Presentation 4
RL + Best Performance

Group: WiseGoose-IntrepidBeluga

Members: Nicholas Chandler, Raphael Bergner

Agenda

01 Nick’s Approaches

1. RL - Q-Learning
2. Review of Best Competitions
3. Conclusion

02 Raphael

Q-Learning vs. XGBoost

Nick

RL - Q-Learning

- **What is Q-Learning?**
- Q-Learning seeks to find an optimal policy by maximization of the Q function:

$$Q^{\pi}(s,a) = \mathbb{E}_{\pi}\left[\sum_{t=0}^{\infty}\gamma^{t}r_{t} \,\middle|\, s_0 = s,\ a_0 = a\right]$$

Q-Function

- The Q function tells us what the expected reward (given a policy) is at any state, s, when an action, a, is taken.
- Arg Maximization of the Q function yields the optimal policy

- In order to maximize the Q function, we use the following update rule:

$$Q(s,a) \leftarrow Q(s,a) + \alpha\left(r + \gamma \max_{a'} Q(s',a') - Q(s,a)\right)$$

Update Rule

- In practice, we tabulate the state action pairs and store the Q values in each cell of the table.
- Each update iterates over all state-action pairs

RL - Q-Learning

- Here we implemented Q-Learning applied to the dynamic pricing competition
- Essentially, we follow what was shown in class but expand the state set

- **Action Set:** Prices from 5 to 95 in increments of 5
- **State Set (Cartesian Product of the two):**
  - Competitor prices from 10 to 90 in increments of 10
  - Units sold from 10 to 80 in increments of 10
- **Reward:** Revenue gained

- **Episodes:** 10
- **Training Data:** 1.8 million rows of DPC data

Decision Table →

	competitor_price	units_sold_bin	optimal_price
0	10	10	20
1	10	20	70
2	10	30	75
3	10	40	25
4	10	50	65
...	...	...	...
67	90	40	70
68	90	50	60
69	90	60	75
70	90	70	60
71	90	80	65

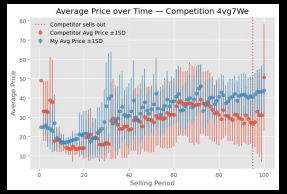
```
# Training the Q-learning model
for episode in range(episodes):
    for idx in tqdm(range(len(df_prices))):

        # Extract state and action from the current row
        # map observed prices to our discrete state & action space
        competitor_price = find_closest( df_prices.loc[idx, "price_competitor"], comp_price_values )
        my_price = find_closest( df_prices.loc[idx, "price"] )

        units_raw = df_units_sold.loc[idx, "units_sold"]
        units_bin = find_closest_bin(units_raw, units_sold)

        # get observed demand
        demand = df_demand.loc[idx, "demand"]

        state = (competitor_price, units_bin)
        state_idx = state_index_map[state]
        action_idx = price_index_map[my_price]

        # Calculate reward
        reward = calculate_reward(my_price, demand)

        # Get the best next action (staying in the same state for simplicity)
        best_next_action = np.max(q_table[state_idx])


        # Update Q-value
        q_table[state_idx, action_idx] += alpha * (
            reward + gamma * best_next_action - q_table[state_idx, action_idx]
        )
```

RL - In DPC

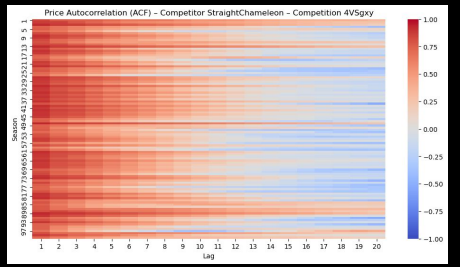
- Basically, forecast the competitor's price (ETS) and look up the nearest price value in the Q-table given our units sold so far.
- Does not account for the stock of the competitor or the period in the season. This is future work but may also need more competition data.

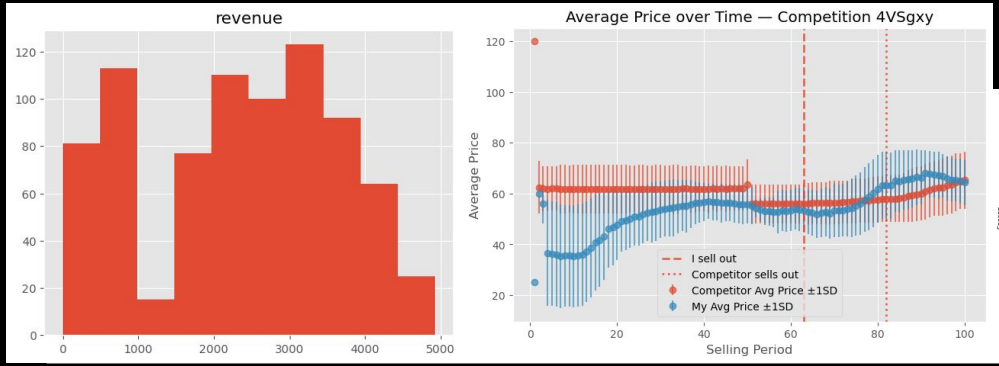
```
# --- Find closest state in Q-table ---
closest_key = find_closest_state(comp_price, units_sold_so_far)

# --- Lookup optimal price from Q-table ---
best_price = q_price_lookup.get(closest_key, BASE_PRICE)
```

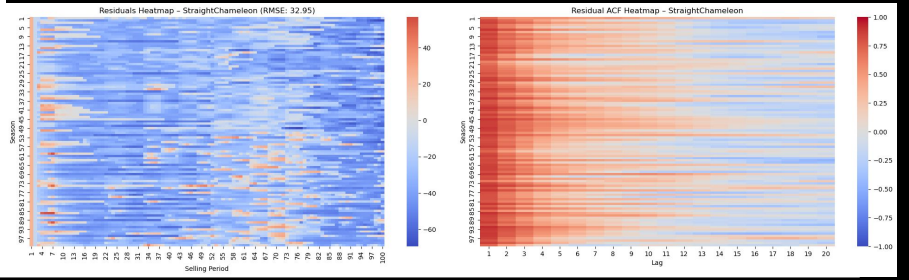
RL - DPC Results



- The results from 01/28 are as follows:
- The best revenue competition had the competitor pricing around a very high point with little variance
- The competition 4VSgxy was best with revenue: $386,520.00
- Competition 4vg7We was against Wise Goose... My teammate!
- From the price ACF plot we see strong correlation at small values of lags and slow tendency towards 0
- The residual heatmap shows we undershoot with our predictions



