worst (min), the price is lower but this in turn also lowers the risk. By then taking the best (max) possible out of this, they are guaranteed a certain payout. It can be modelled generally following the mathematical formulae shown in ***Figure 3*** and is represented below by ***Figure 4*** where the a_{ij} matrices in the expected gain constraints follow the monthly prices in our chosen period. Our monthly price list (a_{ij}) can be also be found in ***Appendix III Table 14***.

Figure 4: Mathematical formulation for based on Wald's (Maximin) Criterion

Introduce an auxiliary variable Z, V:
Z = Expected gains

$$\textbf{Max Z = V}$$

s.t.

$$V \leqq \sum_{i=1,j=1}^{m,n} a_{ij}p_j$$

$$\sum_{j=1}^{n} p_j = 1$$

$$p_j \geqq 0$$

$$i = 1,2,\ldots,m$$

$$j = 1,2,\ldots,n$$

a_{ij}= price per lb matrix

where i = years, j = months

p_j =proportion of sales in each month

By including our data, it can be shown as:

$$\begin{aligned}
&52.8p_1 + 59.9p_2 + 66p_3 + 70.6p_4 + 70.7p_5 + 66p_6 + 58.9p_7 + 55.9p_8 + 55.5p_9 + 57.2p_{10} + 58.1p_{11} + 58p_{12} - V \geq 0\\
&57.3p_1 + 62.3p_2 + 66.9p_3 + 70.3p_4 + 70.9p_5 + 69p_6 + 62.5p_7 + 59.2p_8 + 58.9p_9 + 59.6p_{10} + 61p_{11} + 61p_{12} - V \geq 0\\
&57.5p_1 + 62.1p_2 + 68p_3 + 76.1p_4 + 78.1p_5 + 77.3p_6 + 70.2p_7 + 66.3p_8 + 67.6p_9 + 68.8p_{10} + 70.5p_{11} + 70.4p_{12} - V \geq 0\\
&66.2p_1 + 71.4p_2 + 76.5p_3 + 81.5p_4 + 82.4p_5 + 78.2p_6 + 65.8p_7 + 59.8p_8 + 56.7p_9 + 56.1p_{10} + 56.1p_{11} + 54.6p_{12} - V \geq 0\\
&52p_1 + 58.6p_2 + 67.9p_3 + 73.4p_4 + 76.2p_5 + 76p_6 + 70.7p_7 + 67.7p_8 + 67.2p_9 + 67.9p_{10} + 69.3p_{11} + 69p_{12} - V \geq 0\\
&64.1p_1 + 65.6p_2 + 70.6p_3 + 74p_4 + 74.8p_5 + 73.1p_6 + 66.6p_7 + 62.9p_8 + 62.8p_9 + 63.2p_{10} + 64.5p_{11} + 64.1p_{12} - V \geq 0\\
&60.4p_1 + 65.2p_2 + 71.7p_3 + 77.9p_4 + 78.1p_5 + 75.3p_6 + 68.1p_7 + 64.6p_8 + 66.3p_9 + 68.5p_{10} + 70.9p_{11} + 72.3p_{12} - V \geq 0\\
&73.6p_1 + 85.5p_2 + 98.2p_3 + 110.6p_4 + 114.7p_5 + 117.9p_6 + 108.2p_7 + 104.5p_8 + 103.6p_9 + 103.9p_{10} + 105.9p_{11} + 105.1p_{12} - V \geq 0\\
&98.7p_1 + 104.7p_2 + 109.7p_3 + 113.1p_4 + 101.9p_5 + 89.4p_6 + 72.8p_7 + 65.3p_8 + 64.5p_9 + 66.1p_{10} + 69.3p_{11} + 69.7p_{12} - V \geq 0
\end{aligned}$$

$$p_1 + p_2 + p_3 + p_4 + p_5 + p_6 + p_7 + p_8 + p_9 + p_{10} + p_{11} + p_{12} = 1$$

4.1.2 Laplace Criterion

Laplace's criterion posits that when there is insufficient information on likelihood of events to occur then it is reasonable to assume that they are all likely to occur. For the producers, this translates to them taking an 'indecisive' perspective on the outlook of the catfish industry. In this scenario, the producers assume that there is an equal chance of them getting both the best price and worst price. The formulation of this criterion was also based on ***Figure 3***. However, since the producers assume equal chances, a_{ij} is derived from the sum of 50% of the upper bound prices and 50% of the lower bound prices and is shown in ***Appendix III Table 11***. Refer to ***Appendix II Figure 6***.

4.1.3 Hurwicz criterion

Also known as the criterion of realism, Hurwicz criterion calculates the weighted average between the best (optimistic) price received and the worst (pessimistic). This is done through selecting a parameter α, which represents the 'degree' of optimism. Much like the real world, this seeks to strike a trade-off between both outcomes. In our model, α of 0.8 was chosen as the US government had imposed tariffs to protect its aquaculture industry, and so, producers would be cautiously optimistic. It is a subjective criterion as choosing of α is needed. The formulation too was based on ***Figure 3***. The a_{ij} matrix here is similar to Laplace's with the exception being that it is obtained by taking the summation of the upper bound – price and (1-0.8) of the lower bound price and is depicted in ***Appendix III Table 12***. Refer to ***Appendix II Figure 7***.

4.1.4 Benefit Criterion

The benefit criterion is geared towards producers who are interested in maximizing benefits and are willing to take risks. For the benefit criterion, the matrix a_{ij} is computed based on the original producers' prices where there is no adjustment made for the pessimistic and optimistic scenarios. As such, the returns from this criterion will be lower than those from the Wald's Maximax (Optimistic) criterion because the matrix for Wald's Maximax is computed based on

fully optimistic scenario. The formulation is similarly based on ***Figure 3***. The a_{ij} matrix is obtained by first choosing the worst payoff for each year. This value is then subtracted from the other values (in the same year) to get a comparison of how much better each payoff is. This forms the benefit matrix which is shown in ***Appendix III Table 13***. Refer to ***Appendix II Figure8***.

4.1.5 Wald's Maximax (Optimistic) Criterion

As opposed to Wald's Minimax (Pessimistic) Criterion, Wald's Maximax (Optimistic) Criterion assumes the producers perceive a favourable market and thus is willing to take risk. The producers receive maximum payout in an event of a favourable outcome. This thus entails a Maximax condition where we take the best (max) price out of the best (max) condition. The formulation follows ***Figure 3***; The a_{ij} price matrix follows the optimistic monthly prices and can be found in ***Appendix III Table 10.*** Refer to ***Appendix II Figure 9***.

4.2 Model 2 - Optimise Profit with Linear Programming

The main objective of this model is to maximise the producers' profit by optimising production quantity considering the optimum selling time and price that they receive from Model 1, depending on their risk appetite. This model is formulated based on the farm setup and budget of a medium sized farm at 256 acres discussed in ***Section 2.2***. This is based on the Arkansas catfish farm guidelines to reflect a realistic overview of a catfish farm in the US. This model addresses the two main uncertainties and risks that catfish producers face: production quantity and feed costs. Feed imposes risks to producers due to its costs and quality, which affects survival rate and thus, yield. The Feed Conversion Ratio (FCR) is the amount of feed needed to grow a pound of fish. Hence a lower ratio implies fewer feed needed. The feed costs risk is introduced by giving producers a choice between two feeds of different price and survival rate.

