Presentation 1
OLS + Capacity

Group: WiseGoose-IntrepidBeluga

Members: Nicholas Chandler, Raphael Bergner

Agenda

01 Nick’s Approaches

1. Baseline
2. Capacity Curves
3. LS

02 Raphael’s Approaches

1. Baseline
2. Implementation Details
3. Competition Deep Dive

Nick

Baseline (Nick)

Baseline:

- Assume demand is approximately equally distributed over time
- Adjust price based on remaining inventory vs. expected, given assumptions
- Sell more if competitor is out and consider the time in the season.
- Repeated this from 10-8-25 to 10-20-25 (and 10-22-25 to 10-29-25)

$$P(t) = \text{clip}_{[P_{\min}, P_{\max}]}\left[P_0\left(1 - \alpha\left(\frac{\text{remaining}}{\text{expected\_remaining}} - 1\right)\right) \cdot \left(\mathbf{1}_{\text{competitor in stock}} + \beta \cdot \mathbf{1}_{\text{competitor out of stock}}\right) \cdot \left(1 + \gamma \frac{t}{T_{\text{season}}}\right)\right]$$

Accounting for demand

Accounting for monopoly

Accounting for the time left

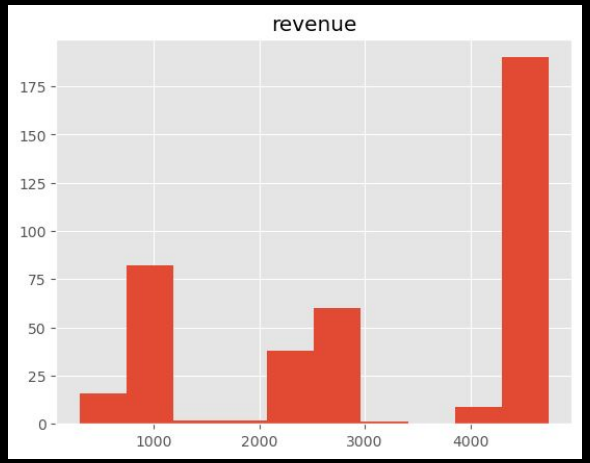
```
# expected inventory remaining if evenly sold
periods_left = max(1, T_SEASON - (selling_period_in_current_season - 1))
expected_remaining = C_INIT * (periods_left / T_SEASON)

ratio = remaining / expected_remaining if expected_remaining > 0 else 1.0

# price adjustment based on inventory ratio
# if ratio > 1: overstock -> reduce price
# if ratio < 1: understock -> raise price
price = BASE_PRICE * (1 - AGGRESSIVENESS * (ratio - 1))
```

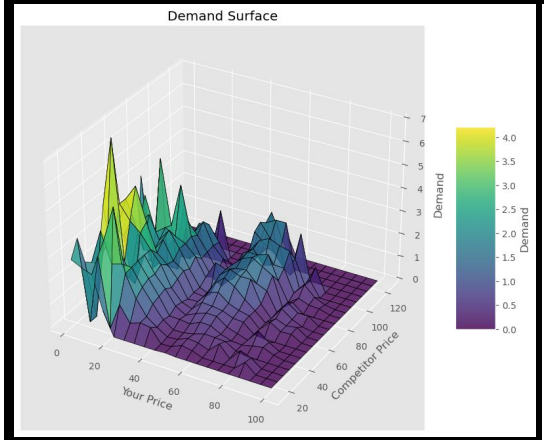
Core of Algo.

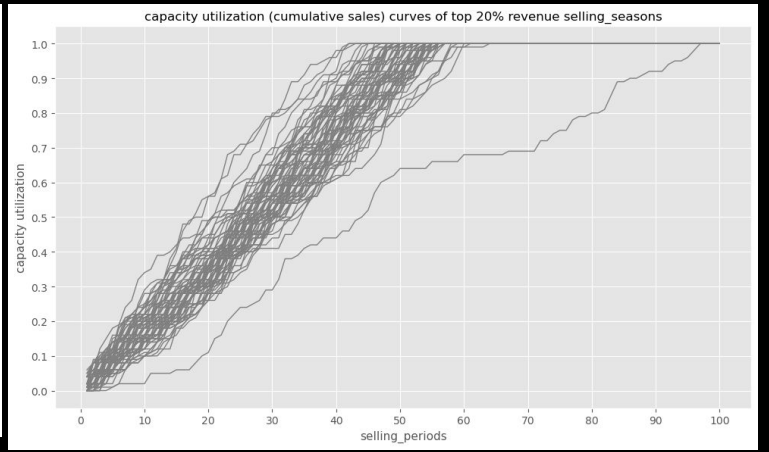
Baseline (Nick)

Results from best baseline competition day:

- Run on 10-20-25
- Investigation of the trimodal revenue distribution yielded few patterns...







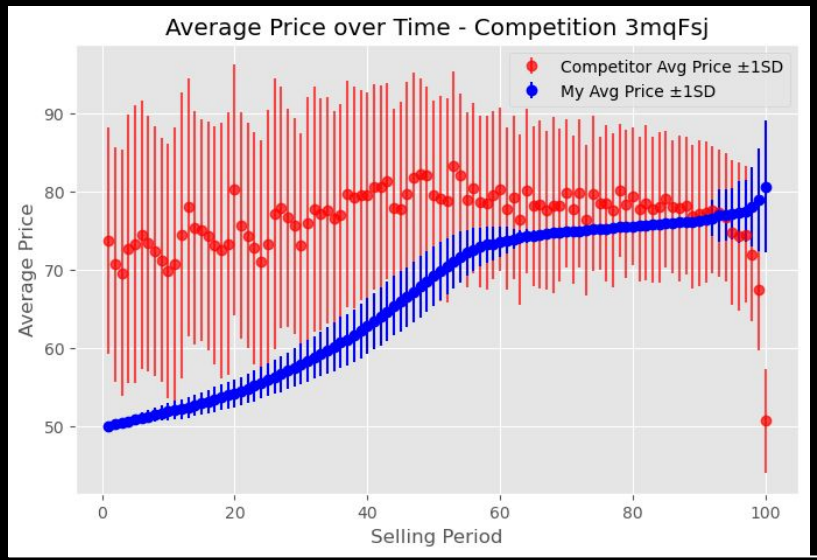
7	IntrepidBeluga	$1,259,441	$314,860	1.25	$1.00	$100.00	$44.04	0.962	0.482

Baseline (Nick)

Results from best baseline competition day:

- Run on 10-20-25
- Maybe the competitor just priced poorly...

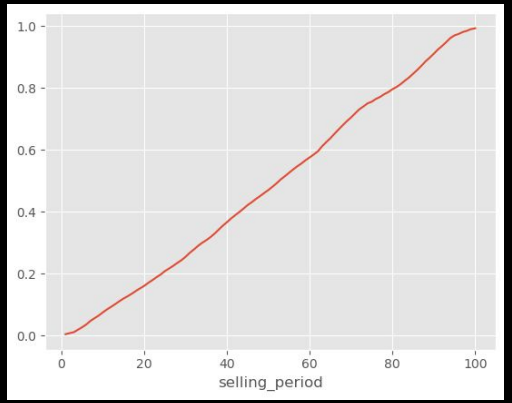
```
Competition with highest revenue: 3mqFsj (Total Revenue: 468015.8)
```



Capacity Curves I (Nick)

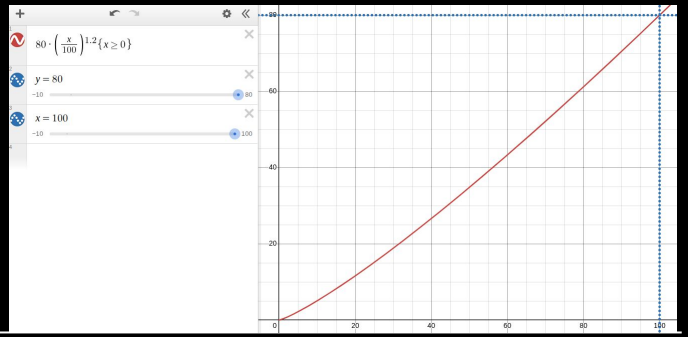
- Similar oversold/undersold based on expected selling amount
  - Season level
- Now we account for the change in sales based on our target
  - Time-step level
- Adds an additional constraint to ensure we don't sell out early
- Tested on 10-22-25

Mean capacity curve of best seasons




```python
# === PRICE DECISION ===
price = last_price  # default: keep same
if selling_period_in_current_season in target_curve:
    target_sales = target_curve[selling_period_in_current_season]["sold_seats"]
    if target_sales > 0:
        delta_sales = (current_sold_seats - target_sales) / target_sales
        # Inventory correction:
        # if we have more remaining than expected, we're lagging -> cheaper
        # if we have less remaining than expected, we're ahead -> pricier
        inv_adjustment = -INVENTORY_SENSITIVITY * (remaining_ratio - 1)
        # Combine sales and inventory deltas
        combined_delta = delta_sales + inv_adjustment
        adj_factor = np.clip(1 + combined_delta, 1 - ADJUSTMENT_CAP, 1 + ADJUSTMENT_CAP)
        price = last_price * adj_factor
```