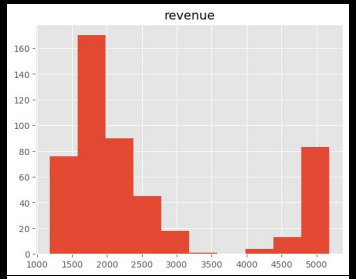


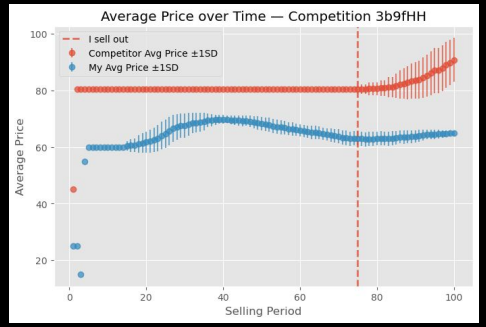


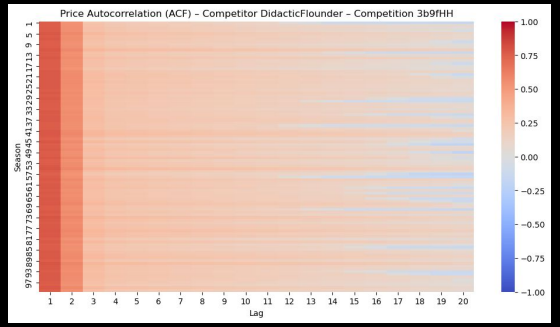
4	IntrepidBeluga	$1,894,070	$236,759	1.5	$15.00	$75.00	$44.20	0.833	0.198

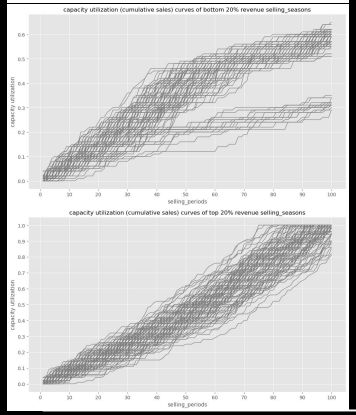
RL - DPC Results

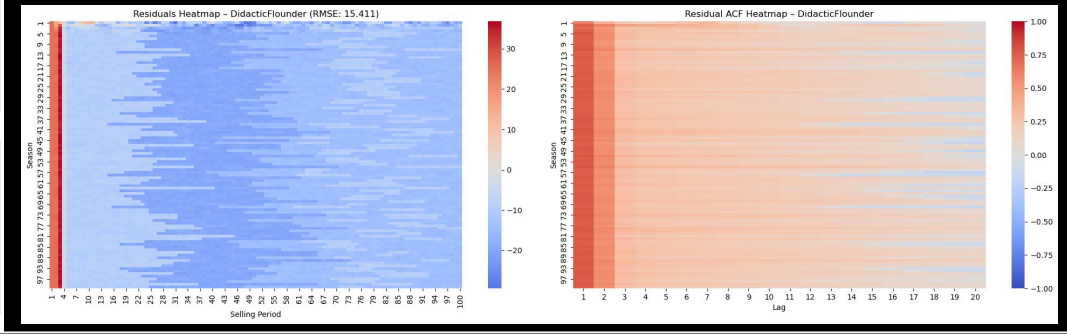
- The results from 01/31. Best competition.

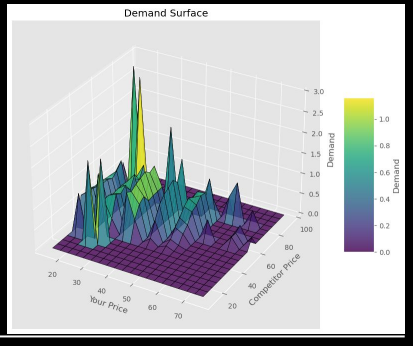








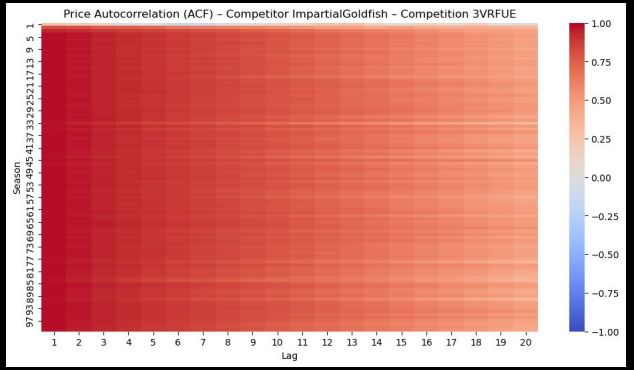


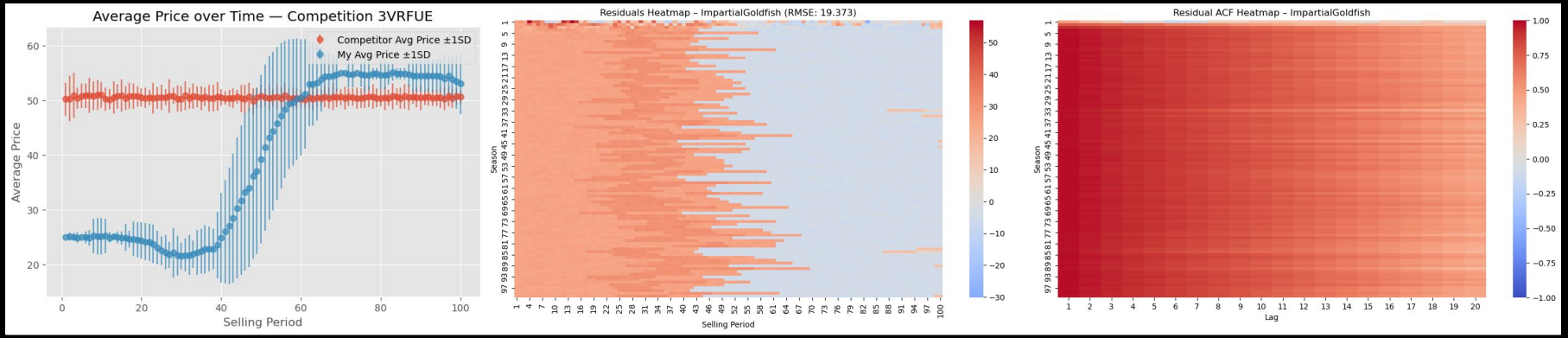


1	IntrepidBeluga	$1,270,140	$254,028	1.2	$15.00	$75.00	$49.36	0.833	0.132

RL - DPC Results

- The results from 01/31. Worst competition.
  - Worst Revenue: $166,245.00
  - Best Revenue: $497,605.00
- Given the price ACF here, we should have been able to predict better.
- This competitor seems to choose random values the first two seasons and then stay at a price near $50 for the remainder.
  - In the past they actually tend to perform well.