This model is formulated based on the following constraints:

- **Inventory Constraint** captures the producers' inventory capacity. Harvested catfish can be kept in an inventory indefinitely, as a flash freeze method is adopted at a cost of $0.036/lb. All unsold harvest will be kept as inventory; the farm is assumed to have no inventory limit or they can store at storage place at a cost of $0.036/lb.
- **Contractual Constraint** assumes that producers have contractual agreements that obliged them to produce and sell at a minimum quantity per month. We assumed that quantity is 60,000lbs/month based on our research. When fulfilling the contracted quantity, producers sell at a fixed guaranteed price, which we derived from the Wald's pessimistic model, giving $0.7066/lb.
- **Production Constraint** reflects the producers' monthly maximum production capacity at 96,000lbs/month which is limited by the size of the producers' farm.
- **Budget Constraint** assumes the producers' fixed annual budget is equal to $673,719 where it has to be used for fixed cost, variable cost (excluding feed cost), feed cost and holding cost.
- **Feed 1 Constraint** assumes that quantity of feed 1 is based on the quantity of catfish produced from the feed 1 and the FCR of 2.2
- **Feed 2 Constraint** assumes that quantity of feed 2 is based on the quantity of catfish produced from the feed 2 and the FCR of 2.2
- **Survival Constraint** stipulates that the survival rate arising from the two different feeds used must equal to 80% as this is part of the industry standard.
- **Planning Constraints** set the cycle and initial assumptions:

  There is no production, sales, fixed and variable cost incurred in t=0 but there will be an inventory of 60,000 pound of catfish from the previous cycle and there will be at least an inventory of 60,000 in t = 13 to meet the minimum quantity of catfish for the next cycle.

Please refer to ***Figure 5*** and ***Table 3*** below for the summary of all variables and mathematical

Figure 5: Mathematical formulation for Model 2

Decision Variables: $q_{1t}, q_{2t}, x_t, d_t. e_{1t}, e_{2t}$

$$Max \sum p_{1t} \times Prod_{min,t} + \sum p_{2t} \times \max\Big(\big(d_t - Prod_{min,t}\big), 0\Big) - 12 \times f_t$$
$$-\sum v_t \times (q_{1t} + q_{2t}) - \sum pe_{1t} \times e_{1t} + \sum pe_{2t} \times e_{2t} - \sum h_t \times x_t$$

s.t

$x_t = sr_1 \times q_{1t} + sr_2 \times q_{2t} + x_{t-1} - d_t$	**Inventory Constraint**
$d_t \geq Prod_{min,t}$	**Contractual Constraint**
$q_{1t} + q_{2t} \leq Prod_{max,t}$	**Production Constraint**
$\sum pe_{1t} \times e_{1t} + \sum pe_{2t} \times e_{2t} \leq Budget - 12 \times f\ - \sum v\_t\ \times (q_{1t} + q_{2t} - \sum h_t \times x_t$	**Budget Constraint**
$\frac{e_{1t}}{fcr} \geq q_{1t}$	**Feed Constraint**
$\frac{e_{2t}}{fcr} \geq q_{2t}$	**Feed 2 Constraint**
$sr_1 \times q_{1t} + sr_2 \times q_{2t} \geq 0.8(e_{1t} + e_{2t})$	**Survival Constraint (from Feeds)**
$q_{10} = 0$ $q_{20} = 0$ $x_0 = 60{,}000$ $d_0 = 0$ $e_{10} = 0$ $e_{20} = 0$ $x_{12} \geq 60{,}000$	**Inventory for Next Cycle**
$q_{1t}, q_{2t}, x_t, d_t, e_{1t}, e_{2t} \geq 0$	**Non-Negative Constraints**

Table 3 – Explanation of Variables used in Model 2

Variables	Variable Name in Model	Values
Maximum pound of catfish that the farm can produce in each month t (lb)	$\mathrm{Prod}_{\max}$	= 96,000
Minimum pound of catfish that the farm commits to sell to the intermediaries in each month t (lb)	$\mathrm{Prod}_{\min}$	= 0 when t = 0 (planning month) = 60,000 otherwise
Maximum production cost that the farm can incur in 1 year (lb)	Budget	= 673,719
Feed conversion rate (ratio)	fcr	= 2.2
Mean survival rate (ratio)	sr_1 sr_2	= 0.8490 for feed 1 = 0.9052 for feed 2
Holding cost per pound of catfish that is kept as inventory ($/lb)	h_t	= US$0.036
Variable cost per pound of catfish (exclude feed cost) ($/lb)	v_t	= 0 when t = 0 = 0.2051444222
Fixed cost incurred in each month t ($)	f_t	= 13,260.67
Price of each pound of feed 1 ($/lb)	$\mathrm{pe}_{1\mathrm{t}}$	= 0 when t =0 = 0.105 otherwise
Price of each pound of feed 2 ($/lb)	$\mathrm{p}e_{2t}$	= 0 when t =0 = 0.115 otherwise
Price of each pound of catfish based on the minimum contracted quantity in month t ($/lb) (from Model 1)	p_{1t}	= 0 when t = 0 = 0.70665 otherwise
Price for each pound of catfish if produced beyond the minimum contracted quantity in month t ($/lb) (from Model 1)	p_{2t}	= 0.7852 when t = 2, 3 (eg. using Laplace)1 = 0.70665 otherwise
Quantity of catfish produced using feed 1 in month t (lb)	q_{1t}	Decision variable but q_one = 0 when t = 0
Quantity of catfish produced using feed 2 in month t (lb)	q_{2t}	Decision variable but q_two = 0 when t = 0
Quantity of catfish kept as inventory in month t (lb)	x_t	Decision variable but x = 60,000 when t = 0 x >= 60,000 when t = 12
Quantity of catfish that the producer sells to the intermediaries in month t (lb) Assume that there is no demand cap from the intermediaries	d_t	Decision variable but d = 0 when t = 0
Quantity of feed 1 used in each month t (lb)	e_{1t}	Decision variable
Quantity of feed 2 used in each month t (lb)	$\mathrm{e}_{2\mathrm{t}}$	Decision variable
Time	t	= {0, 1, 2, …, 13}

5. Results

5.1 Results from Model 1

The solution from Model 1 will enable U.S catfish producers to determine the optimal selling time at different potential monthly catfish price ($/lb) based on their differing risk appetite, as represented by different criteria of Game Theory criteria. As shown in **Table 1**, the difference between the most pessimistic (i.e: Wald Pessimistic) and the most optimistic (i.e: Wald Optimistic) is a potential earning of 20.61%. Most producers' risk appetite would lie within the range provided by Laplace and Benefit criteria, hence these criteria are instrumental in mirroring producers' responses and behaviours in actual market conditions. For example, producers striking a 50-50 balance between risk and certainty are impersonated by Laplace criterion. Meanwhile, producers who are willing to take more risk than Laplace producers can either follow the Hurwicz criterion or the Benefit criterion to receive either 16.29% or 16.43% more return than the pessimistic criterion respectively.

Table 1: Summary of optimal selling months and average prices from Model 1 (ordered based on the risk aptitude of producers)

Criterion	Optimal months	Proportion of Total Quantity Sold	Average prices ($/lb)	Improvement (%)
Wald (Pessimistic) **used as base**	May	100%	0.7067	-
Laplace	February March	23% 77%	0.7853	10.01%
Hurwicz	March	100%	0.8442	16.29%

Benefit	January March	2% 98%	0.8456	16.43%
Wald (Optimistic)	March	199%	0.8901	20.61%

5.2 Results from Model 2

Realistically, it is challenging for producers to sell their produce only during the optimal months that are identified in Model 1 as intermediaries will still require produces for the other months to meet monthly demand. As such, to leverage on the analysis from Model 1, producers should strive to sell more during the optimal months identified from Model 1 and negotiate with their intermediaries to have a clause for higher price for their produce if they produce beyond the minimum contractual quantity to strive to maximise profit.

With reference to ***Table 2***, producers will in general, earn higher profit if they are willing to undertake more risk. The producers will earn a minimum profit of $23,253.68 (based on Wald (Pessimistic) criterion) and a maximum profit of $39,780.48 (based on Benefit criterion). Refer to ***Appendix II*** for the decision variables for each criterion.

Table 2: Summary of profit (from Model 2)

Criterion	**Profit (US$)**
Wald (Pessimistic) <u>used as base</u>	23,253.68
Laplace	30,099.95

Hurwicz	33,422.20
Benefit	39,780.48
Wald (Optimistic)	39,149.75

6. Conclusion

The findings in this paper enable U.S catfish producers to choose the criterion of Game Theory in line with their risk aptitude. The price of catfish ranges from US$0.7067/lb to US$0.8901/lb and the number of optimal selling months is capped at two months (refer to ***Section 5.1***). Henceforth, the producers will be able to plan their production and negotiate with intermediaries for more optimal contractual prices in hope of achieving higher profit, which ranges from US$23,253.68 to US$39,780.48 (refer to ***Section 5.2***). The producers should update both models annually to develop their pricing strategies for each production cycle in order to maintain their bargaining power to negotiate better contractual terms with the intermediaries and plan their production cycle. Arising from the assumptions and limitations that we have discussed under ***Section 2***, more studies can be undertaken to consider the uncertainties in the survival rates from each type of feed. This implies that one can construct the Model 2 by incorporating the duality by transforming the "inventory" and "survival from feed" constraints into 13 sets of duality equations.