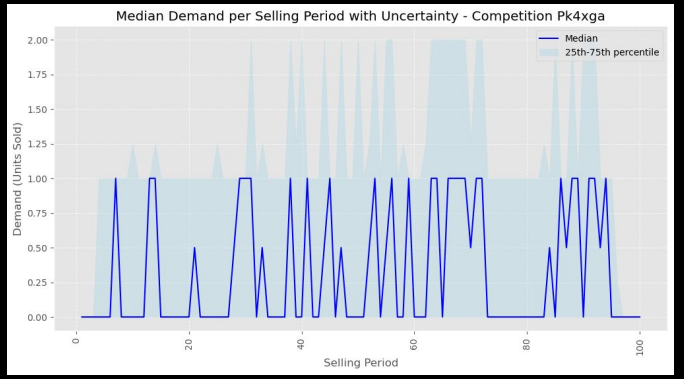
Target capacity curve over time

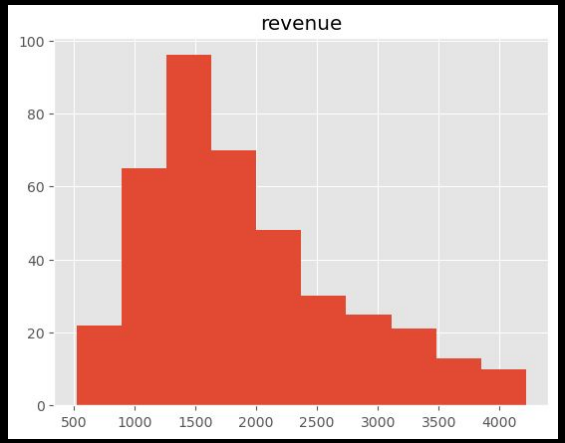


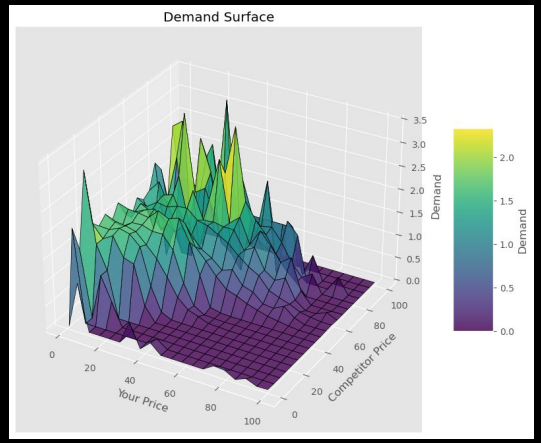
Capacity Curves I (Nick)

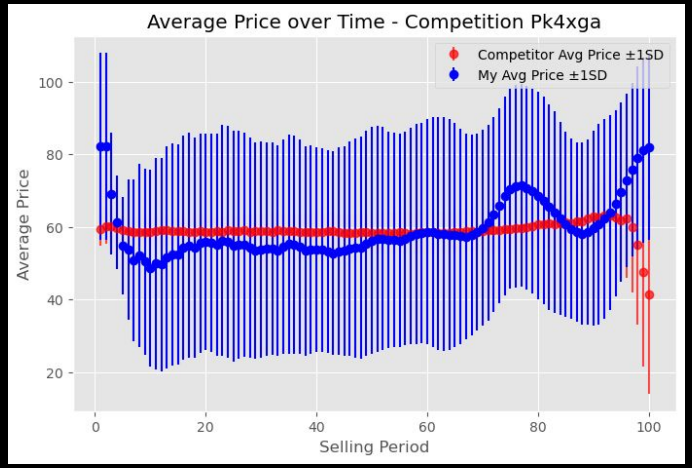
Result:

- Lower placement but better following of the target curve
- More dynamic pricing
- Lower revenue overall (but nicer distribution)
- 10-22-25









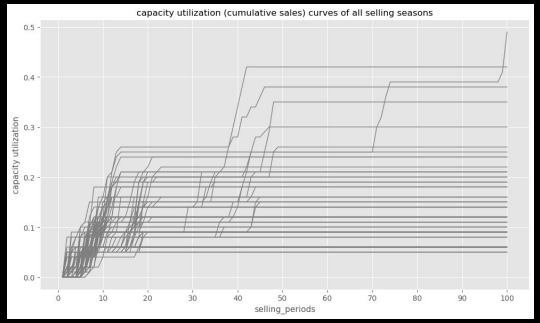

```
Competition with highest revenue: Pk4xga (Total Revenue: 296860.3)
```

18	IntrepidBeluga	$763,955	$190,989	2	$1.00	$100.00	$45.98	0.971	0.608

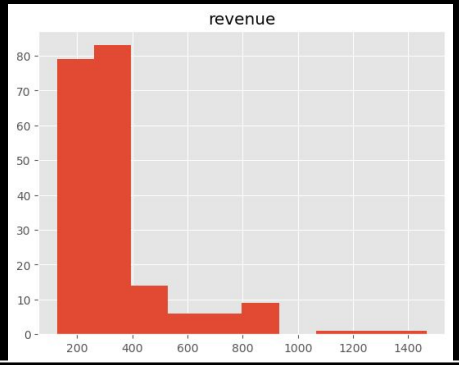
Capacity Curves II (Nick)

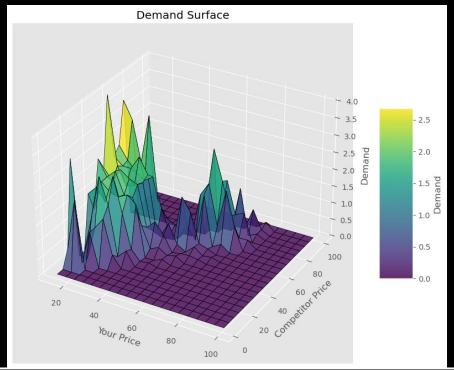
- This time we incorporated **randomized capacity curves** as in the lecture
- In theory we select a new target curve at at various checkpoints and do the same as the other capacity curve approach
- My approach was buggy (often too expensive) however so the results were subpar
- Done on 10-29-25

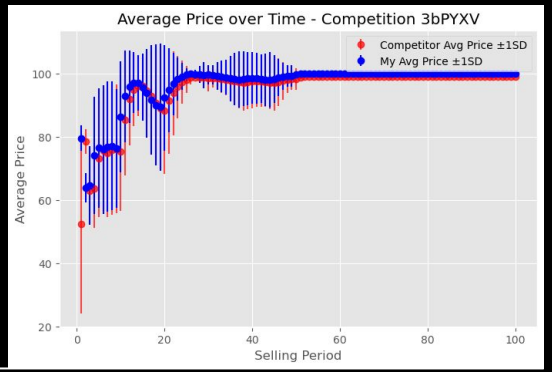
- Lesson: Pricing too high will destroy returns




```
Competition with highest revenue: 3bPYXV (Total Revenue: 40452.6)
```







65	IntrepidBeluga	$68,307	$34,154	2	$13.40	$100.00	$93.95	0.934	0

10 rows 61-65 of 65

LS (Nick)

Tried RLS demand model (Recursive Least Squares):

- Prediction Error: $e_t = y_t - \boldsymbol{x}_t^\top \boldsymbol{\theta}_{t-1}$
- Gain Vector: $\boldsymbol{K}_t = \frac{\boldsymbol{P}_{t-1}\boldsymbol{x}_t}{\lambda + \boldsymbol{x}_t^\top \boldsymbol{P}_{t-1}\boldsymbol{x}_t}$
- Parameter Update: $\boldsymbol{\theta}_t = \boldsymbol{\theta}_{t-1} + \boldsymbol{K}_t e_t$
- Covariance Update: $\boldsymbol{P}_t = \frac{1}{\lambda}\left(\boldsymbol{P}_{t-1} - \boldsymbol{K}_t \boldsymbol{x}_t^\top \boldsymbol{P}_{t-1}\right)$

Used estimated demand to price:

- Demand Model: $D(p, p_c) = \theta_0 + \theta_1 p + \theta_2 p_c$
- Revenue Function: $R(p, p_c) = p \cdot D(p, p_c)$
- Optimization of R: $\frac{\partial R}{\partial p} = \theta_0 + 2\theta_1 p + \theta_2 p_c$

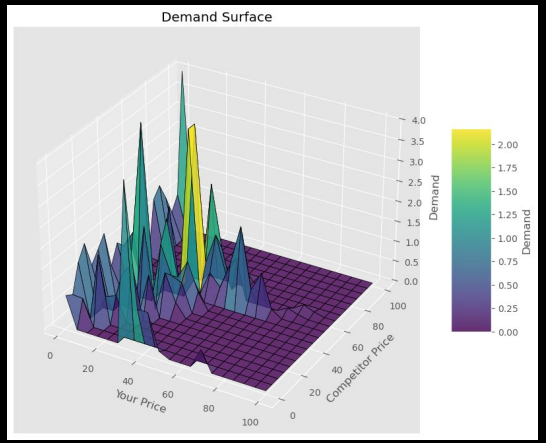
$$p^* = -\frac{\theta_0 + \theta_2 p_c}{2\theta_1} \qquad \frac{\partial^2 R}{\partial p^2} = 2\theta_1 < 0$$

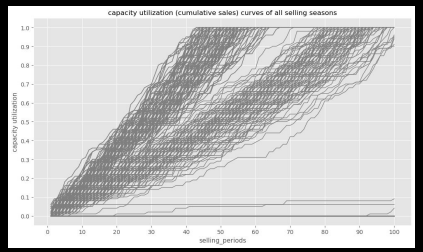
Idea: Update the demand model as data flows in, use analytically optimal price given demand model.

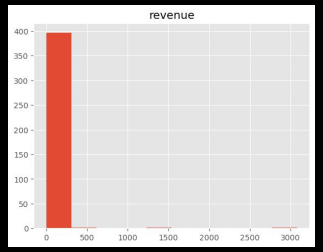
LS (Nick)

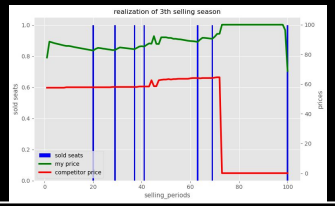
Way messier in practice. (11-1-25)

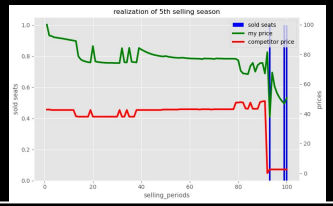
- Needed to clip/clamp for numerical under/overflow
- Estimates tended to be pretty awful a lot of the time
  ⇒ Model doesn't fit the environment very well
- First run encountered adversarial competitors
- Had a bug in the code with my floor after competitor runs out function...