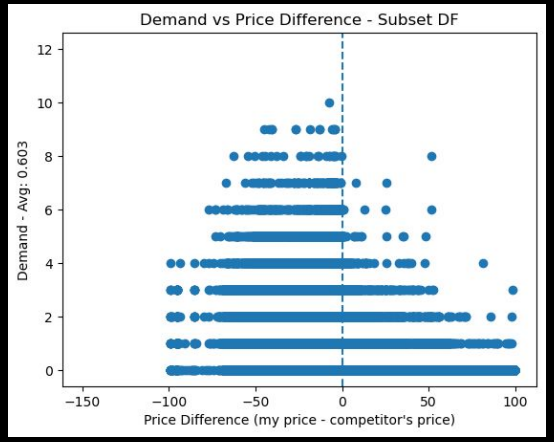


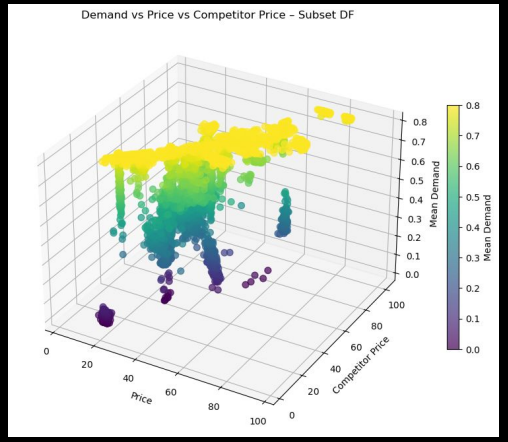
1	IntrepidBeluga	$1,270,140	$254,028	1.2	$15.00	$75.00	$49.36	0.833	0.132

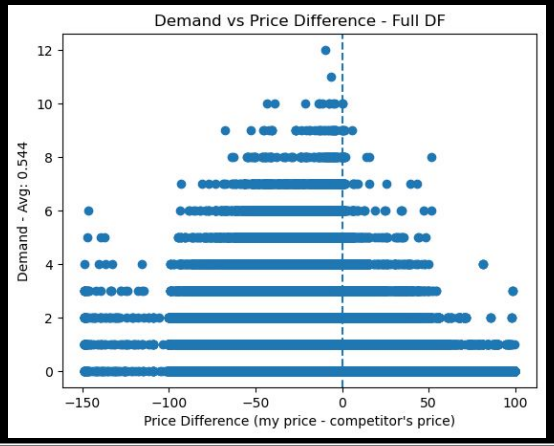
Best Performance

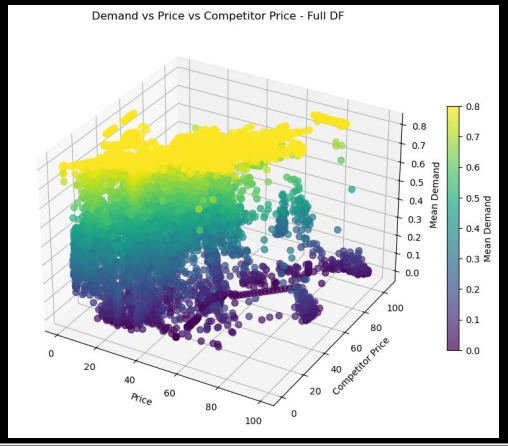
- Given that there is a ranking present in the DPC, we utilize this to measure performance.
- Specifically, we look at all competitions where my user (IntrepidBeluga) is in the top 10 and analyze trends.
- The dates are:
  - 2025-10-20 - Game Theoretic Approach
  - 2025-11-17 - LAD Demand Model
  - 2025-12-10 - Adversarial ETS
  - 2025-12-17 - Adversarial ETS
  - 2025-12-22 - Adversarial ETS
  - 2025-12-31 - Adversarial ETS
  - 2026-01-26 - Neural Network
  - 2026-01-28 - Q-Learning
  - 2026-01-31 - Q-Learning
- In the following the subset data frame refers to the data frame containing competitions from the best dates.