Appendix 0: Sources & References Cited

1) AgFax. "Catfish Production: Value of Sales, Water Surface Acres Down." *AgFax*, 10 February 2021, https://agfax.com/2021/02/10/catfish-production-value-of-sales-water-surface-acres-down/. Accessed 19 November 2021.

2) Mississippi State University. "Catfish Aquaculture Production, Farm-gate Values and Prices and Economic Contributions | Coastal R&E." *Coastal Research & Extension Center*, 2017, https://coastal.msstate.edu/aquaculture-catfish. Accessed 19 November 2021.

3) Hanson, Terrill R. "Catfish Farming in Mississippi - 2006-04." *Mississippi History Now*, 2006, https://mshistorynow.mdah.ms.gov/issue/catfish-farming-in-mississippi. Accessed 19 November 2021.

4) Perez, Karni. "Catfish Industry." Encyclopedia of Alabama, 11 December 2007, http://encyclopediaofalabama.org/article/h-1409. Accessed 21 November 2021.

5) Gro Inteligence. "US Gets Hooked on Vietnamese Catfish." Gro Intelligence, 14 October 2016, https://gro-intelligence.com/insights/articles/us-vietnam-catfish-production. Accessed 19 November 2021.

6) Nahar, et al. "Effect of different feeds on growth, survival and production of African catfish (Clarias gariepinus Burchell)." *Bangladesh Journal of Fisheries Research*, vol. 4, 2000

7) Abdel-Aziz, Mohamed Fathy Aid, et al. "Assessing the effect of different feeding frequencies combined with stocking density, initial weight, and dietary protein ratio on the growth performance of tilapia, catfish and carp." *Scientific African*, vol. 12, 2021.

8) Vasal, Sudhir, and Bangalore I. Sundararaj. "Thermal tolerance and preference of the Indian catfish Heteropneustes fossilis." *Environmental Biology of Fishes*, vol. 3, no. 3, 1978.

9) Ghate, Suhas R., et al. "Water quality in catfish ponds subjected to high stocking density selective harvesting production practice." *Aquacultural Engineering*, vol. 12, no. 3, 1993, pp. 169-181.

10) Sexton, Richard. "A Survey of Noncooperative Game Theory with Reference to Agricultural Markets: Part 1. Theoretical Concept." *Review of Marketing and Agricultural Economics*, vol. 62, no. 2, 1994, pp. 183-200. *Research Gate*, https://www.researchgate.net/publication/23945835_A_Survey_of_Noncooperative_Game_Theory_with_Reference_to_Agricultural_Markets_Part_1_Theoretical_Concepts.

11) Brandenburger, Adam, and Barry Nalebuff. "Right Game: Use Game Theory to Shape Strategy." *Harvard Business Review*, vol. 73, no. 4, 1995, pp. 55-71.

12) Muller, B. "Price competition among dairies: a theoretical approach based on Game Theory." *Kieler Milchwirtschaftliche Forschungsberichte*, vol. 51, no. 2, 1999, pp. 165-190.

13) Lee, and Kenedy. "A political economic analysis of US rice export programs to Japan and South Korea: a Game theoretic approach."" *American Journal of Agricultural Economics*, vol. 89, no. 1, 2007, pp. 104-115.

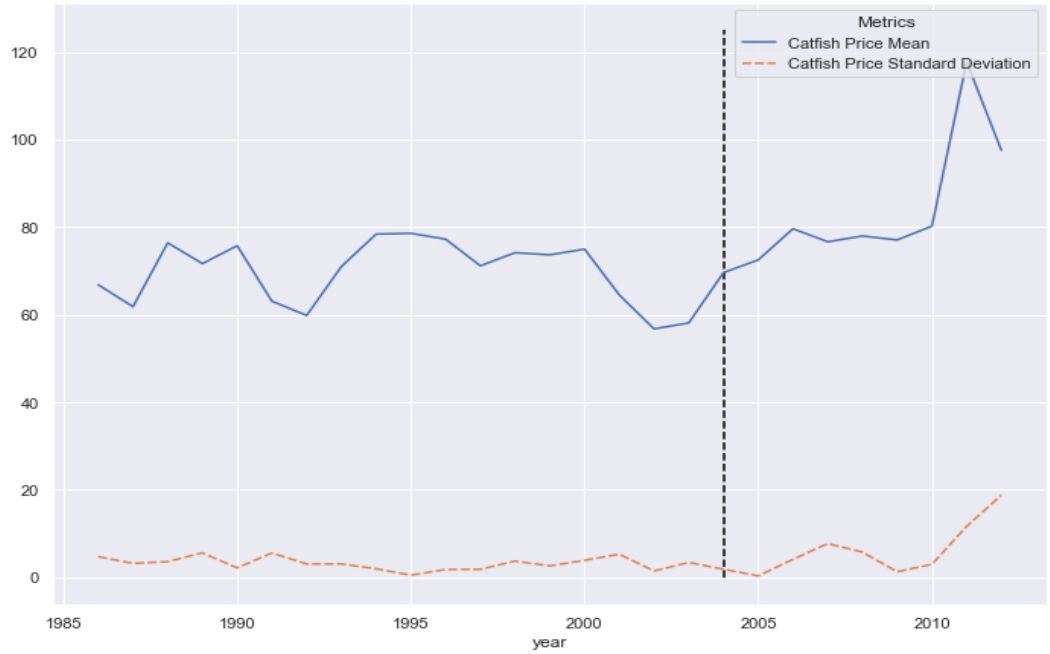
14) Prišenk, Jernej, et al. "Advantages of combining linear programming and weighted goal programming for agriculture application." *Operational Research*, vol. 14, 2014, pp. 253–260.

15) Engle, Carole, et al. "Economic history of U.S. catfish farming: Lessons for growth and development of aquaculture." *Aquaculture Economics and Management*, vol. 1, no. 1, 10/03/21, p. 35

16) US Department of Agriculture. "Catfish species." *Ask USDA*, 17 July 2019, https://ask.usda.gov/s/article/Catfish-species. Accessed 20 November 2021.

Appendix I: Pricing Graph from Dataset

Graph 1: Catfish Price Mean and Standard Deviation (1986 – 2012)



Appendix II: Mathematical Formulation for Game Theory Criterion (Laplace, Hurwicz, Benefit and Wald's Maximax (Optimistic) Criterion

Figure 6: Mathematical formulation for based on Laplace Criterion

Introduce an auxiliary variable Z,V:

$$\textbf{Max Z = V}$$

s.t.