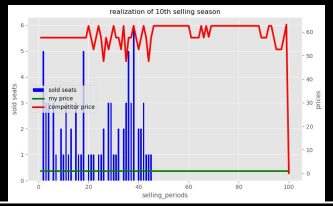


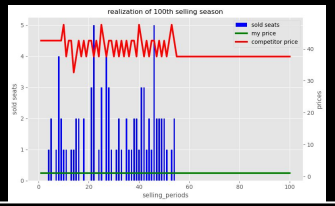












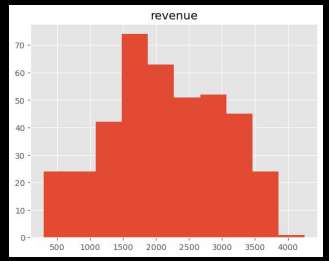
61	IntrepidBeluga	$31,620	$7,905	2	$1.00	$100.00	$18.98	0.837	0.725

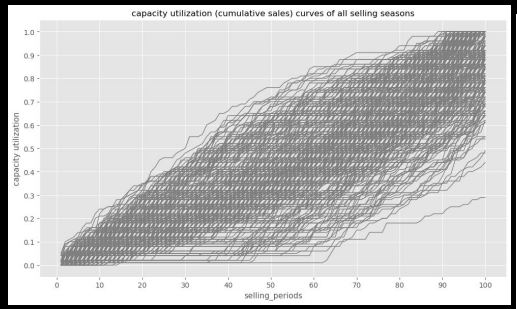
LS (Nick)

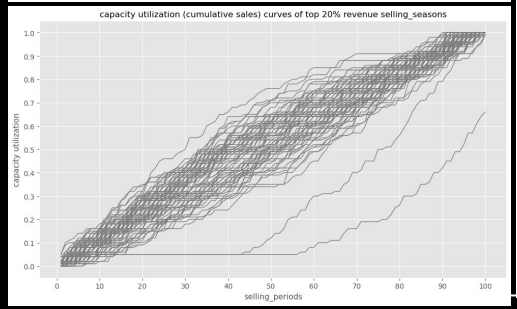
Try again without the bugs. (11-3-25)

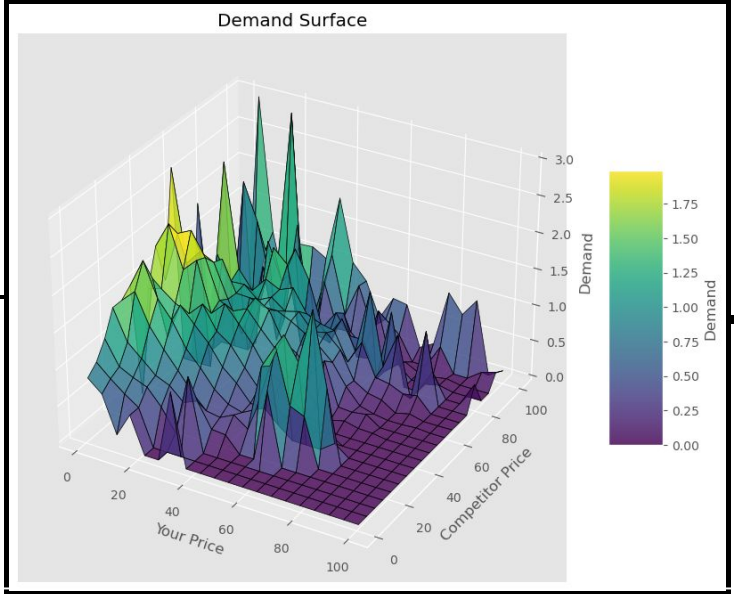
- Same as before but without the end-undercutting bug (removed undercutting)
- Had 5% random chance of pricing $5 below the competitor's last price
- Maybe lower initial price would be better to nix the late-selling seasons

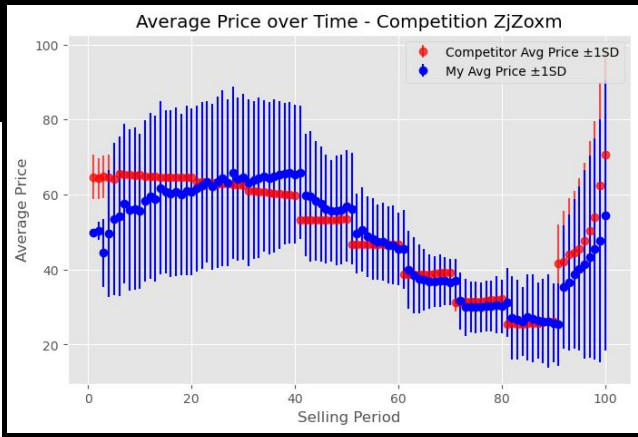
```
Competition with highest revenue: ZjZoxm (Total Revenue: 294460.1)
```

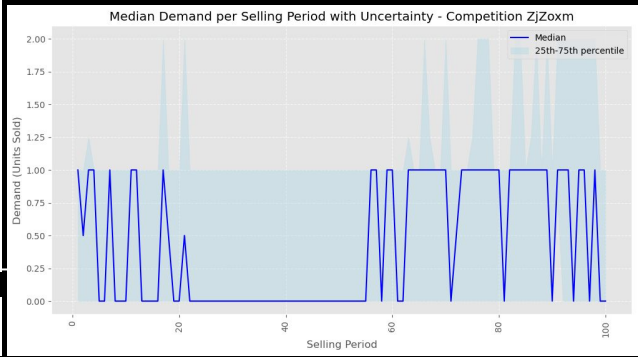












15	IntrepidBeluga	$852,919	$213,230	1	$1.00	$100.00	$36.98	0.816	0.34

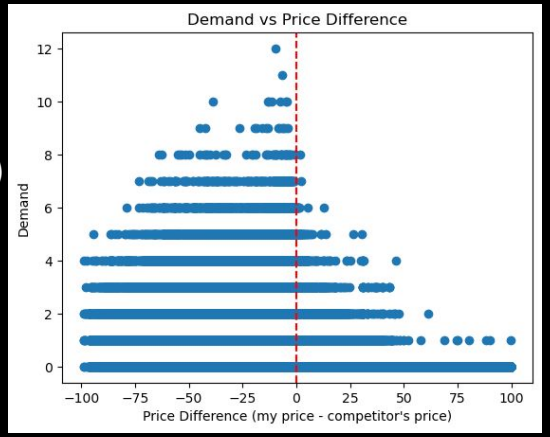
More Analysis (Nick)

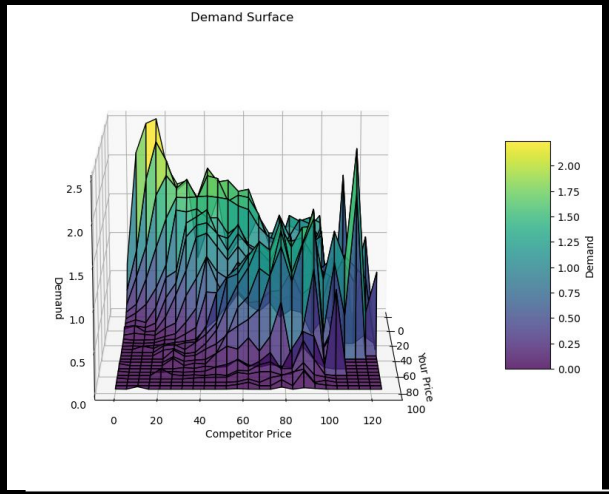
What insights can we glean from lots of price + demand data? (12 competition days)

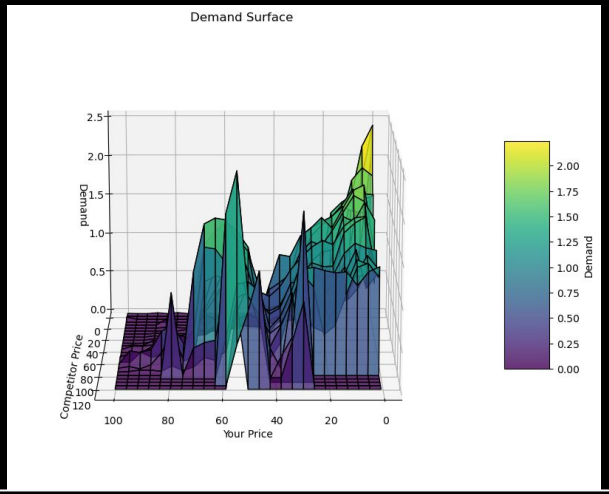
- We ignore the non-stationarity of the demand with these plots
- High demand (>6) rarely occurs when my price is above my competitor's

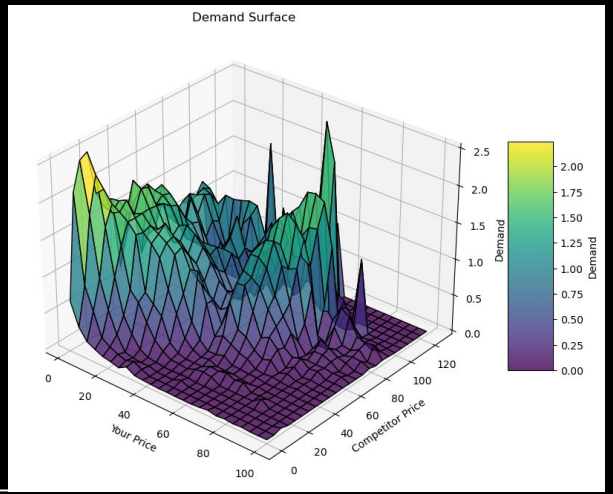
Future Work:

- Look at more complex modeling of demand?
- Implement smarter adversarial behavior









Raphael

Baseline (Raph)

1. Demand is a step function dependent on my price
2. Estimate competitors price using historic data, and move closer towards his likely set price
3. Keep track of your stock and set price to match a linear utilization curve

Baseline (Raph)

1. Demand is a step function dependent on my price
2. Estimate competitors price using historical data, and move closer towards his likely set price
3. Keep track of your stock and set price to match a linear utilization curve

Ranking

- Results from 10-20-25

Search

Overall Ranking	Name	Total Revenue	Avg Revenue per Simulation	Avg Ranking per Simulation	Min Set Price	Max Set Price	Mean Price	Autocorrelation Price	Stockout Frequency
1	WiseGoose	$1,781,451	$445,363	1.75	$27.90	$86.20	$68.50	0.593	0.318
2	WhimsicalMarten	$1,684,124	$421,031	1	$5.10	$97.60	$53.89	0.743	0.425