Best Performance





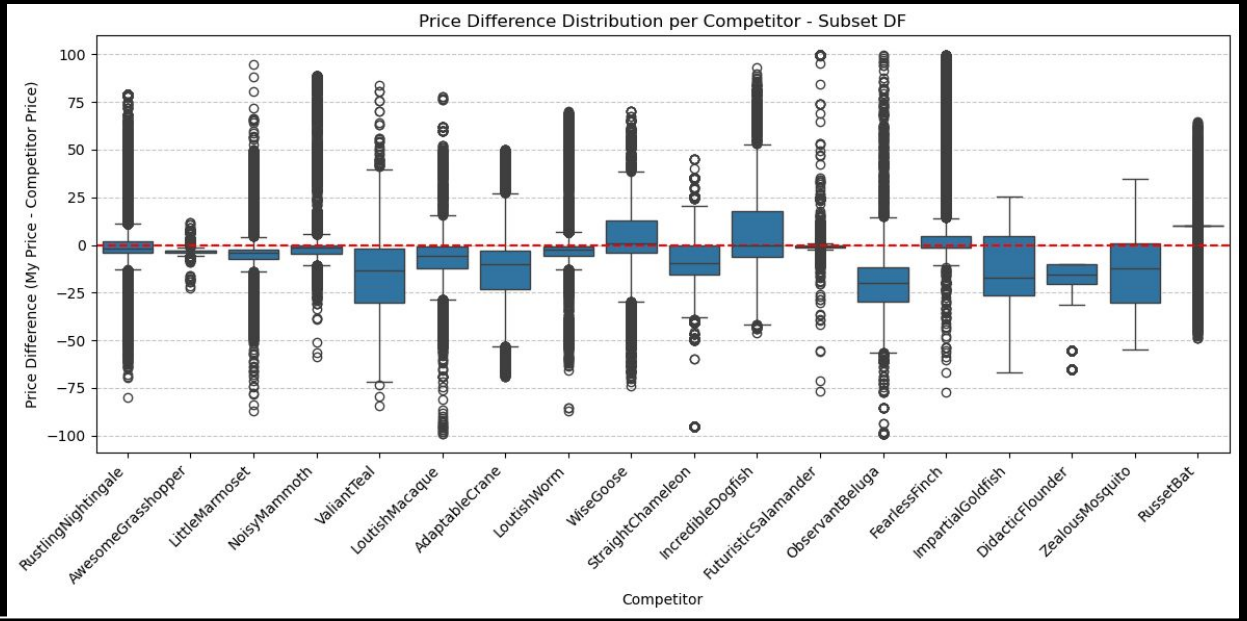




Best Performance

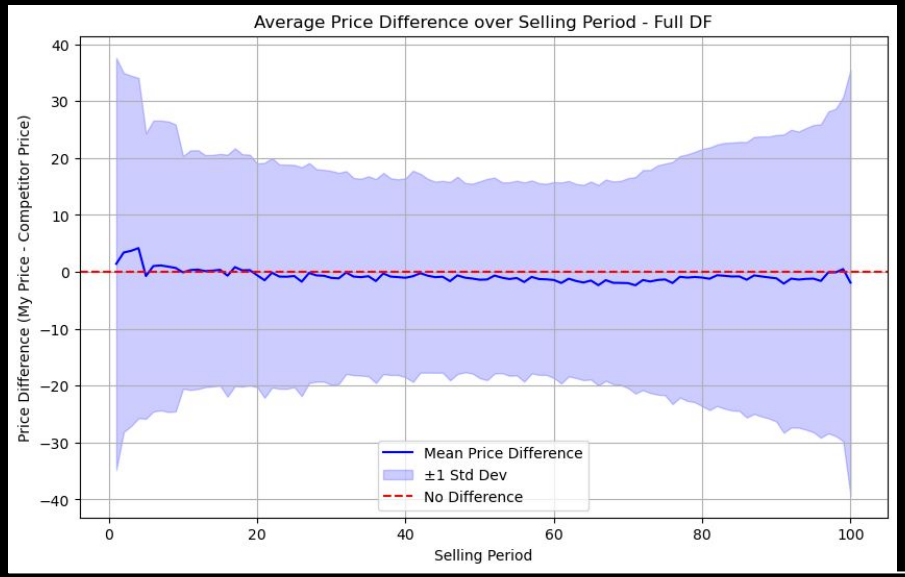
- The frequencies of competitors are given.
- We also see the distributions of price difference split on competitors.

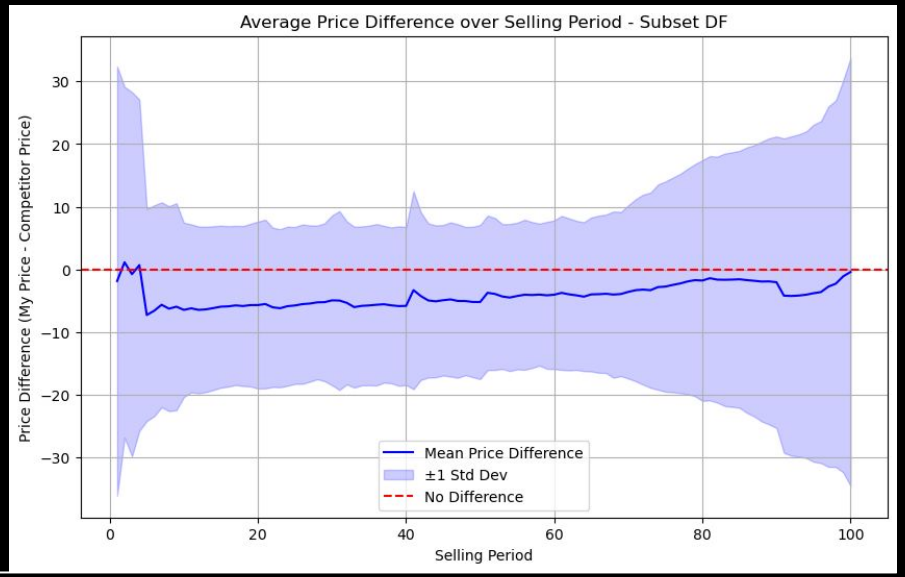
	competitor_id	num_unique_competitions
0	RustlingNightingale	6
1	LoutishMacaque	4
2	LoutishWorm	4
3	IncredibleDogfish	3
4	LittleMarmoset	3
5	AdaptableCrane	2
6	AwesomeGrasshopper	2
7	FuturisticSalamander	2
8	NoisyMammoth	2
9	DidacticFlounder	2
10	StraightChameleon	2
11	ValiantTeal	2
12	ObservantBeluga	2
13	RussetBat	2
14	WiseGoose	2
15	ZealousMosquito	2
16	FearlessFinch	1
17	ImpartialGoldfish	1



Best Performance

- Here we examine the average price difference per time step.
- There is a larger difference where the competitor prices higher in the subset data frame .





Conclusions

- An approach to the next DPC season could be to send out a bunch of duopoly programs which explore many data combinations and then train a strong RL agent on these data.
- Pricing under the competitor correlates with increased competition performance
- Simpler approaches tended to work best in my analysis.
- Even if the model is not completely correct, performance in the competition can still be good.
- Question: When will we know the results from the overall DPC?