$$V \leqq \sum_{i=1,j=1}^{m,n} a_{ij}p_j$$

$$\sum_{j=1}^{n} p_j = 1$$

$$p_j \geqq 0$$

$$i = 1,2,\ldots,m$$

$$j = 1,2,\ldots,n$$

a_{ij}= price per lb matrix

where i = years, j = months

p_j =proportion of sales in each month

By including our data, it can be shown as:

$$\begin{array}{l}
69.1\,p_1 + 74.2\,p_2 + 77.5\,p_3 + 78.8\,p_4 + 75.5\,p_5 + 69.7\,p_6 + 65.7\,p_7 + 65\,p_8 + 64.8\,p_9 + 65.9\,p_{10} + 65.8\,p_{11} + 66.3\,p_{12} - V >= 0 \\
75\,p_1 + 77.2\,p_2 + 78.6\,p_3 + 78.5\,p_4 + 75.7\,p_5 + 72.9\,p_6 + 69.7\,p_7 + 68.9\,p_8 + 68.6\,p_9 + 68.6\,p_{10} + 69.2\,p_{11} + 69.8\,p_{12} - V >= 0 \\
75.3\,p_1 + 77\,p_2 + 79.9\,p_3 + 85\,p_4 + 83.5\,p_5 + 81.6\,p_6 + 78.3\,p_7 + 77.2\,p_8 + 78.9\,p_9 + 79.2\,p_{10} + 80\,p_{11} + 80.6\,p_{12} - V >= 0 \\
86.6\,p_1 + 88.5\,p_2 + 89.9\,p_3 + 91.1\,p_4 + 88.1\,p_5 + 82.6\,p_6 + 73.5\,p_7 + 69.6\,p_8 + 66.1\,p_9 + 64.6\,p_{10} + 63.6\,p_{11} + 62.5\,p_{12} - V >= 0 \\
68.1\,p_1 + 72.7\,p_2 + 79.7\,p_3 + 82\,p_4 + 81.4\,p_5 + 80.3\,p_6 + 78.9\,p_7 + 78.7\,p_8 + 78.4\,p_9 + 78.2\,p_{10} + 78.6\,p_{11} + 78.9\,p_{12} - V >= 0 \\
83.8\,p_1 + 81.3\,p_2 + 82.9\,p_3 + 82.6\,p_4 + 79.9\,p_5 + 77.2\,p_6 + 74.3\,p_7 + 73.2\,p_8 + 73.2\,p_9 + 72.8\,p_{10} + 73.1\,p_{11} + 73.3\,p_{12} - V >= 0 \\
79.1\,p_1 + 80.8\,p_2 + 84.2\,p_3 + 87.1\,p_4 + 83.5\,p_5 + 79.5\,p_6 + 76\,p_7 + 75.2\,p_8 + 77.4\,p_9 + 78.8\,p_{10} + 80.3\,p_{11} + 82.8\,p_{12} - V >= 0 \\
96.4\,p_1 + 105.9\,p_2 + 115.3\,p_3 + 123.6\,p_4 + 122.6\,p_5 + 124.5\,p_6 + 120.7\,p_7 + 121.5\,p_8 + 120.9\,p_9 + 119.6\,p_{10} + 120.1\,p_{11} + 120.3\,p_{12} - V >= 0 \\
129.2\,p_1 + 129.8\,p_2 + 128.8\,p_3 + 126.4\,p_4 + 108.9\,p_5 + 94.5\,p_6 + 81.3\,p_7 + 75.9\,p_8 + 75.2\,p_9 + 76.1\,p_{10} + 78.5\,p_{11} + 79.8\,p_{12} - V >= 0
\end{array}$$

$$p_1 + p_2 + p_3 + p_4 + p_5 + p_6 + p_7 + p_8 + p_9 + p_{10} + p_{11} + p_{12} = 1$$

Figure 7: Mathematical formulation for based on Hurwicz's Criterion

Introduce an auxiliary variable Z,V:

$$\textbf{Max Z = V}$$

s.t.

$$V \leqq \sum_{i=1,j=1}^{m,n} a_{ij}p_j$$

$$\sum_{j=1}^{n} p_j = 1$$

$$p_j \geqq 0$$

$$i = 1,2,\ldots,m$$

$$j = 1,2,\ldots,n$$

a_{ij}= price per lb matrix

where i = years, j = months

p_j =proportion of sales in each month

By including our data, it can be shown as:

$$
\begin{aligned}
&78.9p_1 + 82.8p_2 + 84.4p_3 + 83.8p_4 + 78.4p_5 + 71.9p_6 + 69.8p_7 + 70.5p_8 + 70.3p_9 + 71.1p_{10} + 70.5p_{11} + 71.3p_{12} - V >= 0\\
&85.7p_1 + 86.1p_2 + 85.6p_3 + 83.4p_4 + 78.6p_5 + 75.3p_6 + 74p_7 + 74.7p_8 + 74.5p_9 + 74p_{10} + 74.1p_{11} + 75.1p_{12} - V >= 0\\
&85.9p_1 + 85.9p_2 + 87p_3 + 90.3p_4 + 86.7p_5 + 84.3p_6 + 83.1p_7 + 83.7p_8 + 85.6p_9 + 85.5p_{10} + 85.6p_{11} + 86.6p_{12} - V >= 0\\
&98.9p_1 + 98.7p_2 + 97.8p_3 + 96.8p_4 + 91.5p_5 + 85.3p_6 + 78p_7 + 75.4p_8 + 71.7p_9 + 69.7p_{10} + 68.1p_{11} + 67.2p_{12} - V >= 0\\
&77.8p_1 + 81.1p_2 + 86.8p_3 + 87.1p_4 + 84.5p_5 + 82.9p_6 + 83.7p_7 + 85.3p_8 + 85.1p_9 + 84.3p_{10} + 84.2p_{11} + 84.9p_{12} - V >= 0\\
&95.7p_1 + 90.7p_2 + 90.3p_3 + 87.8p_4 + 83p_5 + 79.7p_6 + 78.9p_7 + 79.3p_8 + 79.5p_9 + 78.5p_{10} + 78.3p_{11} + 78.9p_{12} - V >= 0\\
&90.3p_1 + 90.1p_2 + 91.7p_3 + 92.5p_4 + 86.7p_5 + 82.1p_6 + 80.7p_7 + 81.5p_8 + 84p_9 + 85.1p_{10} + 86p_{11} + 89p_{12} - V >= 0\\
&110p_1 + 118.2p_2 + 125.5p_3 + 131.3p_4 + 127.3p_5 + 128.5p_6 + 128.2p_7 + 131.7p_8 + 131.2p_9 + 129p_{10} + 128.6p_{11} + 129.3p_{12} - V >= 0\\
&147.5p_1 + 144.8p_2 + 140.2p_3 + 134.3p_4 + 113p_5 + 97.5p_6 + 86.3p_7 + 82.3p_8 + 81.6p_9 + 82.1p_{10} + 84.1p_{11} + 85.8p_{12} - V >= 0
\end{aligned}
$$

$$p_1 + p_2 + p_3 + p_4 + p_5 + p_6 + p_7 + p_8 + p_9 + p_{10} + p_{11} + p_{12} = 1$$

Figure 8: Mathematical formulation for based on Benefit Criterion

Introduce an auxiliary variable Z,V:

$$\textbf{Max } \mathbf{Z = V}$$

s.t.

$$V \leqq \sum_{i=1,j=1}^{m,n} a_{ij}p_j$$

$$\sum_{j=1}^{n} p_j = 1$$

$$p_j \geqq 0$$

$$i = 1,2,\ldots,m$$

$$j = 1,2,\ldots,n$$

a_{ij}= price per lb matrix

where i = years, j = months

p_j =proportion of sales in each month

By including our data, it can be shown as:

$$
\begin{aligned}
&0p_1 + 3.5p_2 + 5.5p_3 + 6p_4 + 5.2p_5 + 2.1p_6 + 1.4p_7 + 1.5p_8 + 1.5p_9 + 2.7p_{10} + 2.1p_{11} + 2.2p_{12} - V >= 0\\
&0.4p_1 + 1p_2 + 1.2p_3 + 0.4p_4 + 0.1p_5 + 0p_6 + 0.2p_7 + 0.3p_8 + 0.3p_9 + 0.3p_{10} + 0.3p_{11} + 0.5p_{12} - V >= 0\\
&0p_1 + 0.2p_2 + 1.8p_3 + 5.8p_4 + 6.9p_5 + 8p_6 + 8.5p_7 + 8.4p_8 + 10.5p_9 + 10.9p_{10} + 11p_{11} + 11.1p_{12} - V >= 0\\
&18.7p_1 + 18.8p_2 + 18.8p_3 + 19.1p_4 + 19p_5 + 16.7p_6 + 11.2p_7 + 8.1p_8 + 4.7p_9 + 3.2p_{10} + 1.6p_{11} + 0p_{12} - V >= 0\\
&0p_1 + 3p_2 + 8.5p_3 + 9.9p_4 + 11.8p_5 + 13.6p_6 + 16p_7 + 16.9p_8 + 16.9p_9 + 16.7p_{10} + 16.5p_{11} + 16.3p_{12} - V >= 0\\
&4.8p_1 + 0.8p_2 + 1.1p_3 + 0.1p_4 + 0p_5 + 0.1p_6 + 0.9p_7 + 0.7p_8 + 1p_9 + 0.6p_{10} + 0.3p_{11} + 0.1p_{12} - V >= 0\\
&0p_1 + 0.1p_2 + 2.1p_3 + 4p_4 + 3.2p_5 + 2.2p_6 + 2.4p_7 + 2.6p_8 + 5.2p_9 + 6.8p_{10} + 7.7p_{11} + 9.7p_{12} - V >= 0\\
&0p_1 + 7.2p_2 + 14.4p_3 + 21p_4 + 23.8p_5 + 30p_6 + 32.1p_7 + 34.6p_8 + 34.4p_9 + 33.1p_{10} + 32.6p_{11} + 32p_{12} - V >= 0\\
&45.5p_1 + 43.6p_2 + 40.8p_3 + 37.4p_4 + 24.5p_5 + 14.1p_6 + 5p_7 + 0.5p_8 + 0p_9 + 1p_{10} + 2.9p_{11} + 3.7p_{12} - V >= 0
\end{aligned}
$$