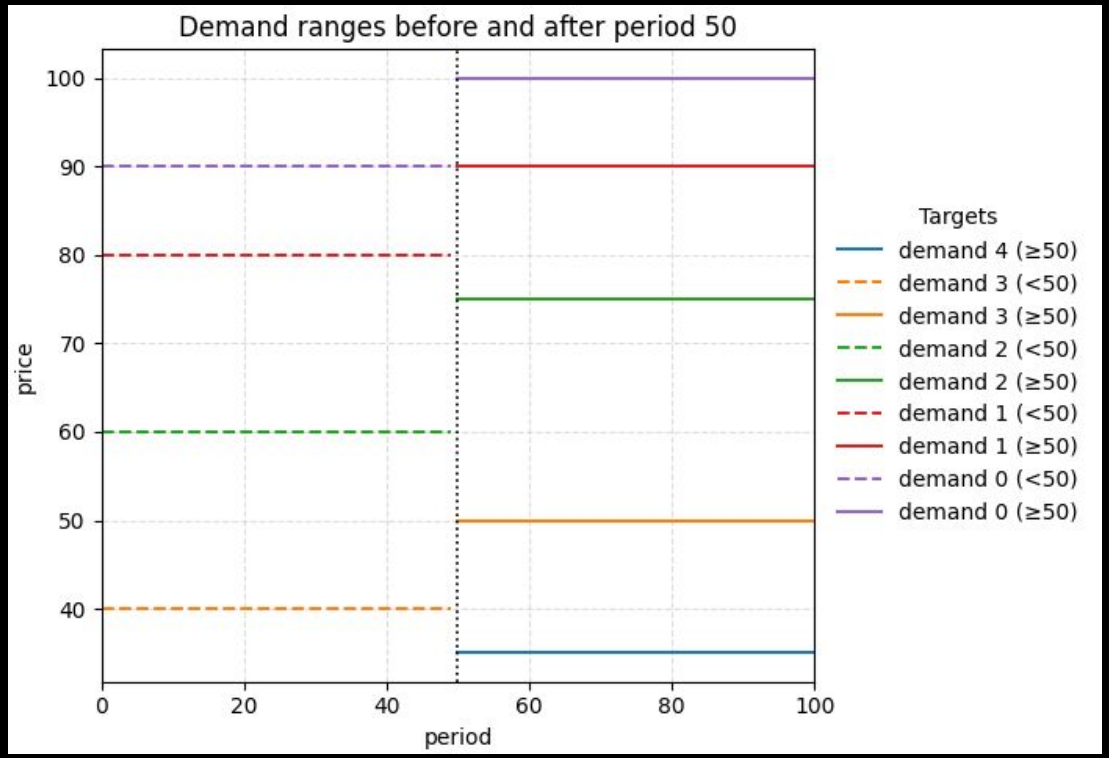
Demand Estimation (Raph)

Demand as a Step-Function

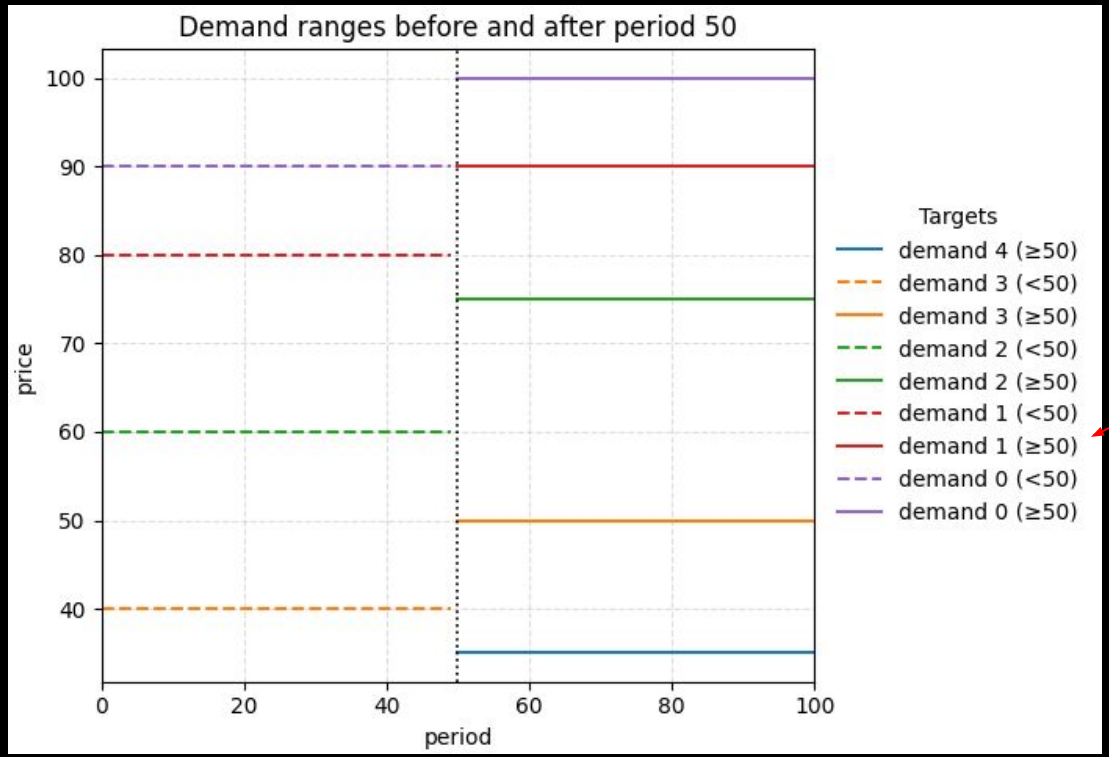
- Good for testing
- Decent base-line

```
if selling_period_in_current_season < 50:
    my_demand_ranges = {4: np.nan, 3: 40, 2: 60, 1: 80, 0: 90}
else:
    my_demand_ranges = {4: 35, 3: 50, 2: 75, 1: 90, 0: 100}
```

Demand Estimation (Raph)



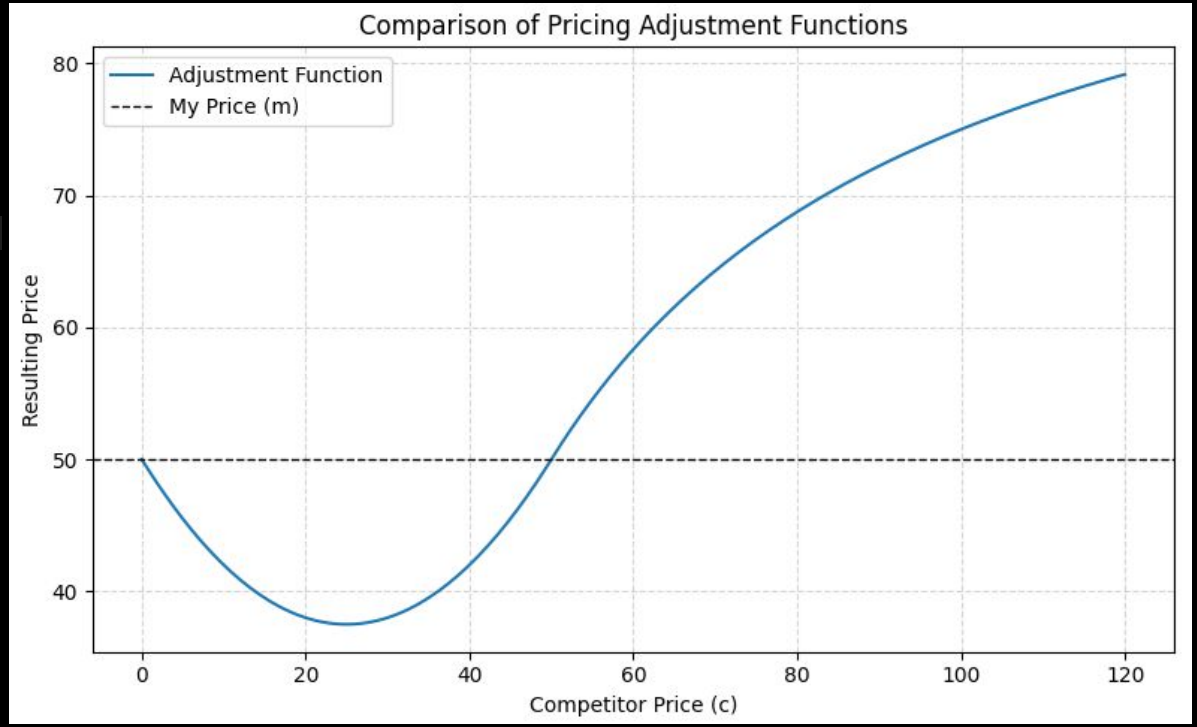
Demand Estimation (Raph)



Example:
Period = 40
Stock left = 20
⇒ 60 periods left to sell 60 seats, meaning 1 seat per period
⇒ set price to 80

Competitor Adjustments (Raph)

```
def comp_linear(c: float, m: float)
-> float:
  if np.isnan(m):
      return np.nan

  diff = abs(m - c)
  if diff < 0.001:
      return m
  perc = min(m, c)/max(m, c)
  if m > c:
      return m - diff * perc
  return m + diff * perc
```



Linear Interpolation (Raph)

```
def next_price_estimator(demand_ranges, period, stock_left, days_left, lower_demand, upper_demand):
  expected_demand_to_sell_for = estimate_next_need_demand(period, stock_left)
  lower_demand = np.floor(expected_demand_to_sell_for)
  upper_demand = np.ceil(expected_demand_to_sell_for)
  lower = demand_ranges[lower_demand]
  upper = demand_ranges[upper_demand]
  stock_per_day = stock_left/days_left
  percentage = stock_per_day - np.floor(stock_per_day)
  return (upper - lower) * percentage + lower
```