Raphael

Data - cleaning

```
# Cleaning rules
SOLD_OUT_PERIOD_CUTOFF = 80    # remove seasons where sould out before period 80
MIN_SOLD_PER_SEASON = 60       # remove seasons where less than 60 units sold
COMP_SOLDO_RATE_MIN = 0.10     # remove competition if competitor sold out in <10% seasons
COMP_SOLDO_RATE_MAX = 0.50     # remove competition if competitor sold out in >50% seasons
```

Data - additional features

```
def period_bin(p: pd.Series) -> pd.Series:
    # 1-20, 21-40, 41-60, 61-80, 81-100 -> labels 0..4
```

```
def stock_left_bin(stock_left: pd.Series) -> pd.Series:
    """
    Remaining stock bins: 80:60, 60:40, 40:20, 20:0.
    Encoded:
      0: (60,80]
      1: (40,60]
      2: (20,40]
      3: (0,20]
    """
```

QLearning - Parameters

```
price_values = [5, 10, 15, 20, 25, 30, 35, 40, 45, 50, 55, 60, 65, 70, 75, 80,
85, 90, 95, 100]

comp_price_values = [10, 20, 30, 40, 50, 60, 70, 80, 90]
period_bins = [0, 1, 2, 3, 4]
stock_left_bins = [0, 1, 2, 3, 4]
competitor_has_capacity_states = [0, 1]

# Q-Learning parameters
alpha = 0.05
gamma = 0.95
episodes = 15
```

QLearning - Sparse

```
competitor_price_state,period_bin,stock_left_bin,competitor_has_capacity,optimal_price,q_value
10,1,0,0,5,0.0
10,1,0,1,25,127.68865873489413
10,1,1,0,5,0.0
10,1,1,1,5,19.719930113057888
10,1,2,0,5,0.0
10,1,2,1,5,0.0
10,1,3,0,5,0.0
10,1,3,1,5,0.0
10,1,4,0,5,0.0
10,1,4,1,5,0.0
10,2,0,0,5,0.0
10,2,0,1,5,96.6931099131614
```

QLearning - Sparse Interpolation

```
competitor_price_state,period_bin,stock_left_bin,competitor_has_capacity,optimal_price,q_value
10,1,0,0,25,176.69154059793834
10,1,0,1,25,176.69154059793834
10,1,1,0,65,189.00647274589065
10,1,1,1,5,70.0902720627384
10,1,2,0,80,314.0175746537975
10,1,2,1,5,62.107443587011446
10,1,3,0,70,206.40639564102287
10,1,3,1,5,61.310876358623254
10,1,4,0,70,180.8433309648089
10,1,4,1,5,62.02554753760482
10,2,0,0,70,151.48666970890943
10,2,0,1,25,151.0374369725856
```

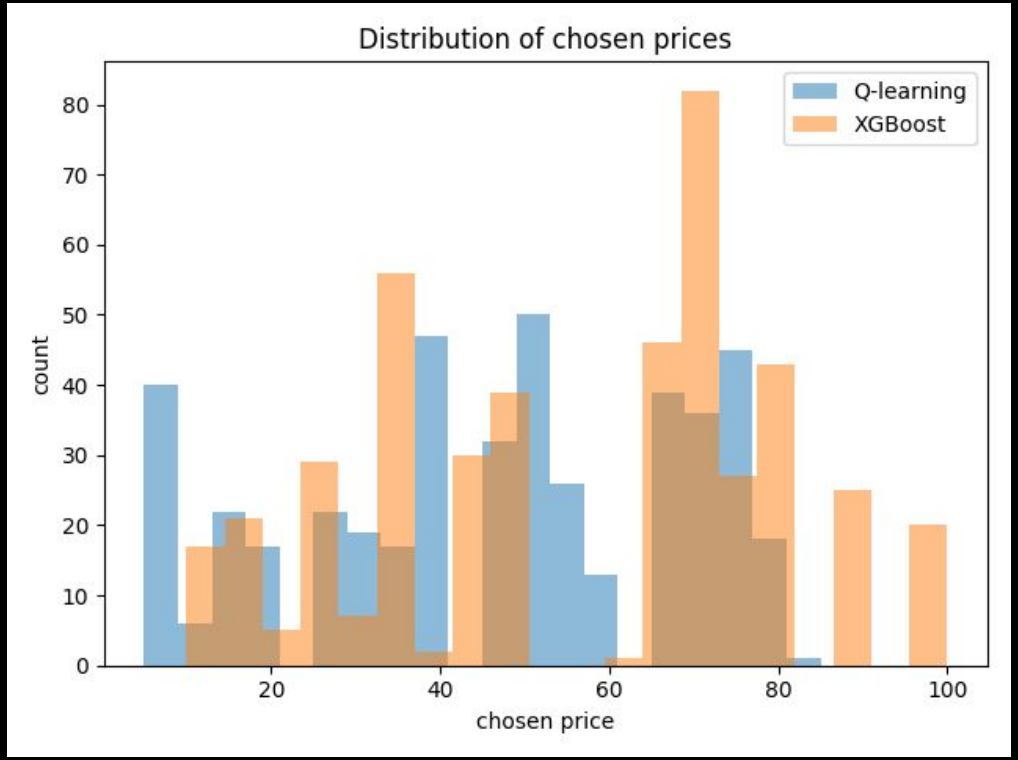
XGBoost - Parameters

```
FEATURES_ALL = ["price_ratio", "price", "abs_diff", "competitor_has_capacity", "period_bin",
"stock_left_bin"]


TARGET = "revenue"
```

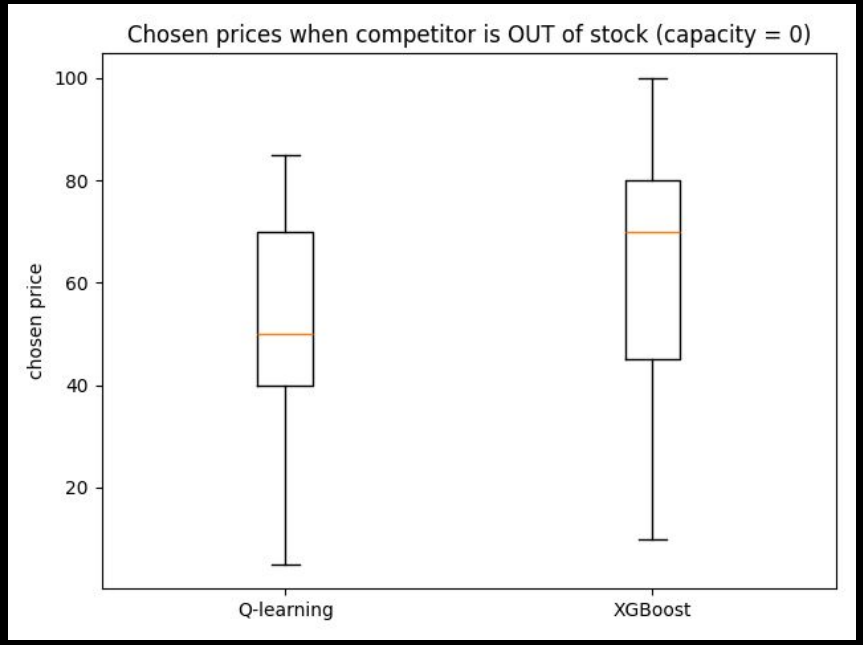
XGBoost - Policy

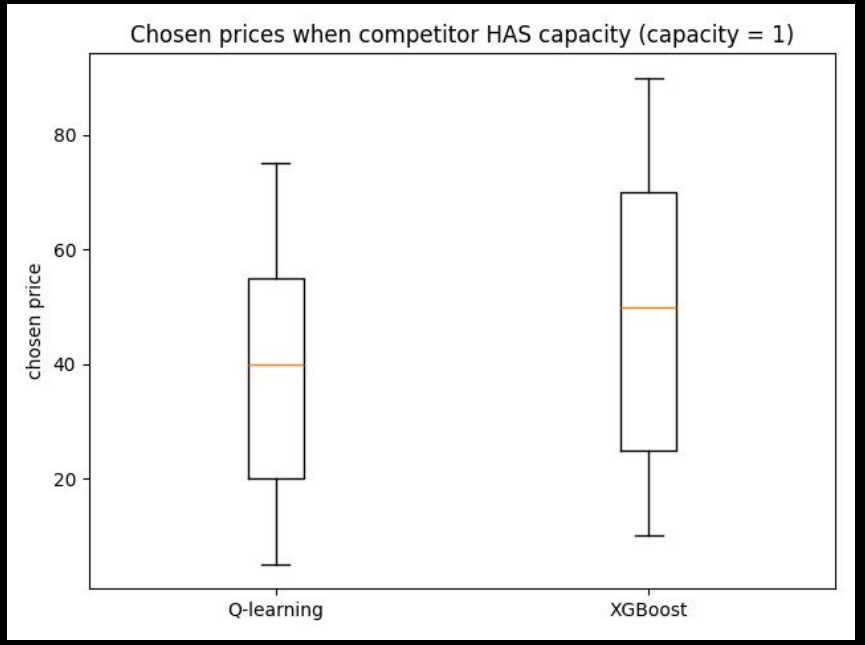
```
competitor_price_state,period_bin,stock_left_bin,competitor_has_capacity,optimal_price,q_value
10,1,0,0,80,124.410888671875
10,1,0,1,10,11.000563621520996
10,1,1,0,80,123.20475769042969
10,1,1,1,10,12.965852737426758
10,1,2,1,10,13.26574420928955
20,3,1,1,15,28.950151443481445
20,3,2,0,90,55.056846618652344
30,0,4,1,80,0.24119974672794342
30,1,0,0,100,114.553955078125
30,1,0,1,25,22.485570907592773
80,1,4,1,70,38.17103576660156
80,2,0,0,70,149.9051971435547
80,2,0,1,70,94.87784576416016
```

Model - Pricing



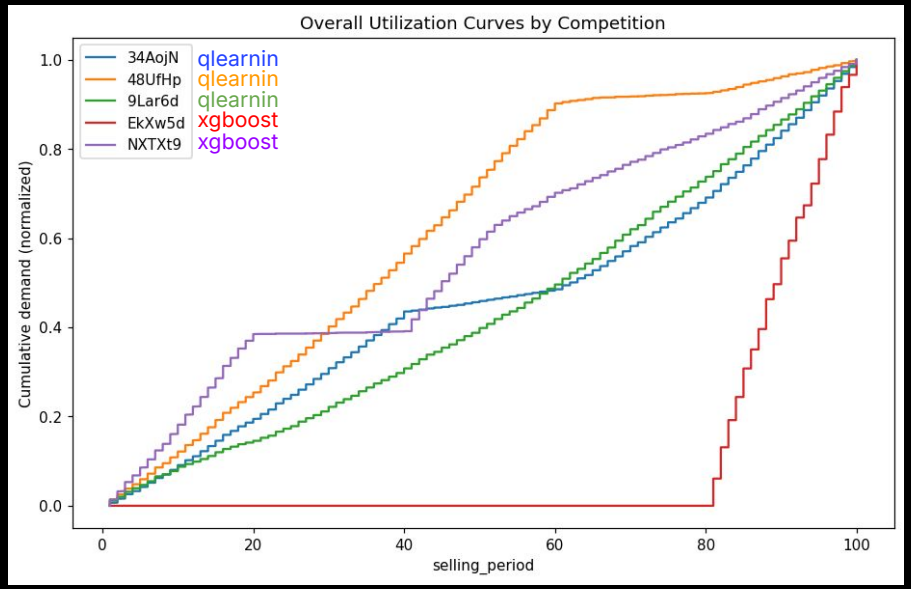
Model - Pricing



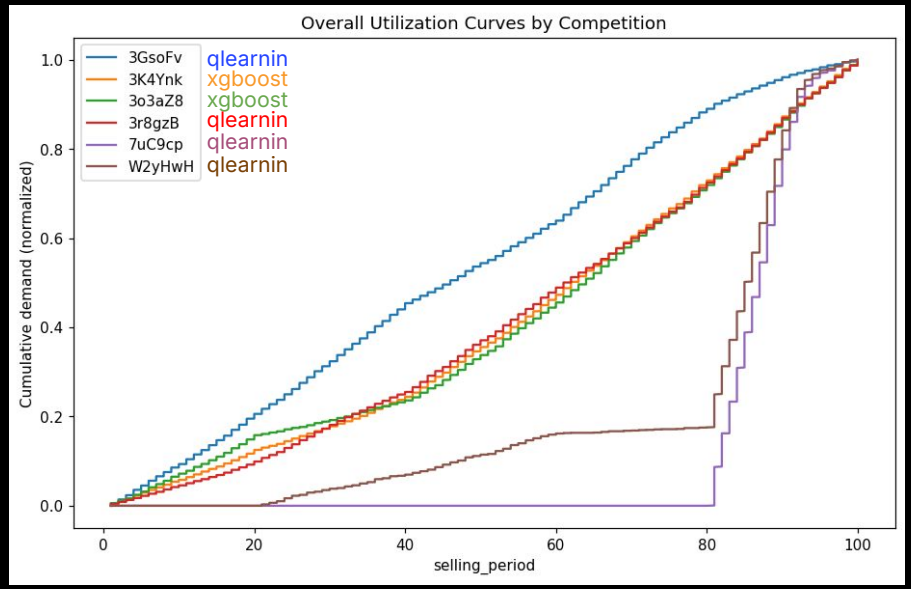


In Practice

2026-01-31

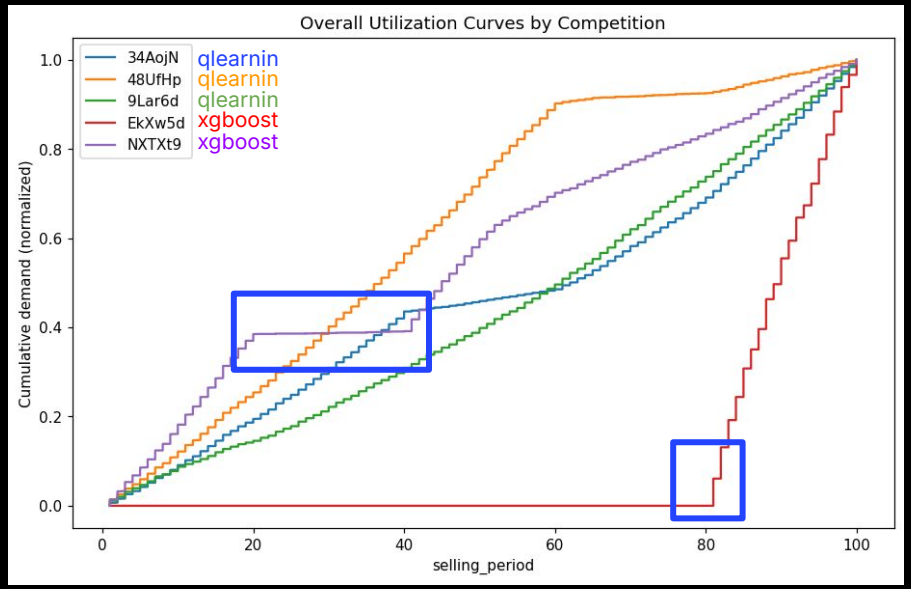


2026-02-02

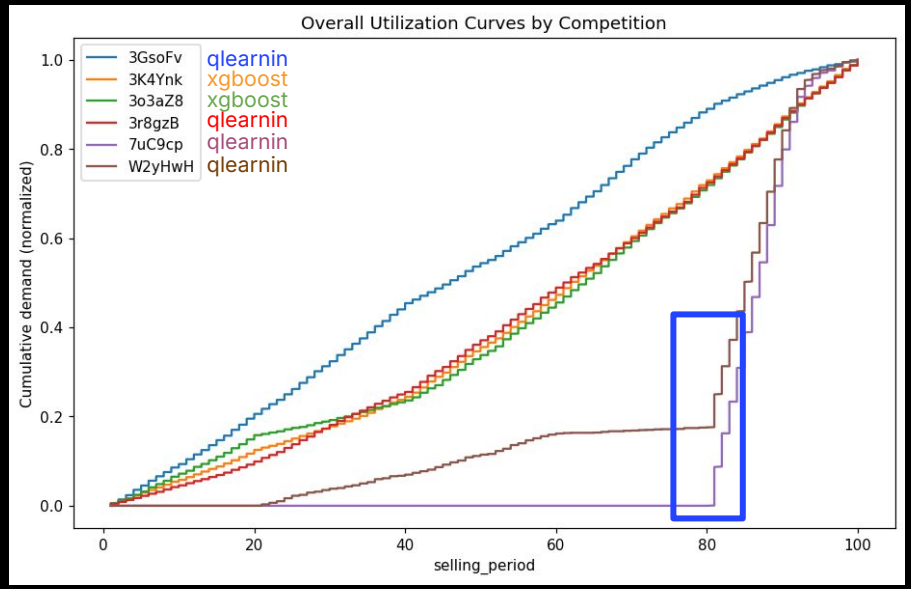


In Practice

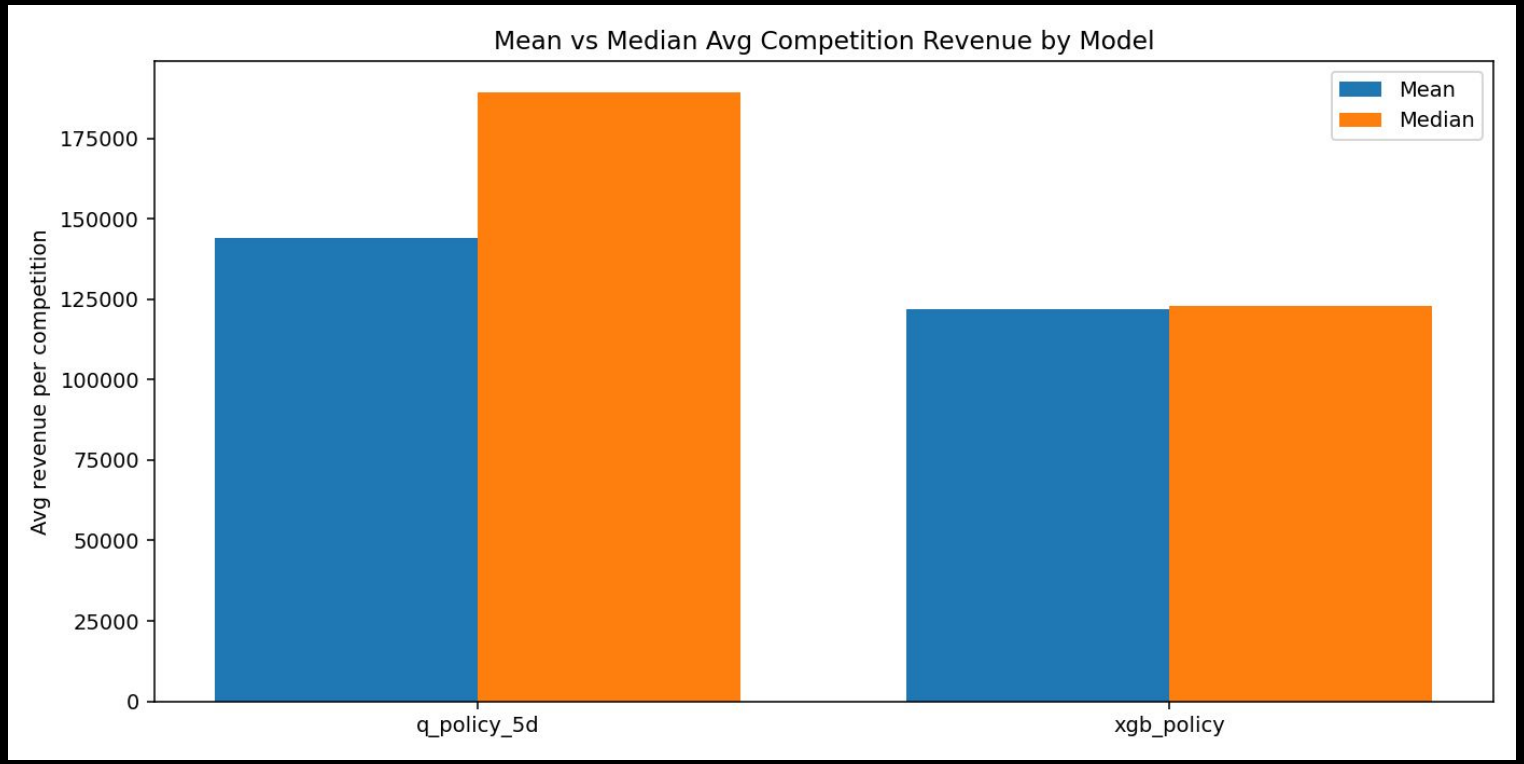
2026-01-31



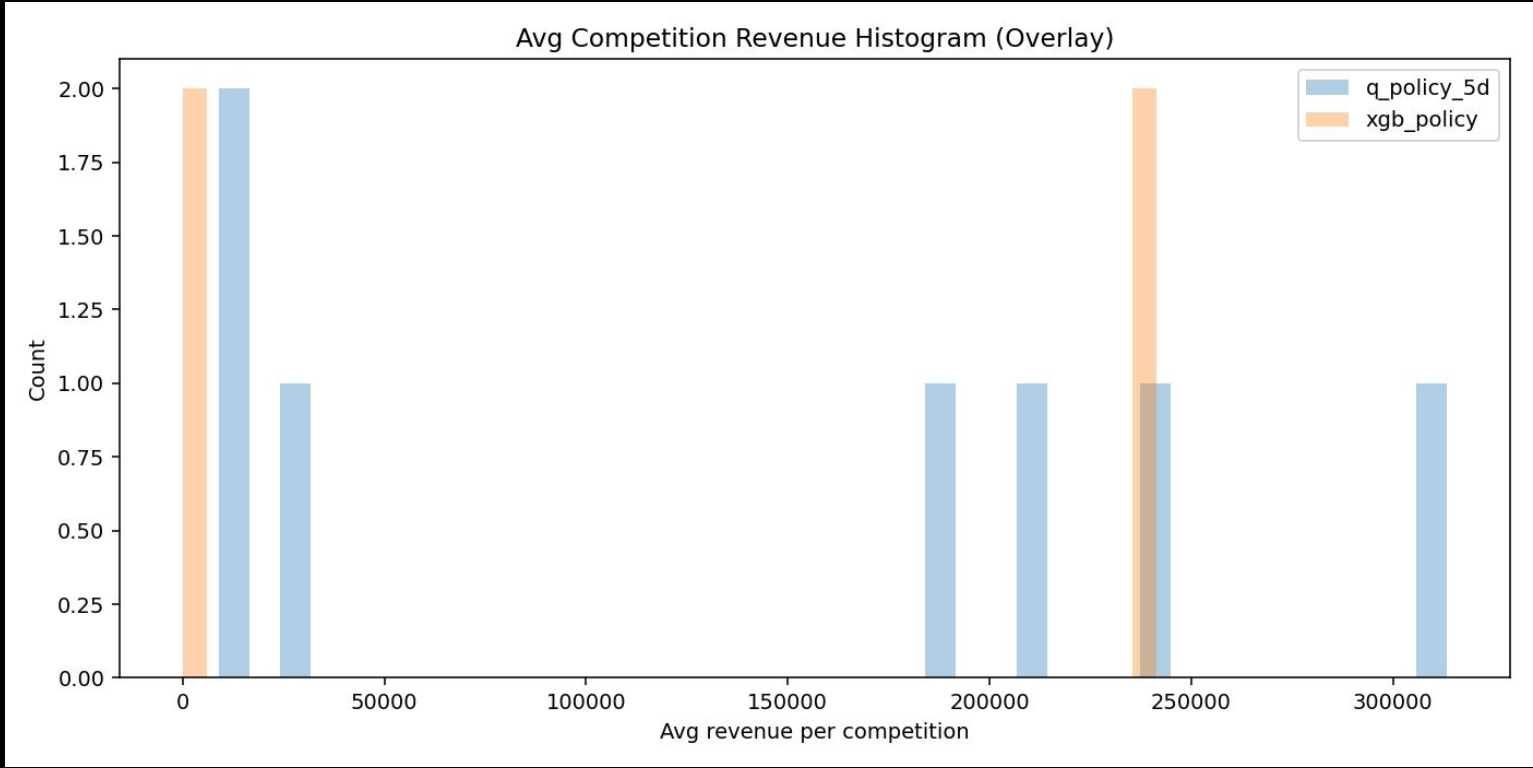
2026-02-02



In Practice


Mean vs Median Avg Competition Revenue by Model
Mean
Median
Avg revenue per competition
175000
150000
125000
100000
75000
50000
25000
0
q_policy_5d
xgb_policy

In Practice


Avg Competition Revenue Histogram (Overlay)
q_policy_5d
xgb_policy
Count
2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
0
50000
100000
150000
200000
250000
300000
Avg revenue per competition

Thanks for the course

All models are wrong, but some are useful

1976, George Box