$$p_1 + p_2 + p_3 + p_4 + p_5 + p_6 + p_7 + p_8 + p_9 + p_{10} + p_{11} + p_{12} = 1$$

Figure 9: Mathematical formulation for based on Wald's Maximax (Optimistic) Criterion

Introduce an auxiliary variable Z,V:

$$\textbf{Max Z = V}$$

s.t.

$$V \leqq \sum_{i=1,j=1}^{m,n} a_{ij}p_j$$

$$\sum_{j=1}^{n} p_j = 1$$

$$p_j \geqq 0$$

$$i = 1,2,\ldots,m$$

$$j = 1,2,\ldots,n$$

a_{ij}= price per lb matrix

where i = years, j = months

p_j =proportion of sales in each month

By including our data, it can be shown as:

$$85.5p_1 + 88.6p_2 + 89p_3 + 87.1p_4 + 80.3p_5 + 73.4p_6 + 72.5p_7 + 74.1p_8 + 74p_9 + 74.5p_{10} + 73.6p_{11} + 74.7p_{12} - V >= 0$$
$$92.8p_1 + 92.1p_2 + 90.2p_3 + 86.7p_4 + 80.6p_5 + 76.8p_6 + 76.9p_7 + 78.6p_8 + 78.4p_9 + 77.6p_{10} + 77.3p_{11} + 78.6p_{12} - V >= 0$$
$$93p_1 + 91.8p_2 + 91.7p_3 + 93.9p_4 + 88.8p_5 + 86p_6 + 86.4p_7 + 88p_8 + 90.1p_9 + 89.6p_{10} + 89.4p_{11} + 90.7p_{12} - V >= 0$$
$$107.1p_1 + 105.6p_2 + 103.2p_3 + 100.6p_4 + 93.7p_5 + 87.1p_6 + 81.1p_7 + 79.3p_8 + 75.5p_9 + 73.1p_{10} + 71.1p_{11} + 70.3p_{12} - V >= 0$$
$$84.2p_1 + 86.7p_2 + 91.5p_3 + 90.6p_4 + 86.6p_5 + 84.6p_6 + 87p_7 + 89.7p_8 + 89.6p_9 + 88.5p_{10} + 87.9p_{11} + 88.9p_{12} - V >= 0$$
$$103.6p_1 + 97p_2 + 95.2p_3 + 91.3p_4 + 85p_5 + 81.3p_6 + 82p_7 + 83.4p_8 + 83.6p_9 + 82.3p_{10} + 81.7p_{11} + 82.6p_{12} - V >= 0$$
$$97.7p_1 + 96.4p_2 + 96.6p_3 + 96.2p_4 + 88.8p_5 + 83.8p_6 + 83.8p_7 + 85.7p_8 + 88.4p_9 + 89.2p_{10} + 89.8p_{11} + 93.2p_{12} - V >= 0$$
$$119.1p_1 + 126.4p_2 + 132.4p_3 + 136.5p_4 + 130.4p_5 + 131.2p_6 + 133.2p_7 + 138.5p_8 + 138.1p_9 + 135.3p_{10} + 134.2p_{11} + 135.4p_{12} - V >= 0$$
$$159.7p_1 + 154.8p_2 + 147.9p_3 + 139.6p_4 + 115.8p_5 + 99.5p_6 + 89.7p_7 + 86.6p_8 + 85.9p_9 + 86.1p_{10} + 87.8p_{11} + 89.8p_{12} - V >= 0$$

$$p_1 + p_2 + p_3 + p_4 + p_5 + p_6 + p_7 + p_8 + p_9 + p_{10} + p_{11} + p_{12} = 1$$

Appendix III: Price Matrices for all Game Theory Criterion (Wald's Maximin (Pessimistic), Laplace, Hurwicz, Benefit and Wald's Maximax (Optimistic) Criterion

Table 10: Catfish prices for Wald (Optimistic) criterion (from Model 2)

	Optimistic													
Year	**January**	**February**	**March**	**April**	**May**	**June**	**July**	**August**	**September**	**October**	**November**	**December**	**Min**	**Max**
2004	6.00	2.50	0.50	0.00	0.80	3.90	4.60	4.50	4.50	3.30	3.90	3.80	0.00	6.00
2005	0.80	0.20	0.00	0.80	1.10	1.20	1.00	0.90	0.90	0.90	0.90	0.70	0.00	1.20
2006	11.10	10.90	9.30	5.30	4.20	3.10	2.60	2.70	0.60	0.20	0.10	0.00	0.00	11.10
2007	0.40	0.30	0.30	0.00	0.10	2.40	7.90	11.00	14.40	15.90	17.50	19.10	0.00	19.10
2008	16.90	13.90	8.40	7.00	5.10	3.30	0.90	0.00	0.00	0.20	0.40	0.60	0.00	16.90
2009	0.00	4.00	3.70	4.70	4.80	4.70	3.90	4.10	3.80	4.20	4.50	4.70	0.00	4.80
2010	9.70	9.60	7.60	5.70	6.50	7.50	7.30	7.10	4.50	2.90	2.00	0.00	0.00	9.70
2011	34.60	27.40	20.20	13.60	10.80	4.60	2.50	0.00	0.20	1.50	2.00	2.60	0.00	34.60
2012	0.00	1.90	4.70	8.10	21.00	31.40	40.50	45.00	45.50	44.50	42.60	41.80	0.00	45.50

Table 11: Catfish prices for Laplace criterion (from Model 2)

	Laplace													
Year	**January**	**February**	**March**	**April**	**May**	**June**	**July**	**August**	**September**	**October**	**November**	**December**	**Min**	**Max**
2004	69.1	74.2	77.5	78.8	75.5	69.7	65.7	65.0	64.8	65.9	65.8	66.3	64.8	78.8
2005	75.0	77.2	78.6	78.5	75.7	72.9	69.7	68.9	68.6	68.6	69.2	69.8	68.6	78.6
2006	75.3	77.0	79.9	85.0	83.5	81.6	78.3	77.2	78.9	79.2	80.0	80.6	75.3	85.0
2007	86.6	88.5	89.9	91.1	88.1	82.6	73.5	69.6	66.1	64.6	63.6	62.5	62.5	91.1
2008	68.1	72.7	79.7	82.0	81.4	80.3	78.9	78.7	78.4	78.2	78.6	78.9	68.1	82.0
2009	83.8	81.3	82.9	82.6	79.9	77.2	74.3	73.2	73.2	72.8	73.1	73.3	72.8	83.8
2010	79.1	80.8	84.2	87.1	83.5	79.5	76.0	75.2	77.4	78.8	80.3	82.8	75.2	87.1
2011	96.4	105.9	115.3	123.6	122.6	124.5	120.7	121.5	120.9	119.6	120.1	120.3	96.4	124.5
2012	129.2	129.8	128.8	126.4	108.9	94.5	81.3	75.9	75.2	76.1	78.5	79.8	75.2	129.8