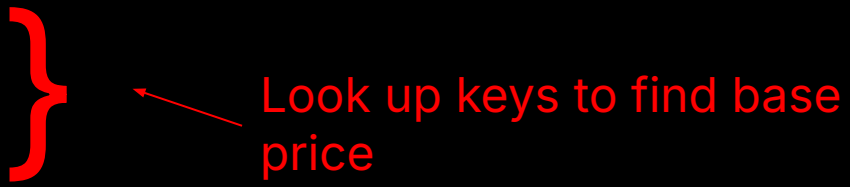
Linear Interpolation (Raph)

```
def next_price_estimator(demand_ranges, period, stock_left, days_left, lower_demand, upper_demand):
  expected_demand_to_sell_for = estimate_next_need_demand(period, stock_left)
  lower_demand = np.floor(expected_demand_to_sell_for)
  upper_demand = np.ceil(expected_demand_to_sell_for)
  lower = demand_ranges[lower_demand]
  upper = demand_ranges[upper_demand]
  stock_per_day = stock_left/days_left
  percentage = stock_per_day - np.floor(stock_per_day)
  return (upper - lower) * percentage + lower
```

How many seats to sell now
As float, example: 1.2

Linear Interpolation (Raph)

```
def next_price_estimator(demand_ranges, period, stock_left, days_left, lower_demand, upper_demand):
  expected_demand_to_sell_for = estimate_next_need_demand(period, stock_left)
  lower_demand = np.floor(expected_demand_to_sell_for)
  upper_demand = np.ceil(expected_demand_to_sell_for)
  lower = demand_ranges[lower_demand]
  upper = demand_ranges[upper_demand]
  stock_per_day = stock_left/days_left
  percentage = stock_per_day - np.floor(stock_per_day)
  return (upper - lower) * percentage + lower
```



Linear Interpolation (Raph)

```
def next_price_estimator(demand_ranges, period, stock_left, days_left, lower_demand, upper_demand):
  expected_demand_to_sell_for = estimate_next_need_demand(period, stock_left)
  lower_demand = np.floor(expected_demand_to_sell_for)
  upper_demand = np.ceil(expected_demand_to_sell_for)
  upper = demand_ranges[lower_demand]
  lower = demand_ranges[upper_demand]
  stock_per_day = stock_left/days_left
  percentage = stock_per_day - np.floor(stock_per_day)
  return (upper - lower) * percentage + lower
```

80

60

For 1.2

```
  if selling_period_in_current_season < 50:
      my_demand_ranges = {4: np.nan, 3: 40, 2: 60, 1: 80, 0: 90}
  else:
      my_demand_ranges = {4: 35, 3: 50, 2: 75, 1: 90, 0: 100}
```

Linear Interpolation (Raph)

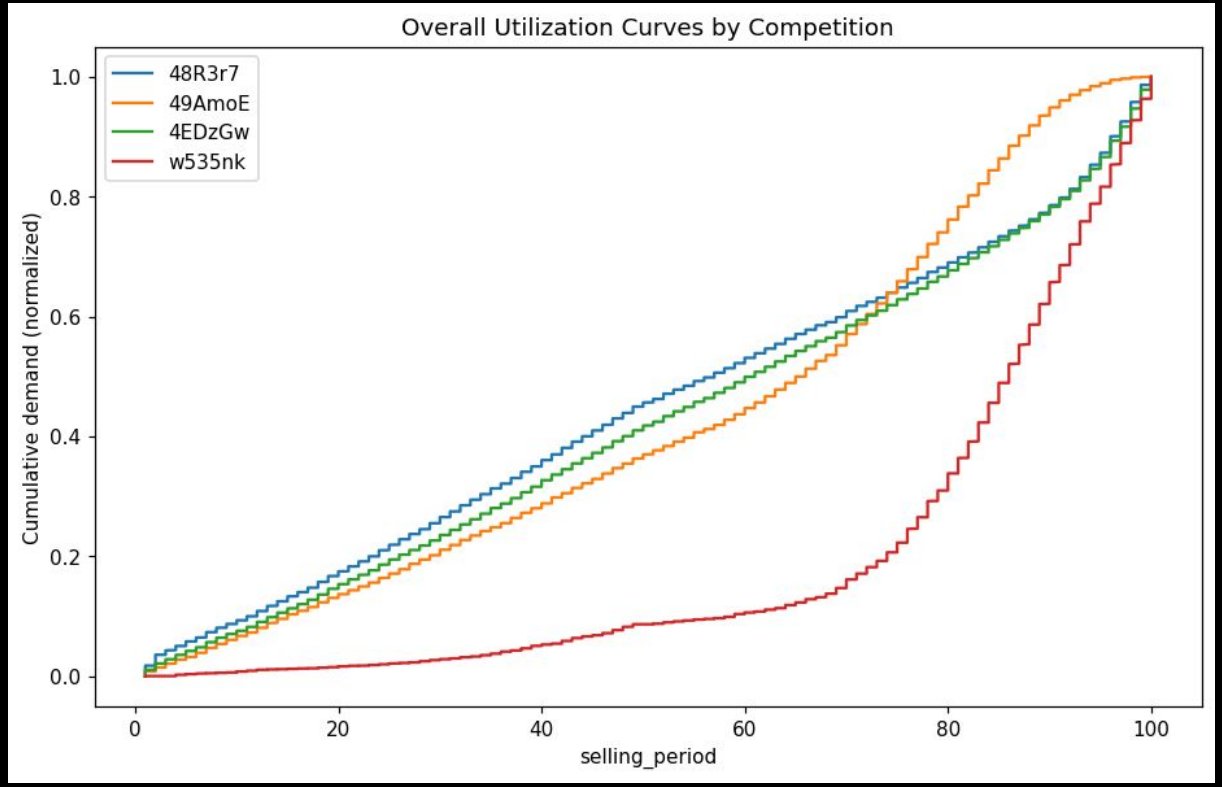
```
def next_price_estimator(demand_ranges, period, stock_left, days_left, lower_demand, upper_demand):
  expected_demand_to_sell_for = estimate_next_need_demand(period, stock_left)
  lower_demand = np.floor(expected_demand_to_sell_for)
  upper_demand = np.ceil(expected_demand_to_sell_for)
  upper = demand_ranges[lower_demand]
  lower = demand_ranges[upper_demand]
  stock_per_day = stock_left/days_left
  percentage = stock_per_day - np.floor(stock_per_day)
  return (upper - lower) * percentage + lower
```

80

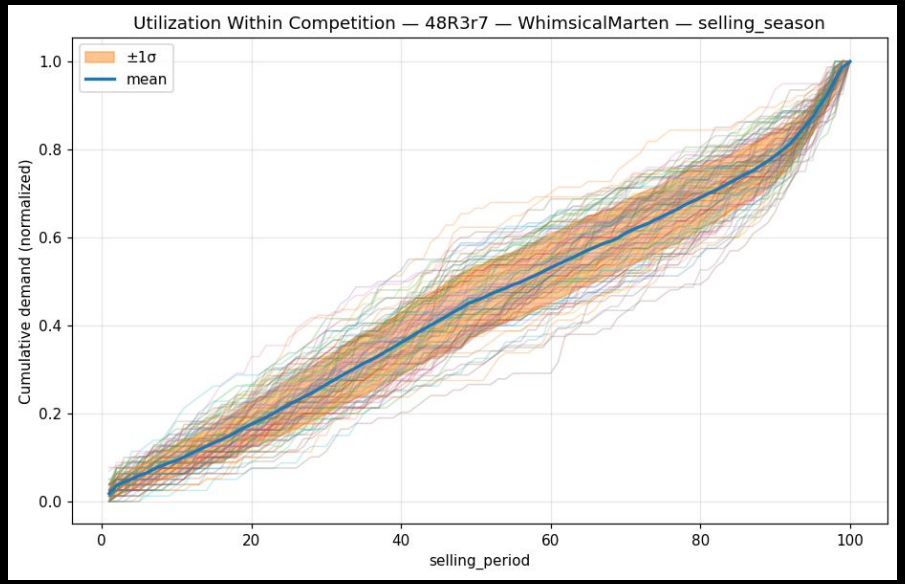
60

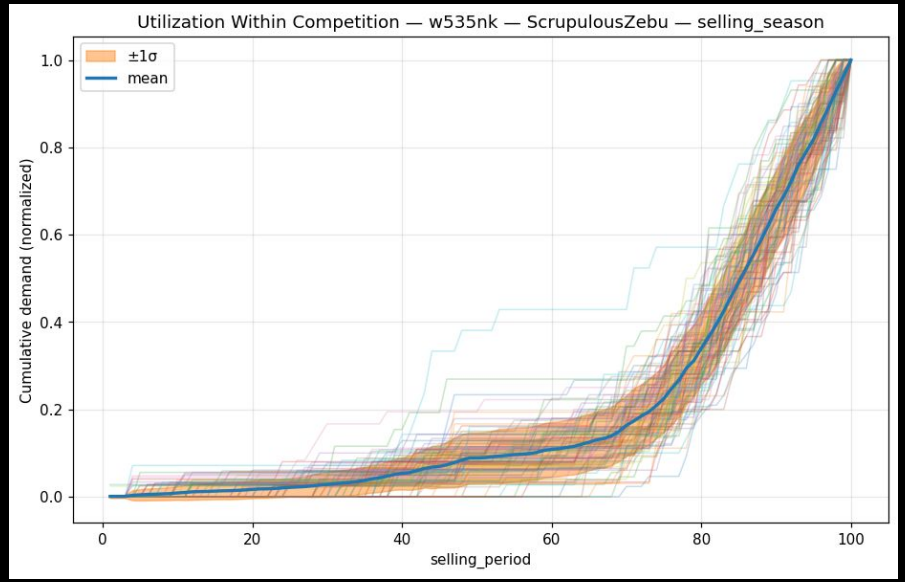
Are we closer to upper or to lower?