Table 12: Catfish prices for Hurwicz criterion (from Model 2)

	Hurwicz													
Year	**January**	**February**	**March**	**April**	**May**	**June**	**July**	**August**	**September**	**October**	**November**	**December**	**Min**	**Max**
2004	78.9	82.8	84.4	83.8	78.4	71.9	69.8	70.5	70.3	71.1	70.5	71.3	69.8	84.4
2005	85.7	86.1	85.6	83.4	78.6	75.3	74.0	74.7	74.5	74.0	74.1	75.1	74.0	86.1
2006	85.9	85.9	87.0	90.3	86.7	84.3	83.1	83.7	85.6	85.5	85.6	86.6	83.1	90.3
2007	98.9	98.7	97.8	96.8	91.5	85.3	78.0	75.4	71.7	69.7	68.1	67.2	67.2	98.9
2008	77.8	81.1	86.8	87.1	84.5	82.9	83.7	85.3	85.1	84.3	84.2	84.9	77.8	87.1
2009	95.7	90.7	90.3	87.8	83.0	79.7	78.9	79.3	79.5	78.5	78.3	78.9	78.3	95.7
2010	90.3	90.1	91.7	92.5	86.7	82.1	80.7	81.5	84.0	85.1	86.0	89.0	80.7	92.5
2011	110.0	118.2	125.5	131.3	127.3	128.5	128.2	131.7	131.2	129.0	128.6	129.3	110.0	131.7
2012	147.5	144.8	140.2	134.3	113.0	97.5	86.3	82.3	81.6	82.1	84.1	85.8	81.6	147.5

Table 13: Catfish prices for Benefit criterion (from Model 2)

	Benefit													
Year	**January**	**February**	**March**	**April**	**May**	**June**	**July**	**August**	**September**	**October**	**November**	**December**	**Min**	**Max**
2004	0.00	3.50	5.50	6.00	5.20	2.10	1.40	1.50	1.50	2.70	2.10	2.20	0.00	6.00
2005	0.40	1.00	1.20	0.40	0.10	0.00	0.20	0.30	0.30	0.30	0.30	0.50	0.00	1.20
2006	0.00	0.20	1.80	5.80	6.90	8.00	8.50	8.40	10.50	10.90	11.00	11.10	0.00	11.10
2007	18.70	18.80	18.80	19.10	19.00	16.70	11.20	8.10	4.70	3.20	1.60	0.00	0.00	19.10
2008	0.00	3.00	8.50	9.90	11.80	13.60	16.00	16.90	16.90	16.70	16.50	16.30	0.00	16.90
2009	4.80	0.80	1.10	0.10	0.00	0.10	0.90	0.70	1.00	0.60	0.30	0.10	0.00	4.80
2010	0.00	0.10	2.10	4.00	3.20	2.20	2.40	2.60	5.20	6.80	7.70	9.70	0.00	9.70
2011	0.00	7.20	14.40	21.00	23.80	30.00	32.10	34.60	34.40	33.10	32.60	32.00	0.00	34.60
2012	45.50	43.60	40.80	37.40	24.50	14.10	5.00	0.50	0.00	1.00	2.90	3.70	0.00	45.50

Table 14: Catfish prices for pessimistic Wald's criterion (from Model 2)

							Pessimistic							
Year	**January**	**February**	**March**	**April**	**May**	**June**	**July**	**August**	**September**	**October**	**November**	**December**	**Min**	**Max**
2004	52.8	59.9	66.0	70.6	70.7	66.0	58.9	55.9	55.5	57.2	58.1	58.0	52.8	70.7
2005	57.3	62.3	66.9	70.3	70.9	69.0	62.5	59.2	58.9	59.6	61.0	61.0	57.3	70.9
2006	57.5	62.1	68.0	76.1	78.1	77.3	70.2	66.3	67.6	68.8	70.5	70.4	57.5	78.1
2007	66.2	71.4	76.5	81.5	82.4	78.2	65.8	59.8	56.7	56.1	56.1	54.6	54.6	82.4
2008	52.0	58.6	67.9	73.4	76.2	76.0	70.7	67.7	67.2	67.9	69.3	69.0	52.0	76.2
2009	64.1	65.6	70.6	74.0	74.8	73.1	66.6	62.9	62.8	63.2	64.5	64.1	62.8	74.8
2010	60.4	65.2	71.7	77.9	78.1	75.3	68.1	64.6	66.3	68.5	70.9	72.3	60.4	78.1
2011	73.6	85.5	98.2	110.6	114.7	117.9	108.2	104.5	103.6	103.9	105.9	105.1	73.6	117.9
2012	98.7	104.7	109.7	113.1	101.9	89.4	72.8	65.3	64.5	66.1	69.3	69.7	64.5	113.1

Table 15: Catfish prices for Lower Bound (from Model 2)

							Lower Bound					
Year	**January**	**February**	**March**	**April**	**May**	**June**	**July**	**August**	**September**	**October**	**November**	**December**
2004	52.8	59.9	66.0	70.6	70.7	66.0	58.9	55.9	55.5	57.2	58.1	58.0
2005	57.3	62.3	66.9	70.3	70.9	69.0	62.5	59.2	58.9	59.6	61.0	61.0
2006	57.5	62.1	68.0	76.1	78.1	77.3	70.2	66.3	67.6	68.8	70.5	70.4
2007	66.2	71.4	76.5	81.5	82.4	78.2	65.8	59.8	56.7	56.1	56.1	54.6
2008	52.0	58.6	67.9	73.4	76.2	76.0	70.7	67.7	67.2	67.9	69.3	69.0
2009	64.1	65.6	70.6	74.0	74.8	73.1	66.6	62.9	62.8	63.2	64.5	64.1
2010	60.4	65.2	71.7	77.9	78.1	75.3	68.1	64.6	66.3	68.5	70.9	72.3
2011	73.6	85.5	98.2	110.6	114.7	117.9	108.2	104.5	103.6	103.9	105.9	105.1
2012	98.7	104.7	109.7	113.1	101.9	89.4	72.8	65.3	64.5	66.1	69.3	69.7

Table 16: Catfish prices for Upper Bound (from Model 2)

							Upper Bound					
Year	**January**	**February**	**March**	**April**	**May**	**June**	**July**	**August**	**September**	**October**	**November**	**December**
2004	85.5	88.6	89.0	87.1	80.3	73.4	72.5	74.1	74.0	74.5	73.6	74.7
2005	92.8	92.1	90.2	86.7	80.6	76.8	76.9	78.6	78.4	77.6	77.3	78.6
2006	93.0	91.8	91.7	93.9	88.8	86.0	86.4	88.0	90.1	89.6	89.4	90.7
2007	107.1	105.6	103.2	100.6	93.7	87.1	81.1	79.3	75.5	73.1	71.1	70.3
2008	84.2	86.7	91.5	90.6	86.6	84.6	87.0	89.7	89.6	88.5	87.9	88.9
2009	103.6	97.0	95.2	91.3	85.0	81.3	82.0	83.4	83.6	82.3	81.7	82.6
2010	97.7	96.4	96.6	96.2	88.8	83.8	83.8	85.7	88.4	89.2	89.8	93.2
2011	119.1	126.4	132.4	136.5	130.4	131.2	133.2	138.5	138.1	135.3	134.2	135.4
2012	159.7	154.8	147.9	139.6	115.8	99.5	89.7	86.6	85.9	86.1	87.8	89.8

Appendix IV: Complete Decision Variable Results for Model 2

Table 4: Summary of decision variables for Wald (Pessimistic) criterion (from Model 2)

Profit = $23,253.68

Month	q_one	q_two	x	d	e_one	e_two
0	-	-	60,000	-	-	-
1	96,000	-	-	141,504	211,200	-
2	96,000	-	-	81,504	211,200	-
3	96,000	-	-	81,504	211,200	-
4	96,000	-	-	81,504	211,200	-
5	96,000	-	-	81,504	211,200	-
6	96,000	-	-	81,504	211,200	-
7	96,000	-	-	81,504	211,200	-
8	96,000	-	-	81,504	211,200	-
9	96,000	-	-	81,504	211,200	-
10	96,000	-	16,992	64,512	211,200	-
11	96,000	-	38,496	60,000	211,200	-
12	96,000	-	60,000	60,000	211,200	-