Capacity Curves (Raph)

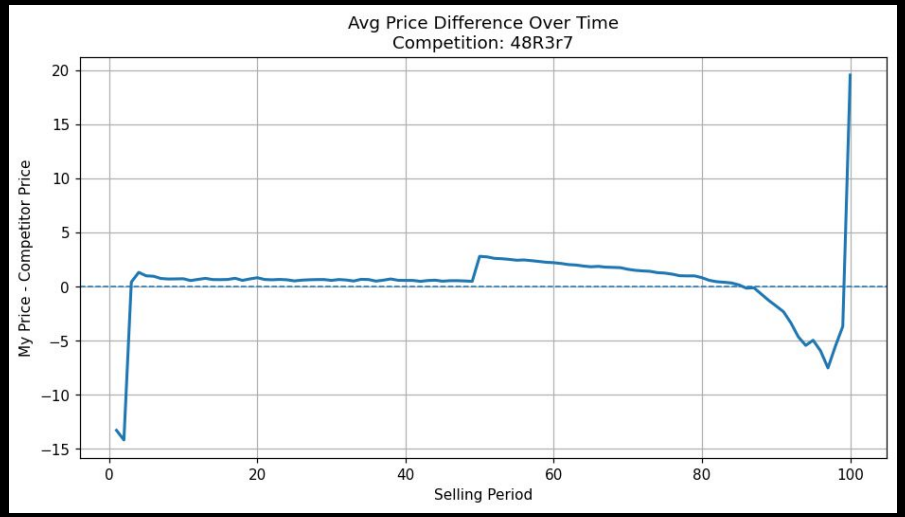


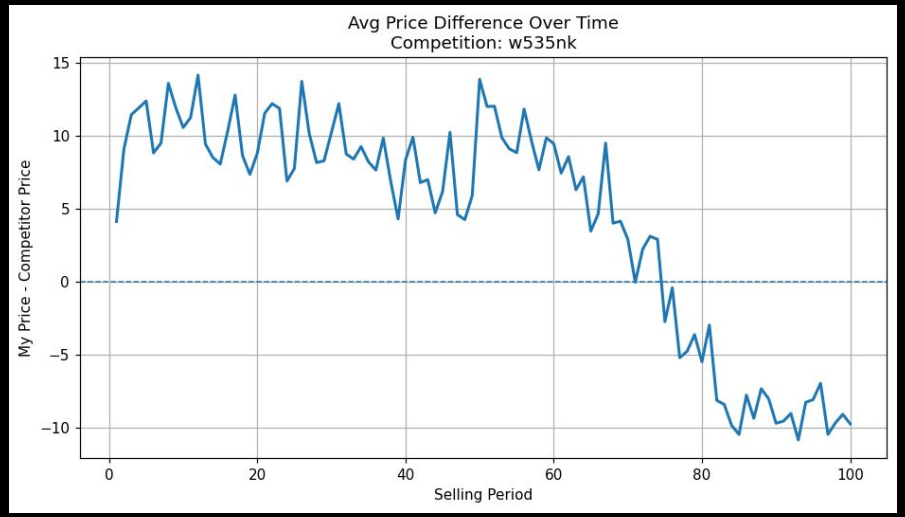
Capacity Curves - differences (Raph)



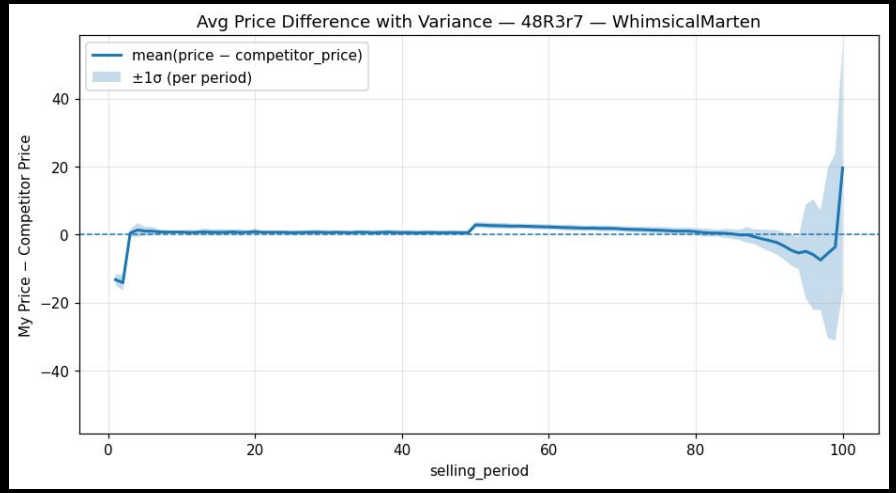


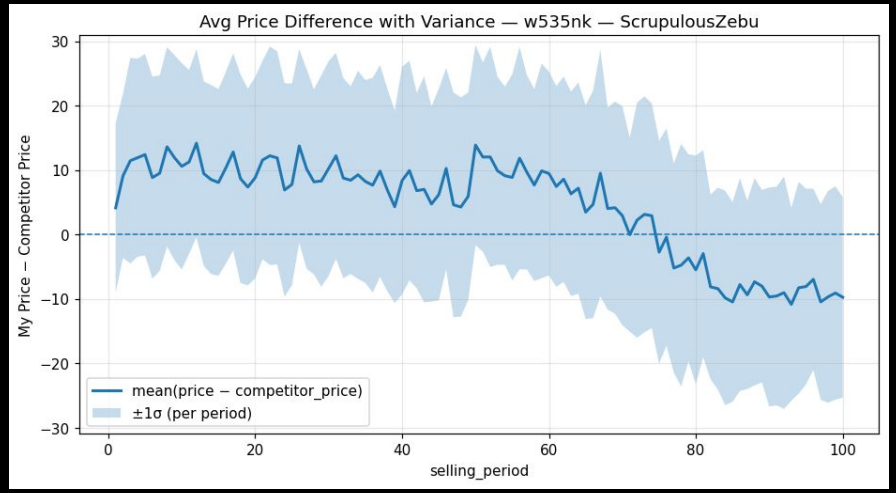
Capacity Curves - price differences (Raph)



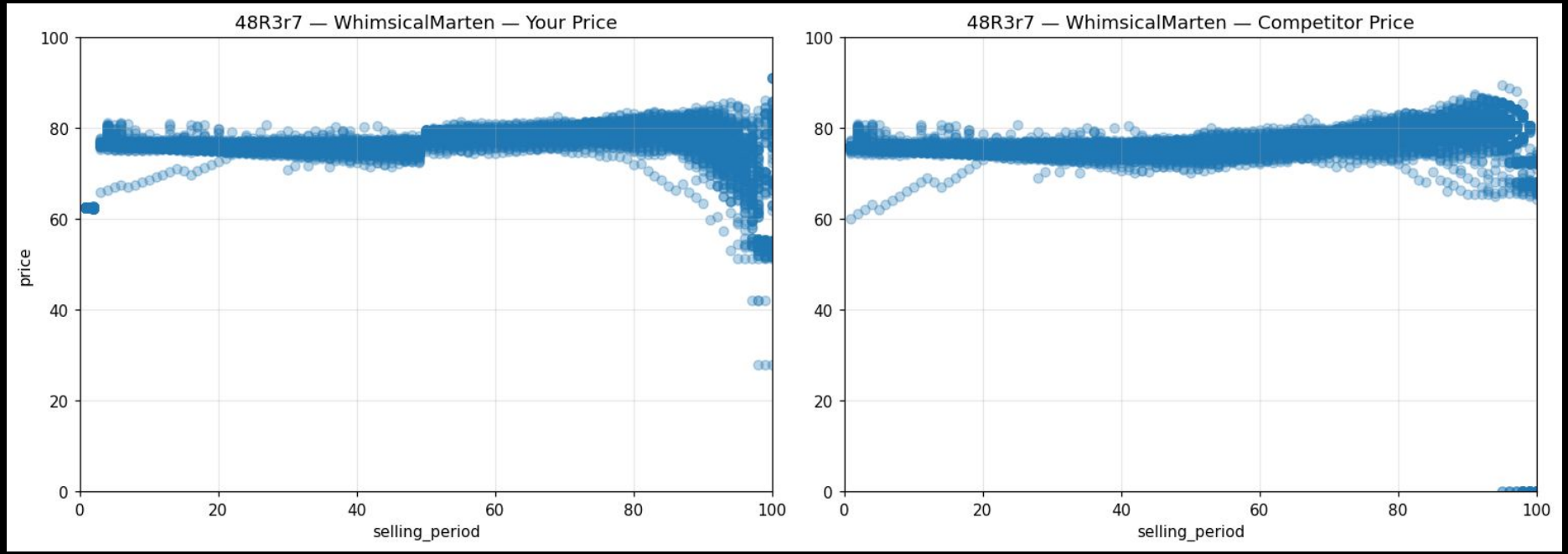


Capacity Curves - price differences (Raph)