Table 5: Summary of decision variables for Laplace criterion (from Model 2)

Profit = $30,099.95

Month	q_one	q_two	x	d	e_one	e_two
0	-	-	60,000	-	-	-
1	96,000	-	81,504	60,000	211,200	-
2	96,000	-	-	163,008	211,200	-
3	96,000	-	-	81,504	211,200	-
4	96,000	-	-	81,504	211,200	-
5	96,000	-	-	81,504	211,200	-
6	96,000	-	-	81,504	211,200	-
7	96,000	-	-	81,504	211,200	-
8	96,000	-	-	81,504	211,200	-
9	96,000	-	-	81,504	211,200	-
10	96,000	-	16,992	64,512	211,200	-
11	96,000	-	38,496	60,000	211,200	-
12	96,000	-	60,000	60,000	211,200	-

Table 6: Summary of decision variables for Max regret criterion (from Model 2)

Profit = $29,907.01

Month	q_one	q_two	x	d	e_one	e_two
0	-	-	60,000	-	-	-
1	96,000	-	-	141,504	211,200	-
2	96,000	-	-	81,504	211,200	-
3	96,000	-	21,504	60,000	211,200	-
4	96,000	-	43,008	60,000	211,200	-
5	96,000	-	64,512	60,000	211,200	-
6	96,000	-	-	146,016	211,200	-
7	96,000	-	-	81,504	211,200	-
8	96,000	-	-	81,504	211,200	-
9	96,000	-	-	81,504	211,200	-
10	96,000	-	16,992	64,512	211,200	-
11	96,000	-	38,496	60,000	211,200	-
12	96,000	-	60,000	60,000	211,200	-

Table 7: Summary of decision variables for Hurwicz criterion (from Model 2)

Profit = $33,422.20

Month	q_one	q_two	x	d	e_one	e_two
0	-	-	60,000	-	-	-
1	96,000	-	81,504	60,000	211,200	-
2	96,000	-	103,008	60,000	211,200	-
3	96,000	-	-	184,512	211,200	-
4	96,000	-	-	81,504	211,200	-
5	96,000	-	-	81,504	211,200	-
6	96,000	-	-	81,504	211,200	-
7	96,000	-	-	81,504	211,200	-
8	96,000	-	-	81,504	211,200	-
9	94,149	-	-	79,932	207,127	-
10	96,000	-	16,992	64,512	211,200	-
11	96,000	-	38,496	60,000	211,200	-
12	96,000	-	60,000	60,000	211,200	-

Table 8: Summary of decision variables for Benefit criterion (from Model 2)

Profit = $39,780.48

Month	q_one	q_two	x	d	e_one	e_two
0	-	-	60,000	-	-	-
1	96,000	-	-	141,504	211,200	-
2	96,000	-	21,504	60,000	211,200	-
3	96,000	-	-	103,008	211,200	-
4	96,000	-	-	81,504	211,200	-
5	96,000	-	-	81,504	211,200	-
6	96,000	-	-	81,504	211,200	-
7	96,000	-	-	81,504	211,200	-
8	96,000	-	-	81,504	211,200	-
9	96,000	-	-	81,504	211,200	-
10	96,000	-	16,992	64,512	211,200	-
11	96,000	-	38,496	60,000	211,200	-
12	96,000	-	60,000	60,000	211,200	-

Table 9: Summary of decision variables for Wald (Optimistic) criterion (from Model 2)

Profit = $39, 149.74

Month	q_one	q_two	x	d	e_one	e_two
0	-	-	60,000	-	-	-
1	96,000	-	81,504	-	-	-
2	96,000	-	103,008	-	-	-
3	96,000	-	-	-	-	-
4	96,000	-	-	-	-	-
5	96,000	-	-	-	-	-
6	96,000	-	-	-	-	-
7	96,000	-	-	81,504	211,200	-
8	96,000	-	-	81,504	211,200	-
9	94,149	-	-	79,932	207,127	-
10	96,000	-	16,992	64,512	211,200	-
11	96,000	-	38,496	60,000	211,200	-
12	96,000	-	60,000	60,000	211,200	-

Appendix V: Python code

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']]
```

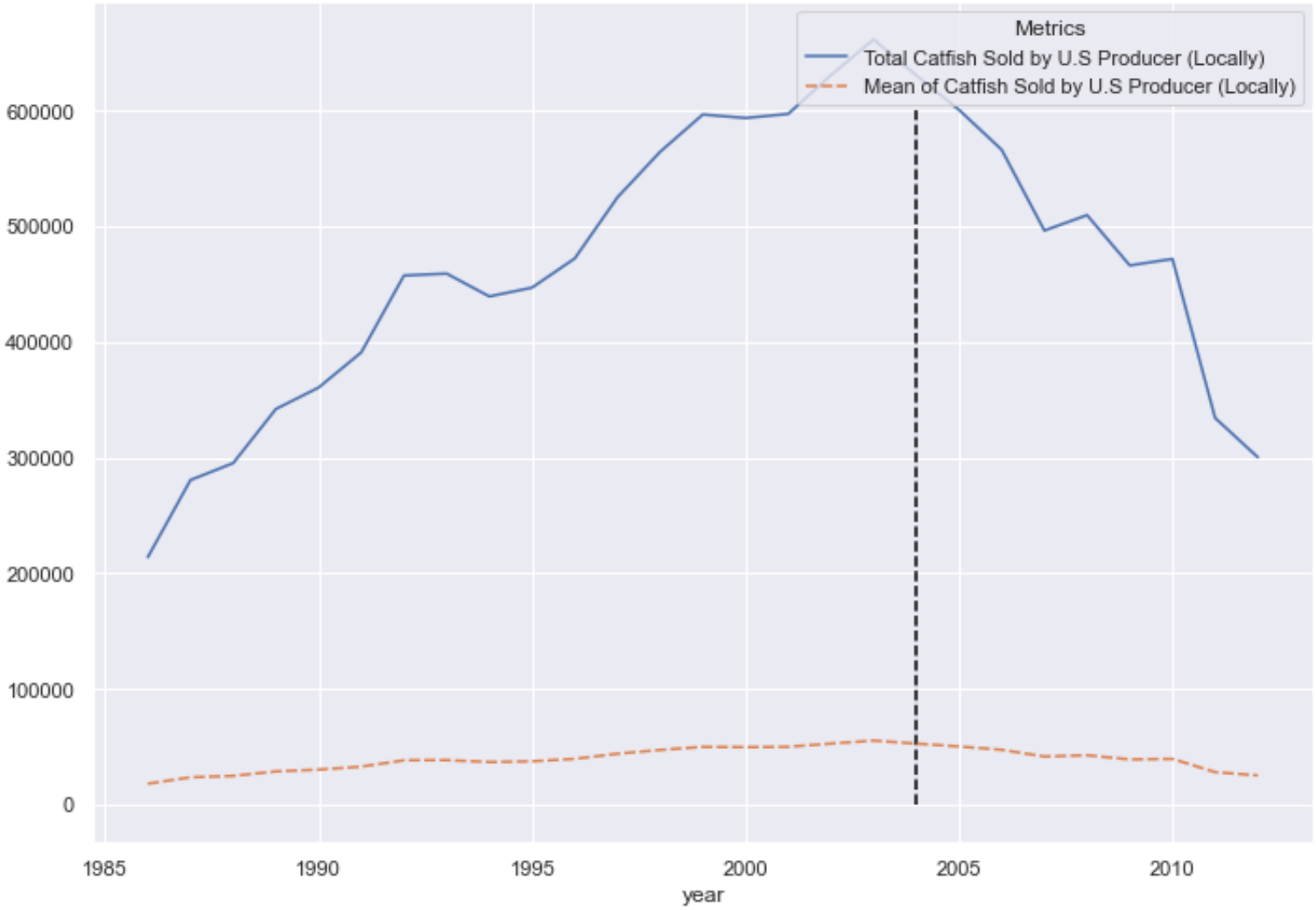
In [18]:
```
sns.set(rc={'figure.figsize':(11.7,8.27)})
sns.lineplot(data = df_plot1, legend=False)
plt.vlines(x=2004, color='black', linestyle='--', ymin = 0, ymax = 125)
plt.legend(title='Metrics', loc='upper right', labels=['Catfish Price Mean', 'Catfish Price Standard Deviation'])
```

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)
```