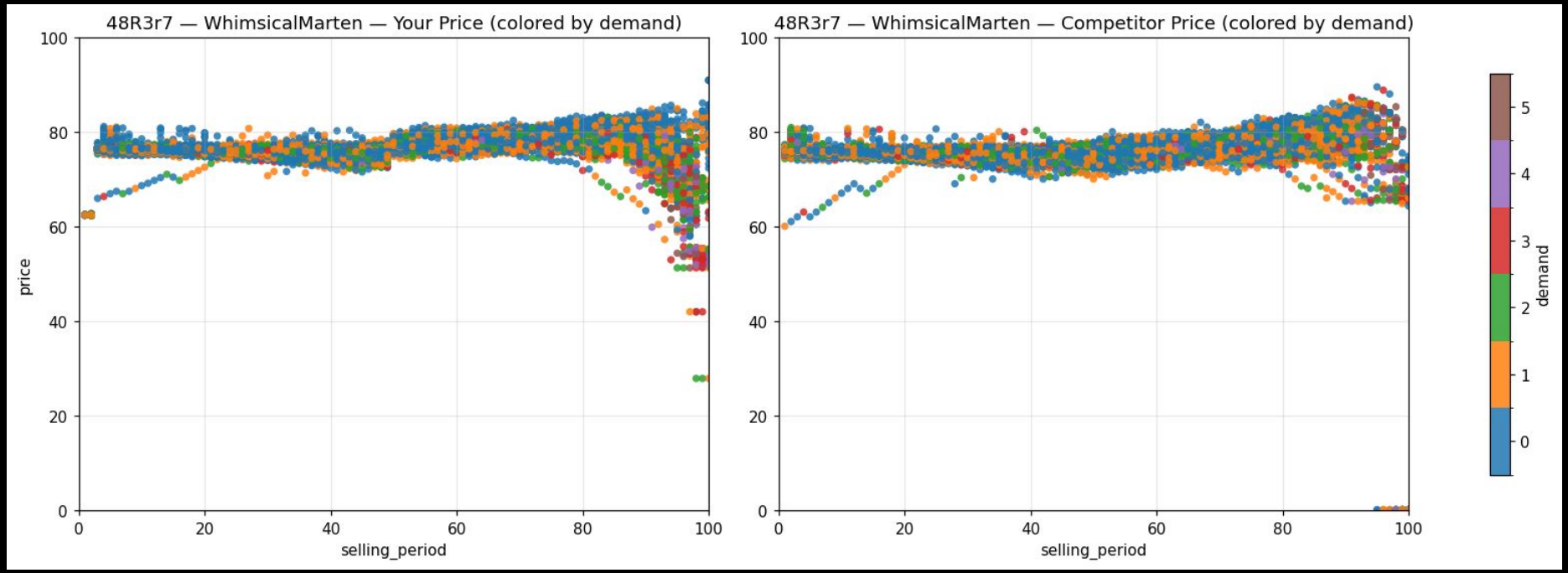




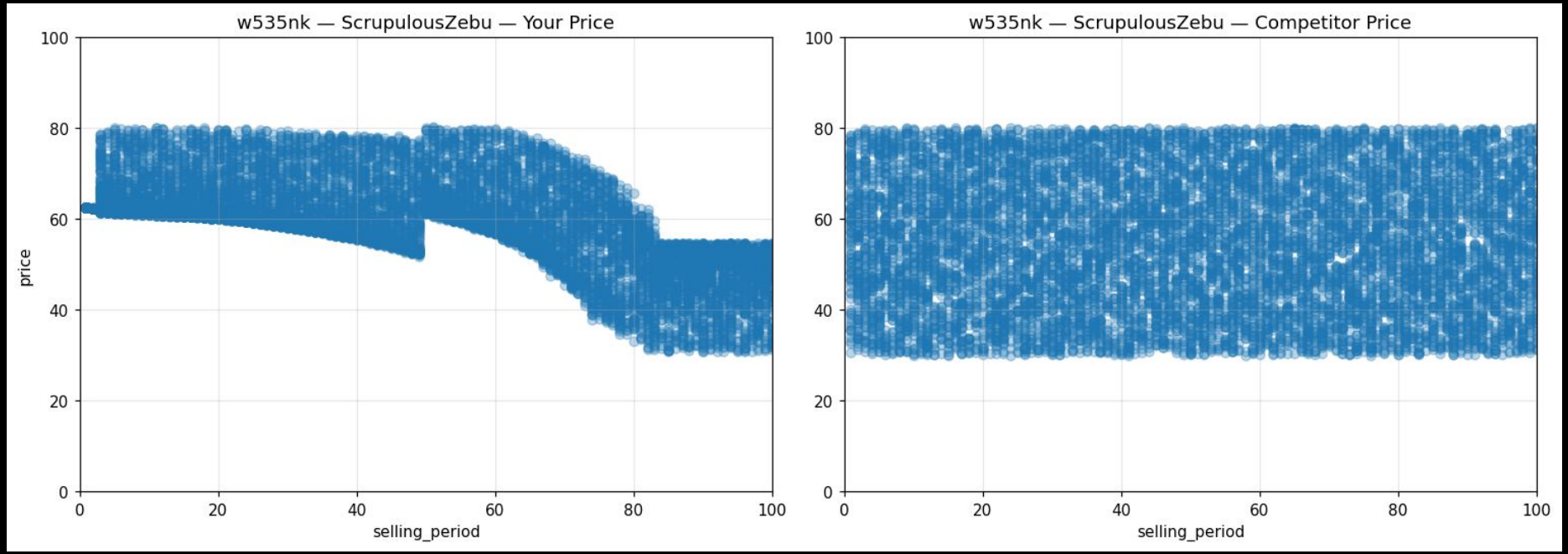
Capacity Curves - price distributions - good case (Raph)



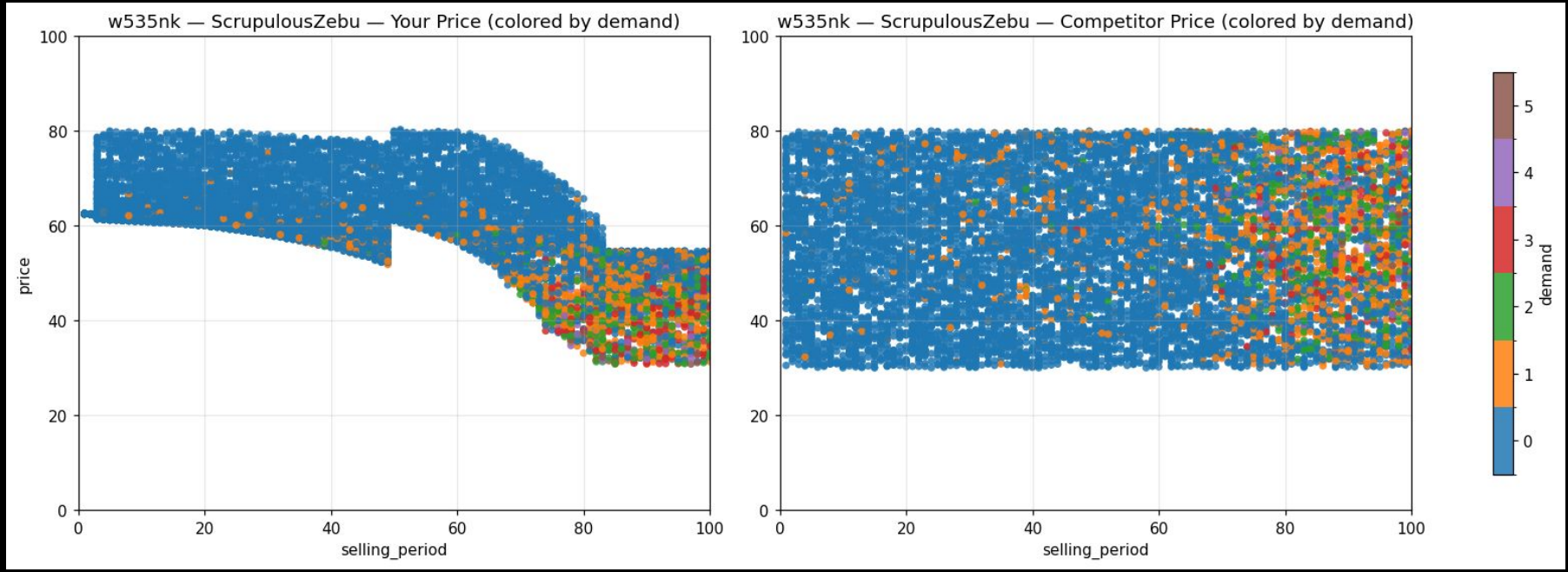
Capacity Curves - price distributions - good case (Raph)



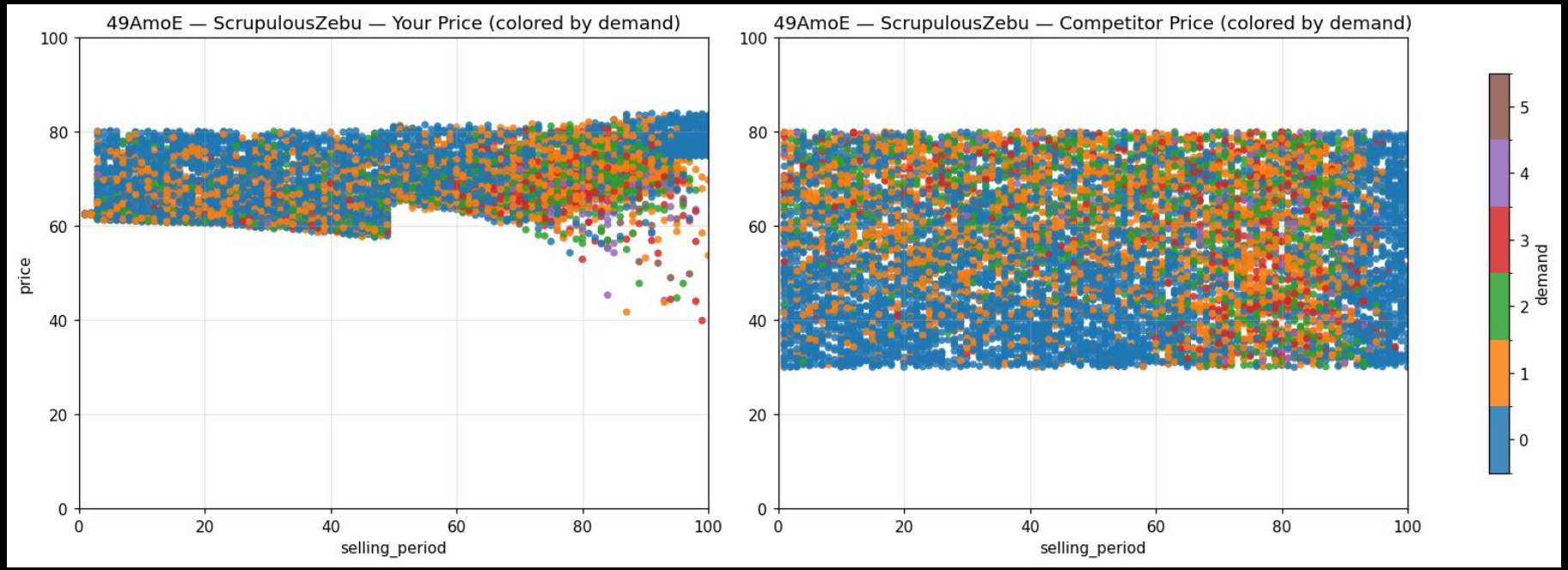
Capacity Curves - price distributions - bad case (Raph)



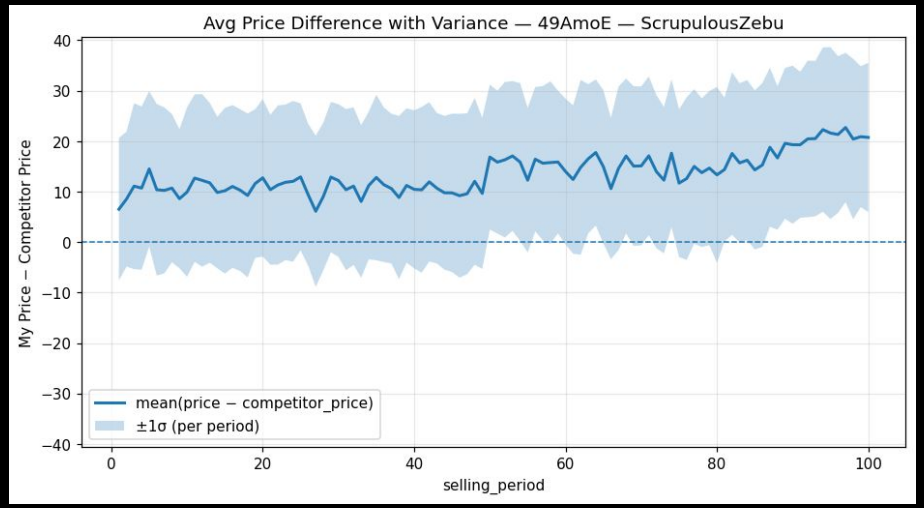
Capacity Curves - price distributions - bad case (Raph)

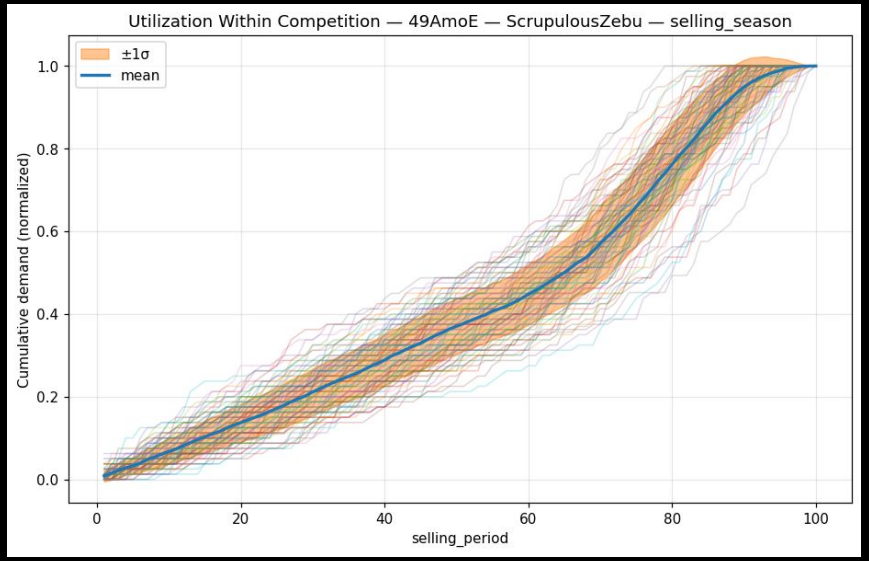


High Demand vs Random Competitor (Raph)

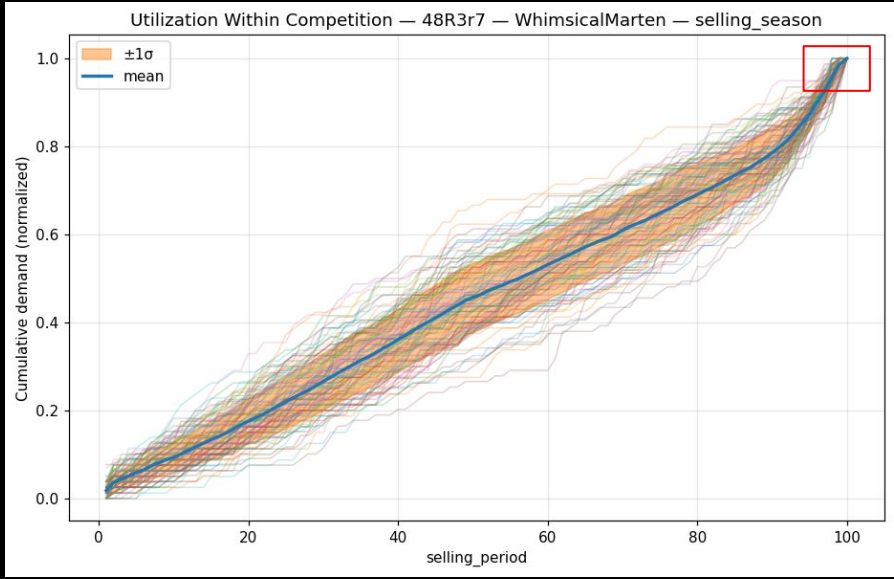


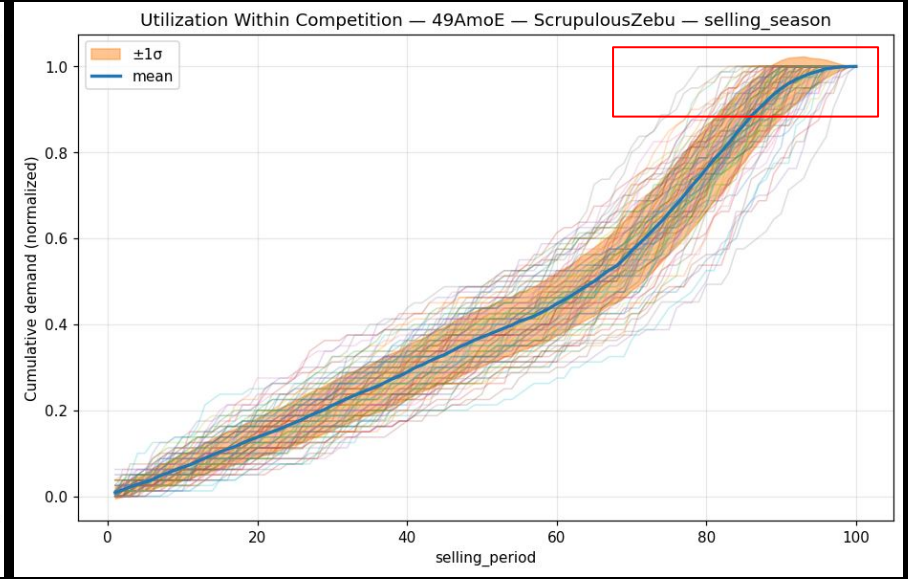
High Demand vs Random Competitor (Raph)



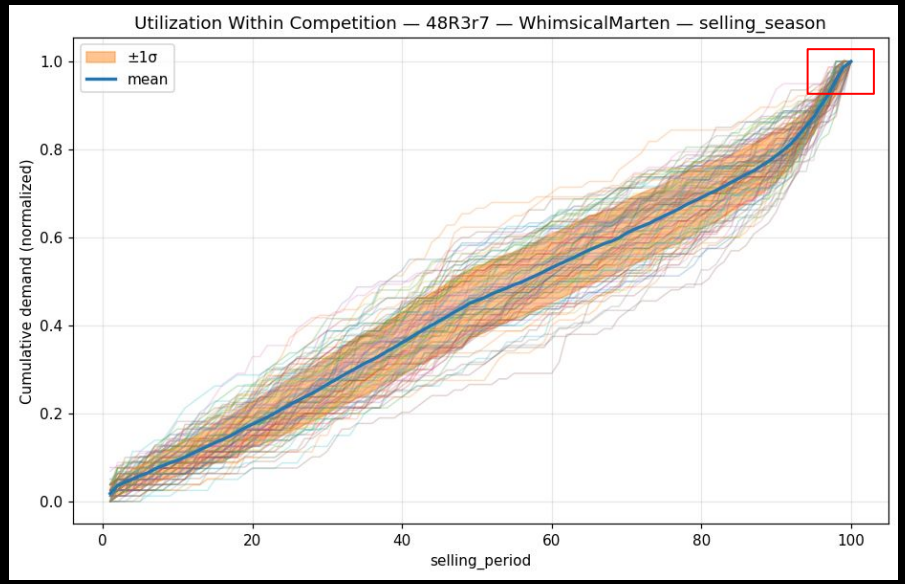


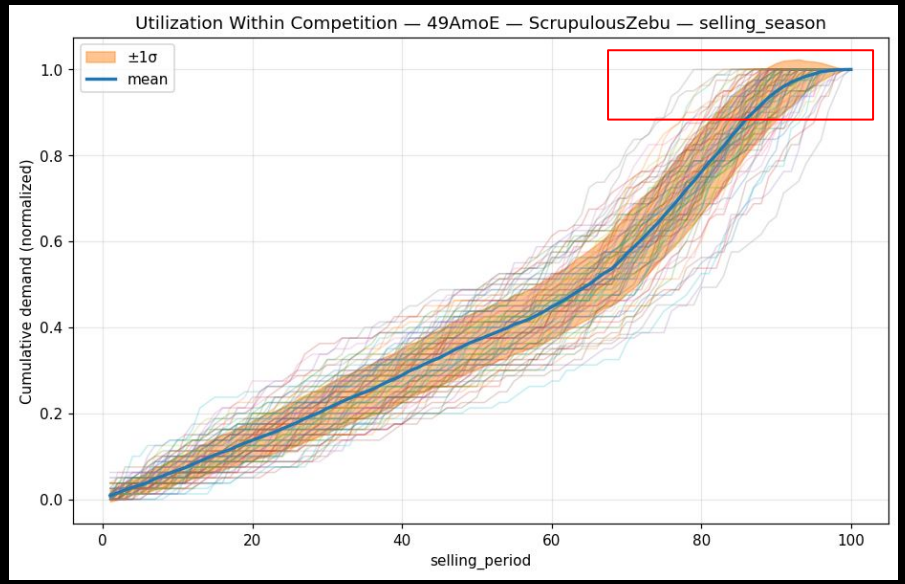
Difference to Random in high Demand (Raph)





Difference to Random in high Demand (Raph)





Difference probably because of adjustments to competitor

Conclusion (Raph)

Next Improvement:

- The base prices in the dictionary need be be set dynamically
- Better Competitor adjustments

Thank you for your attention