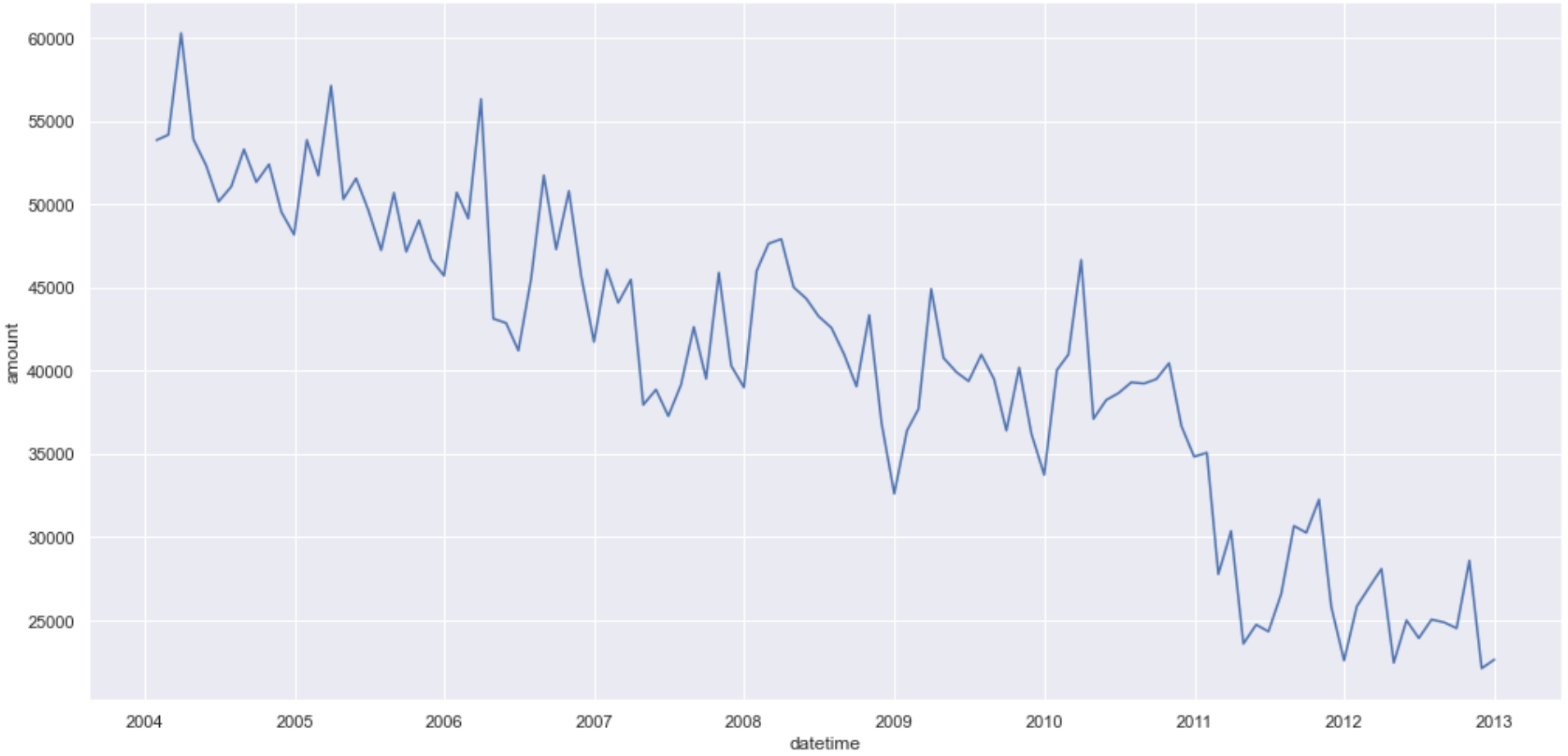
In [22]:
```
df_sales = df_sales[df_sales['year_id'] >= 2004]
```

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)
```

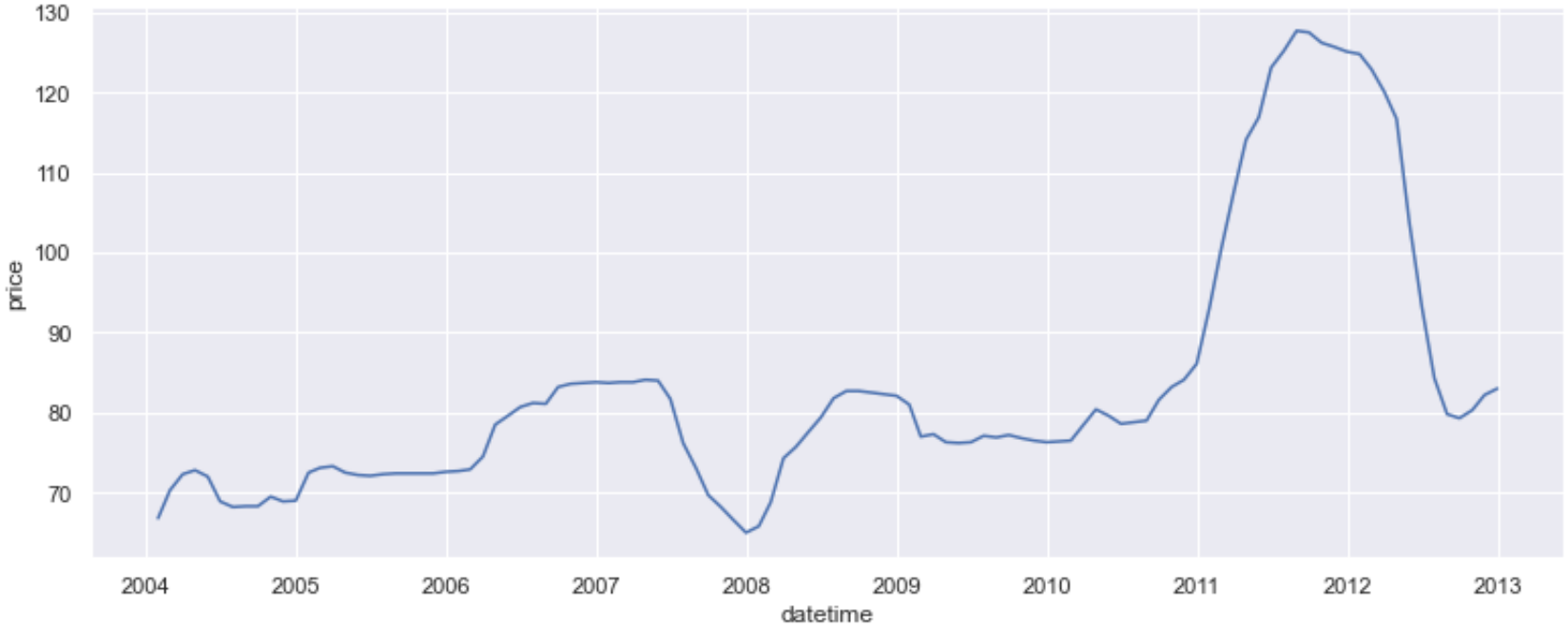
In [25]:
```
sns.set(rc={'figure.figsize':(17,8.27)})
sns.lineplot(data=df_sales, x='datetime', y='amount')
```

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]:

datetime	index	year_id	timeperiod_id	commodity_desc	amount	price	benefit_criterion	regret_criterion	lower_bound	upper_bound	b
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")
```

```
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: 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],
```

```
                    '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")
```

```
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: 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]:

```
###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")
```

```
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],
```

```
                    '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]:

```
###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")
```

```
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: 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")
```

```
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: 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 [ ]: