Presentation 2 Optimization, Dynamic Programming, & Time Series Forecasting

Group: WiseGoose-IntrepidBeluga

Members: Nicholas Chandler, Raphael Bergner

Agenda

01 Nick’s Approaches

1. LAD Optimized Prices
2. Time Series Forecasting
3. Dynamic Pricing: A Learning Approach
4. Dynamic Programming

02 Which Forecast to choose

1. Competitor types
2. Forecast models
3. Analysis

Nick

LAD Optimized Prices Revisited (Nick)

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: $\frac{\partial R}{\partial p} = \theta_0 + 2\theta_1 p + \theta_2 p_c$

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

Wanted to account for the target demand:

- Sell remaining inventory evenly: $D_{\text{target}} = \frac{I}{T - t + 1}$
- Solve for price needed: $p_{\text{inv}} = \frac{\theta_0 + \theta_2 p_c - D_{\text{target}}}{-\theta_1}$
- Inventory weight: $\alpha = \text{clip}\left(\frac{I}{I_{\text{total}}}, 0, 1\right)$
- Final Price: $p = \alpha \cdot p_{\text{opt}} + (1 - \alpha) \cdot p_{\text{inv}}$

LAD Optimized Prices (Nick)

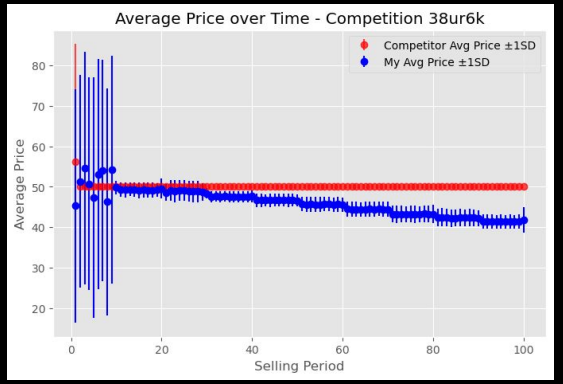
- After experimenting further with RLS, switched to integer linearized LAD to stay with the course.
- Retrained every 10 periods.
- Added heuristics to account for competitor action, exploration, and dealing with unrealistic LAD models.
- Also stored coefficients for analysis.

```
# Residual decomposition
def residual_rule(m, i):
    return y[i] - m.y_hat_int[i] == m.e_pos[i] - m.e_neg[i]
model.residuals = pyo.Constraint(model.I, rule=residual_rule)

# Linearization of rounding
def lower_rule(m, i):
    return sum(X[i, j] * m.m[j] for j in m.J) + m.b - 0.499 <= m.y_hat_int[i]
model.lower = pyo.Constraint(model.I, rule=lower_rule)

def upper_rule(m, i):
    return sum(X[i, j] * m.m[j] for j in m.J) + m.b + 0.5 >= m.y_hat_int[i]
model.upper = pyo.Constraint(model.I, rule=upper_rule)

# Objective: sum of absolute deviations
model.obj = pyo.Objective(expr=sum(model.e_pos[i] + model.e_neg[i] for i in model.I),
                          sense=pyo.minimize)

# Solve with GLPK
solver = pyo.SolverFactory("glpk")
solver.solve(model, tee=False)

coefs = np.array([pyo.value(model.b)] + [pyo.value(model.m[j]) for j in model.J])
return coefs
```

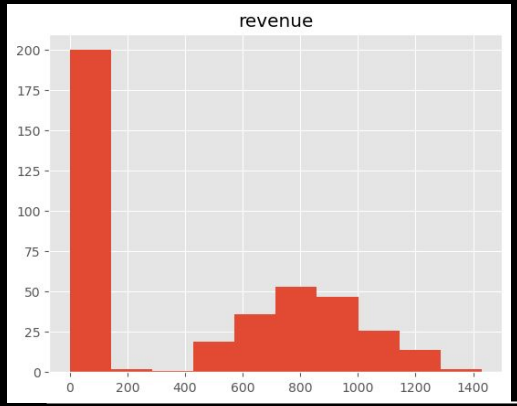
```
# Ensure price isn't too far from competitor last price - consult the demand, price diff plot
MAX_DIFF = 10.0  # max difference from competitor
if competitor_has_capacity_current_period_in_current_season:
    price = np.clip(price, comp_price - MAX_DIFF, comp_price + MAX_DIFF)


# --- Exploratory Heuristics ---
# Random price to get some data
if selling_period_in_current_season < 10:
    price = random.random() * 100

# Randomly (5% of the time, price $5 below the competitor's last price).
if random.random() < 0.05 and competitor_has_capacity_current_period_in_current_season:
    price = comp_price - 5.0

# Monopoly adjustment
if not competitor_has_capacity_current_period_in_current_season:
    price *= MONOPOLY_MARKUP
```

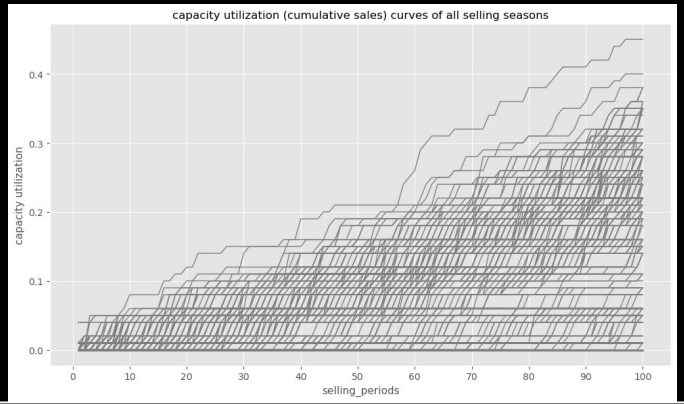
LAD Optimized Prices (Nick)

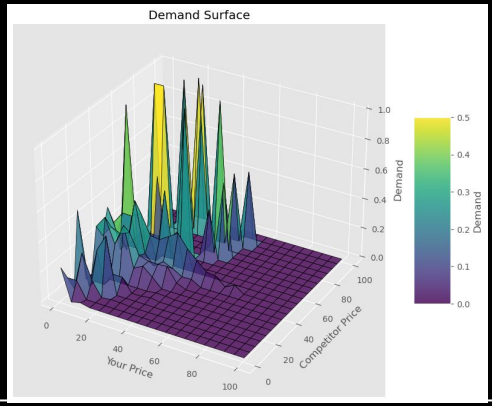
- Results (11-17-25):
  - Best placing round with the LAD model as described.
  - Low demand day (but I placed 8th).
  - My competitor here priced ~$50 the entire time and topped the leaderboard. (I still won against him in this one though)




```
Competition with highest revenue: 38ur6k (Total Revenue: 84969.8)
```



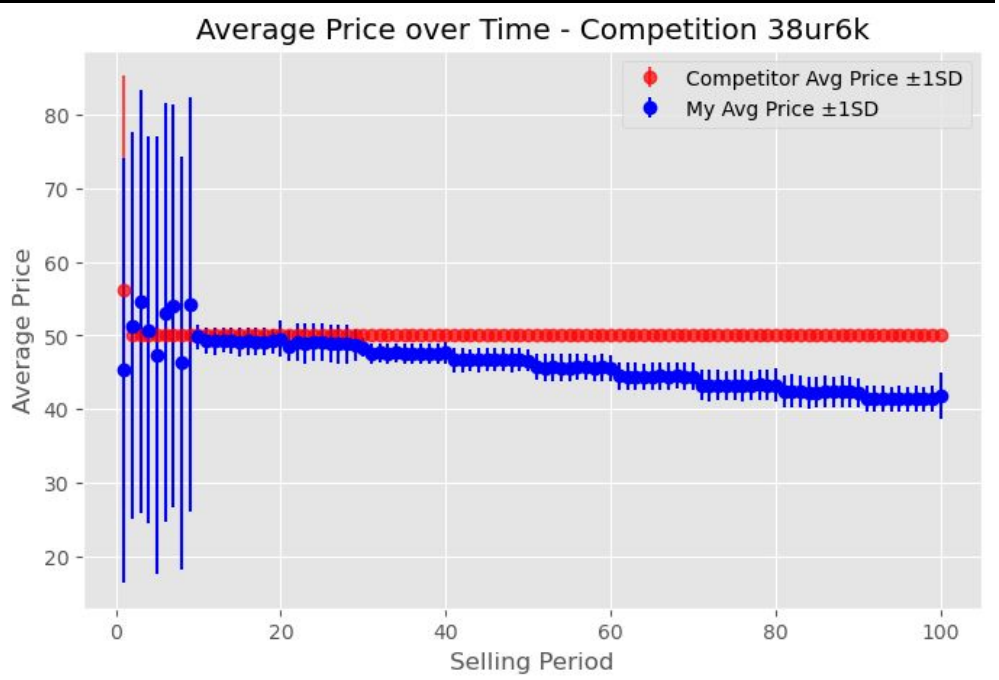


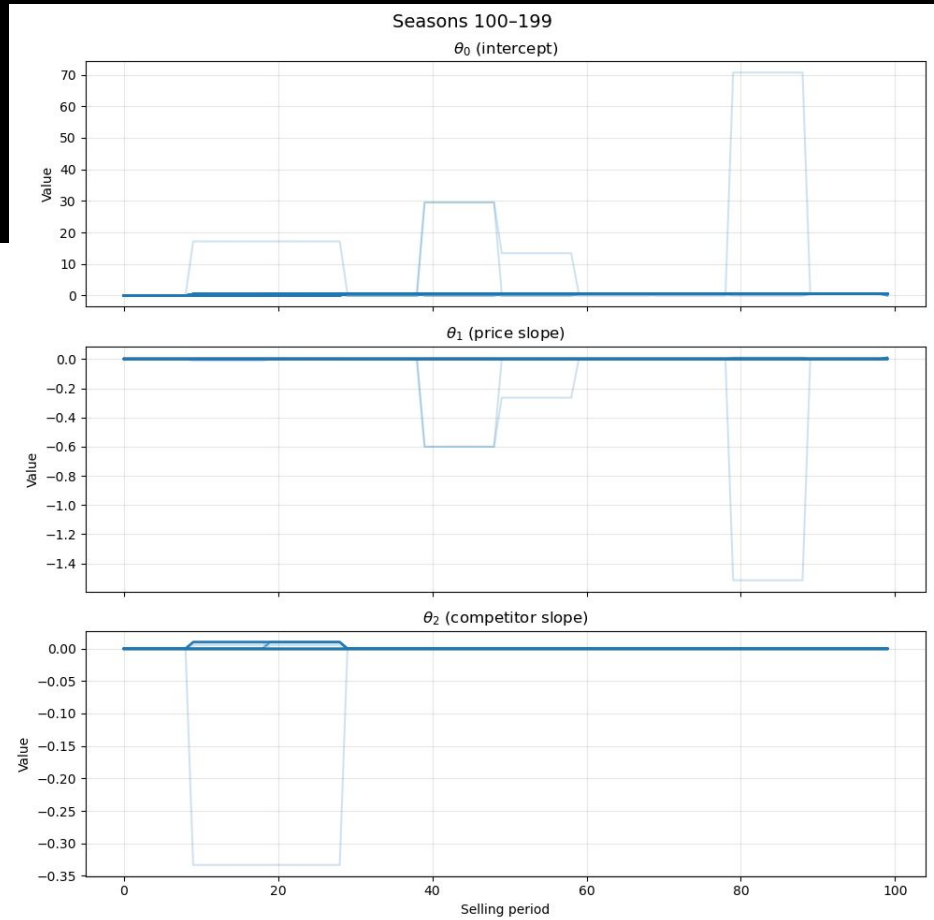


8	IntrepidBeluga	$173,997	$43,499	1.5	$1.00	$100.00	$31.84	0.383	0

LAD Optimized Prices (Nick)

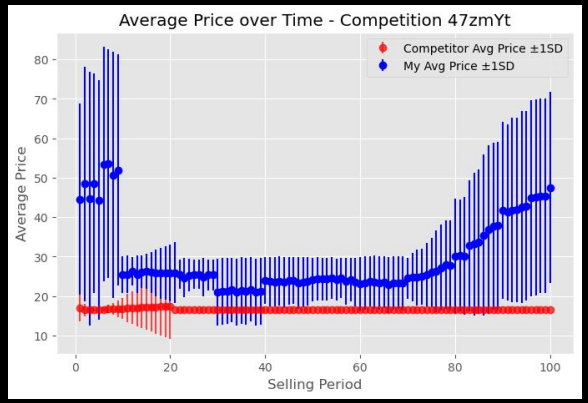
- Results (11-17-25):
  - Examination of LAD coefficients in best run.



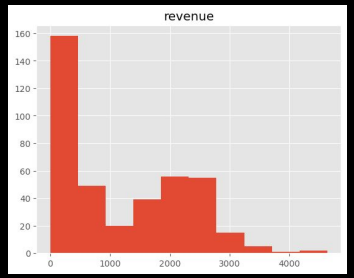


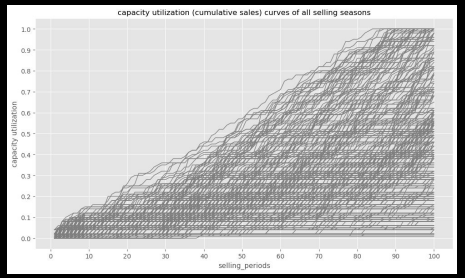
LAD Optimized Prices (Nick)

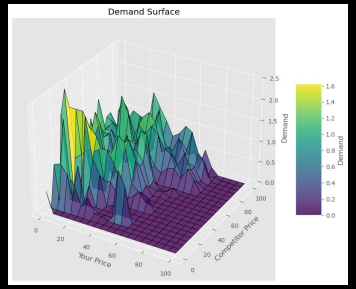
- Results (11-19-25):
  - Had trouble selling enough...
  - One can see the relaxation of the competitor price constraint →
  - The first 10 seasons were random uniform to get data for LAD
  - Was a relatively high demand day.

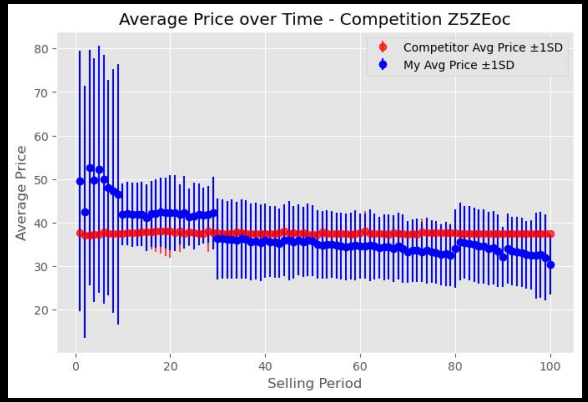



```
Competition with highest revenue: Z5ZEoc (Total Revenue: 211881.6)
```





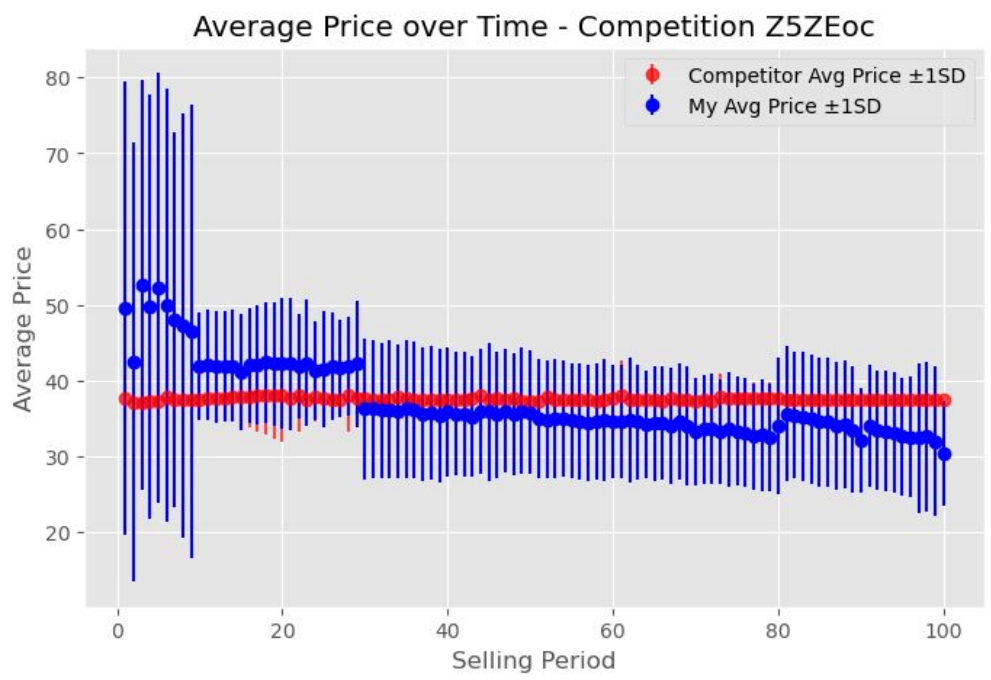


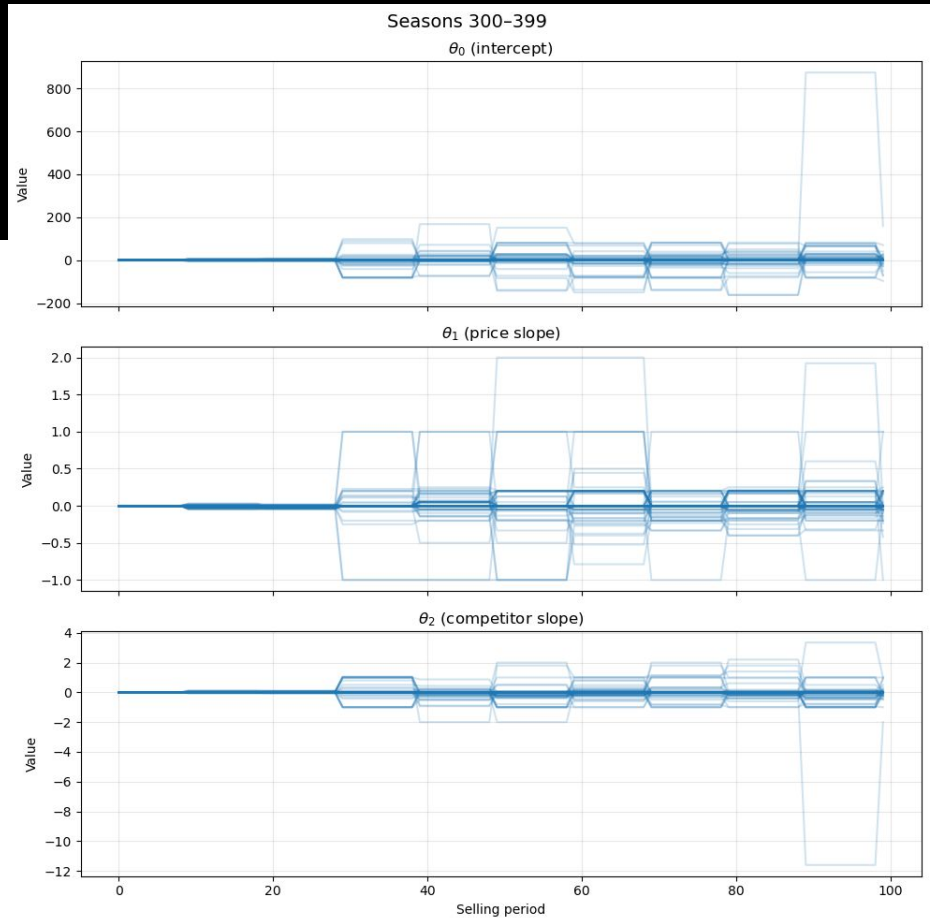


26	IntrepidBeluga	$485,516	$121,379	1.25	$1.00	$100.00	$50.60	0.477	0.087

LAD Optimized Prices (Nick)

- Results (11-19-25):
  - Examination of the LAD coefficients in the best competition.





Dynamic Pricing: A Learning Approach (Nick)

- Using historical prices (and assuming optimal competitor behavior) we implement the following model:

$$\max p_{1,t}\left(\beta_{1,t}^{0}+\beta_{1,t}^{1}p_{1,t}+\beta_{1,t}^{2}\frac{\beta_{2,t}^{0}+\beta_{2,t}^{1}p_{1,t}}{-2\beta_{2,t}^{2}}\right)$$

- But how do we get the competitor coefficients? By minimizing:

$$\sum_{\tau=1}^{t-1}\left|p_{2,\tau}-\left(\frac{\beta_{2,\tau}^{0}+\beta_{2,\tau}^{1}p_{1,\tau}}{-2\beta_{2,\tau}^{2}}\right)\right|$$

Dynamic Pricing: A Learning Approach (Nick)

```python
def build_beta_estimator(p1, p2, max_periods=20):
    """
    Given:
        p1[i] = your price at step i
        p2[i] = competitor price at step i
    Fit betas in:
        p2_pred = (beta0 + beta1 * p1) / (-2 * beta2)
    by minimizing L1 error.
    Returns [beta0, beta1, beta2].
    """

    p1 = p1[-max_periods:]
    p2 = p2[-max_periods:]

    n = len(p1)
    m = pyo.ConcreteModel()
    m.I = pyo.RangeSet(0, n - 1)

    # Decision variables: the betas
    m.beta0 = pyo.Var(domain=pyo.Reals)
    m.beta1 = pyo.Var(domain=pyo.Reals)
    m.beta2 = pyo.Var(domain=pyo.Reals, bounds=(-1000, -1e-6))  # beta2 < 0

    # L1 loss auxiliary variables
    m.z = pyo.Var(m.I, domain=pyo.NonNegativeReals)

    # Absolute deviation constraints
    def abs_lo(m, i):
        pred = (m.beta0 + m.beta1 * p1[i]) / (-2 * m.beta2)
        return m.z[i] >= p2[i] - pred
    m.abs_lo = pyo.Constraint(m.I, rule=abs_lo)

    def abs_hi(m, i):
        pred = (m.beta0 + m.beta1 * p1[i]) / (-2 * m.beta2)
        return m.z[i] >= -(p2[i] - pred)      You, yesterday • Do optimization
    m.abs_hi = pyo.Constraint(m.I, rule=abs_hi)

    # Minimize sum of absolute errors
    m.obj = pyo.Objective(expr=sum(m.z[i] for i in m.I), sense=pyo.minimize)

    # Solve
    solver = pyo.SolverFactory("ipopt")
    solver.solve(m, tee=False)

    # Return coefficients like regression_lad
    coefs = [pyo.value(m.beta0), pyo.value(m.beta1), pyo.value(m.beta2)]
    return coefs
```

```python
# --- Solve integer-linearized LAD for coefficients ---
coefs = regression_lad(X_arr, y_arr)  # returns [b, m1, m2]
theta = torch.tensor([[coefs[0]], [coefs[1]], [coefs[2]]], dtype=torch.float32)

comp_coefs = build_beta_estimator(my_prices, comp_prices)
comp_theta = torch.tensor([[comp_coefs[0]], [comp_coefs[1]], [comp_coefs[2]]], dtype=torch.float32)
```

Estimation of Competitor Coefficients (Inside p)

← Estimation of Competitor Coefficients

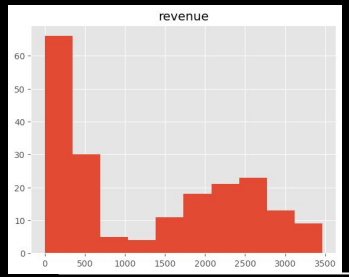
Inventory Management Heuristic

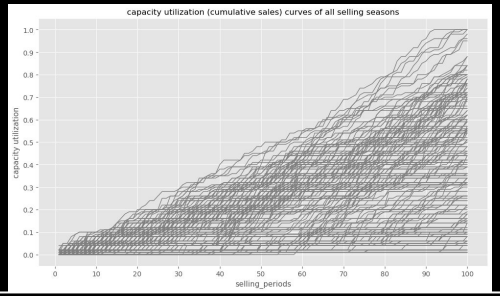
```python
MIN_SALE_FRACTION = 0.05  # never price so high you sell <5% of remaining inventory
MAX_SALE_FRACTION = 0.25  # never price so low you sell >25% remaining inventory
# Estimate expected units to sell
expected_demand = theta0_mine + theta1_mine*price + theta2_mine*comp_price
if expected_demand > remaining_inventory * MAX_SALE_FRACTION:
    price *= 1.05  # slightly increase to slow sales
elif expected_demand < remaining_inventory * MIN_SALE_FRACTION:
    price *= 0.95  # slightly decrease to speed up sales
```

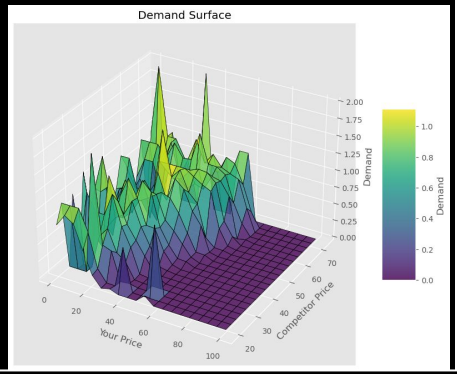
Dynamic Pricing: A Learning Approach (Nick)

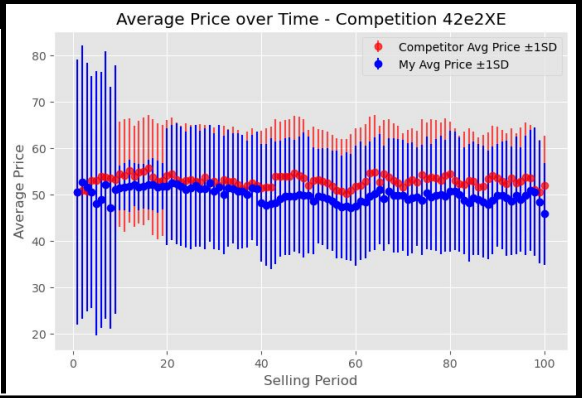
- Results (11-24-25):
  - Most of the revenue was from the below competition.
  - Near perfect performance on bottom, worse on top.
  - No early sell outs, many runs with stock remaining.


```
Competition with highest revenue: 42e2XE (Total Revenue: 232012.4)
```





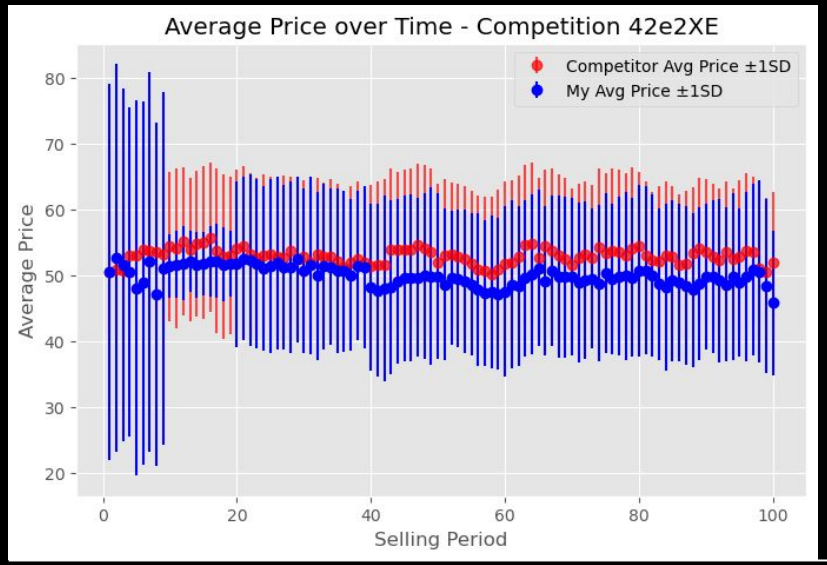


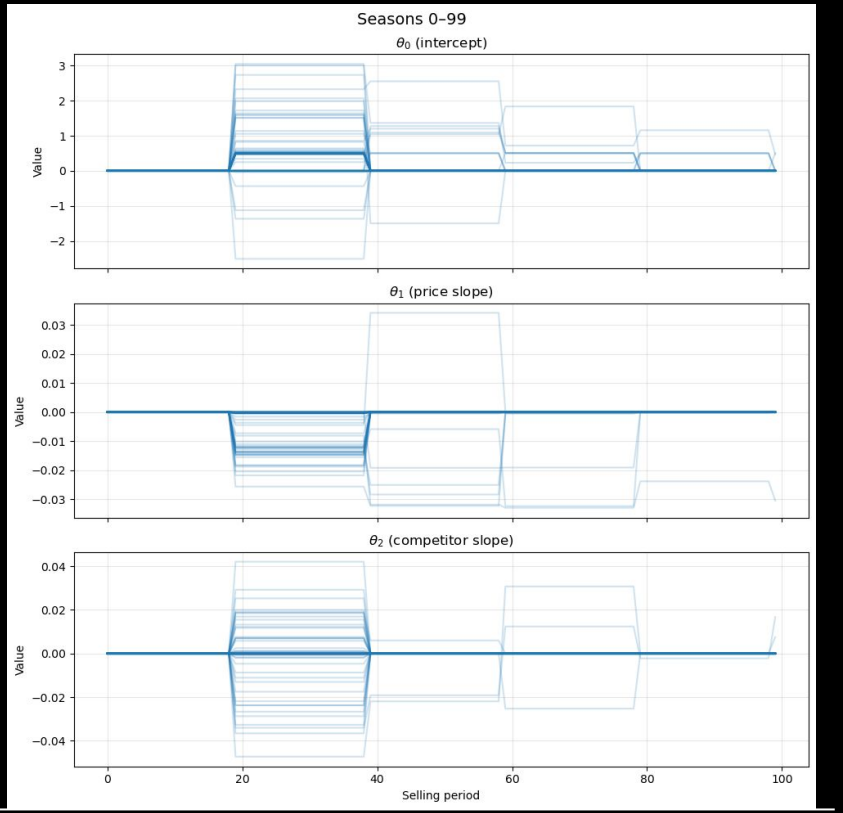


27	IntrepidBeluga	$260,748	$130,374	1.5	$1.00	$100.00	$40.45	0.449	0.015

Dynamic Pricing: A Learning Approach (Nick)

- Results (11-24-25):
  - Still my coefficients
  - Can see good values for theta 0 and theta 1





Forecasting (Nick)

- In the next three approaches, we use the linear demand model and plug in the forecasted competitor's price.
  - First, naive
  - Second, ETS
  - Third, AR(2)

$$\hat{p}^c_{t+1} = p^c_t$$

$$\hat{p}^c_{t+1} = \hat{\beta}_0 + \sum_{i=1}^{L} \hat{\beta}_i \, p^c_{t+1-i}$$

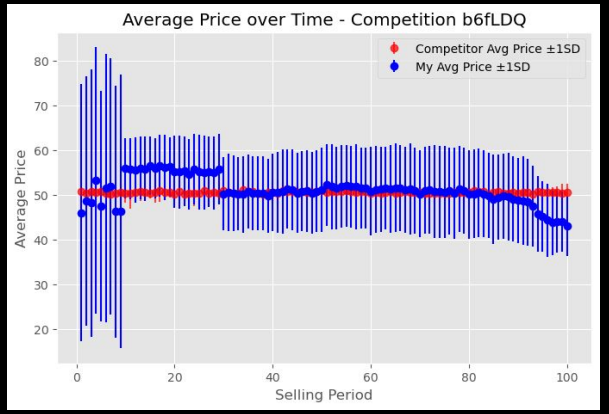
Let $\hat{p}^c_t$ be yesterday's forecast and p^c_t today's observation:

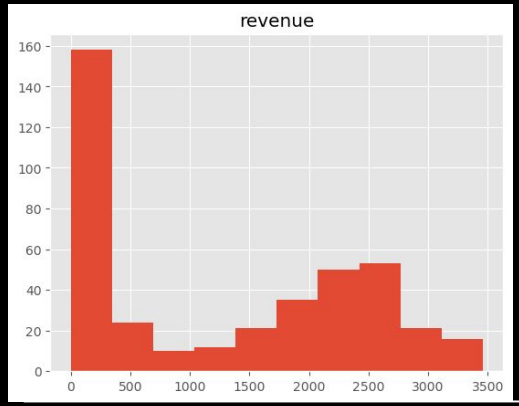
$$\hat{p}^c_{t+1} = (1-\alpha)\hat{p}^c_t + \alpha p^c_t \quad \text{with } \alpha \in (0,1)$$

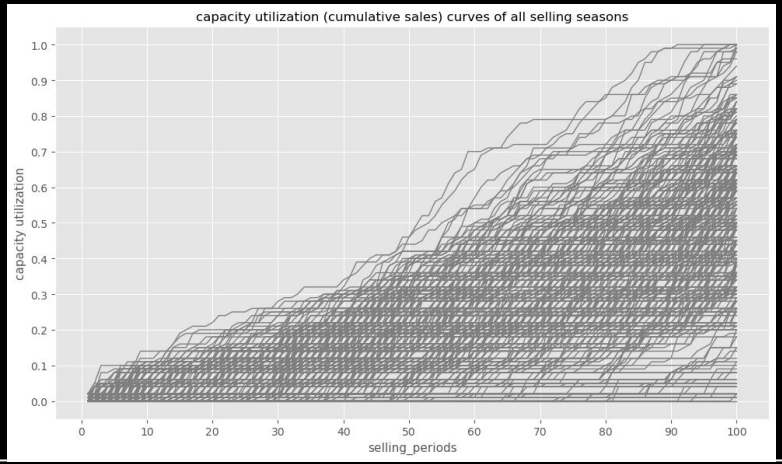
Forecasting (Nick)

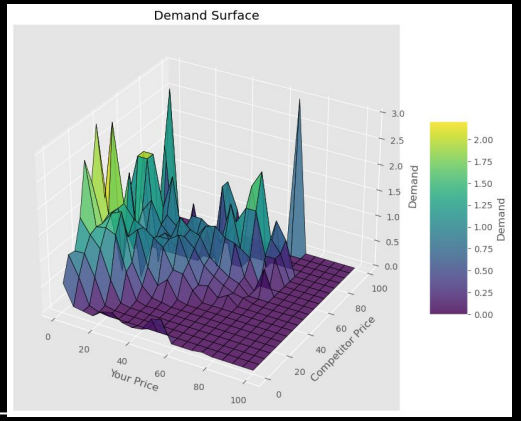
- Results (11-26-25) - Naive Forecast:
  - Decent but still not enough selling!
  - Competitor was ImpartialGoldfish #1 on the leaderboard. (With constant price strategy)

```
Competition with highest revenue: b6fLDQ (Total Revenue: 239430.0)
```





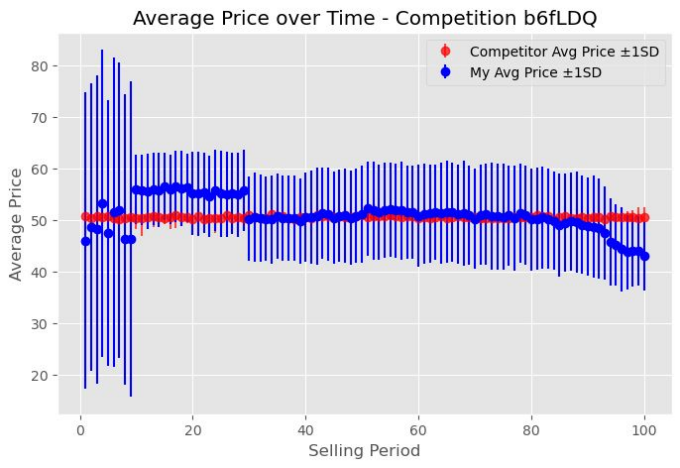


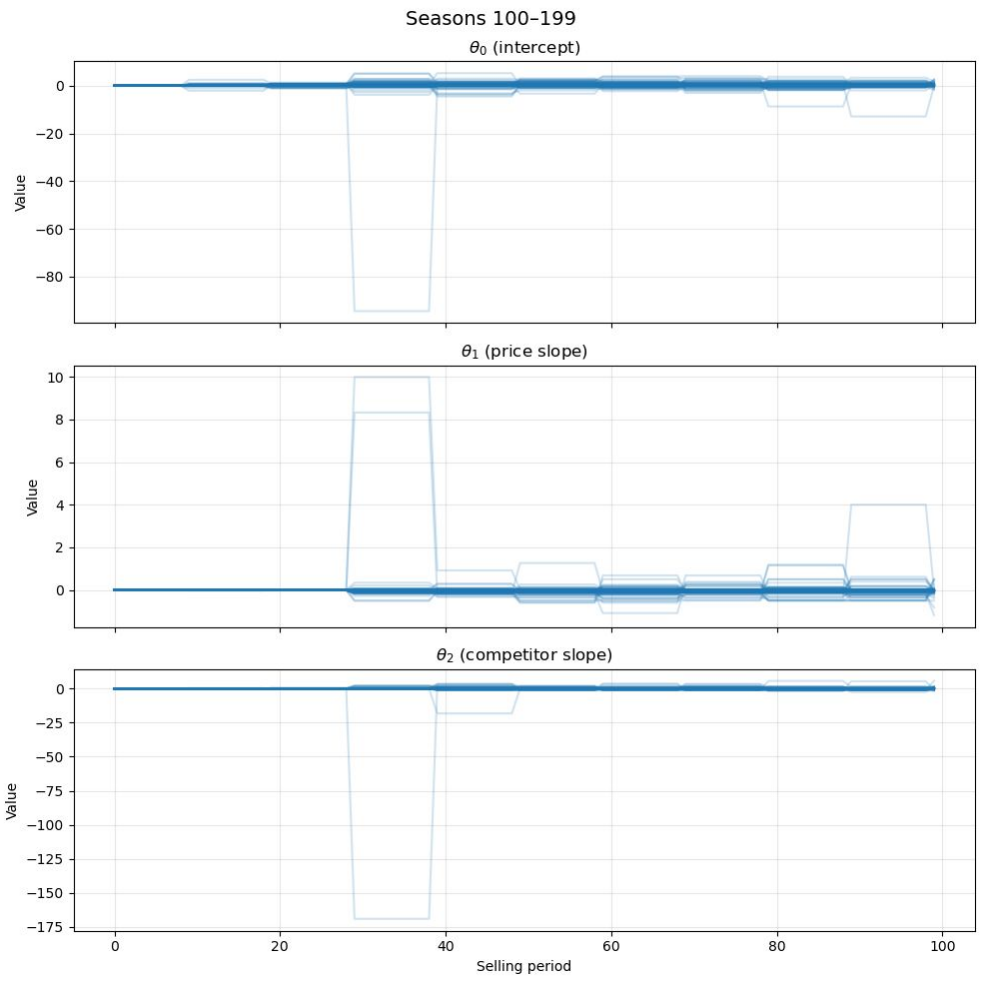


20	IntrepidBeluga	$520,579	$130,145	2	$1.00	$100.00	$34.77	0.486	0.01

Forecasting (Nick)

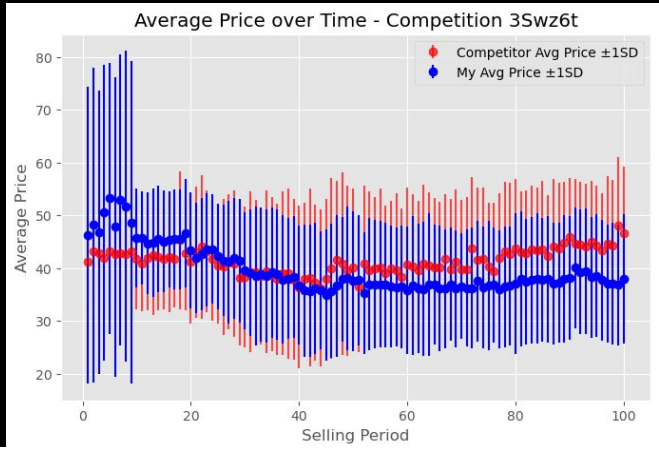
- Results (11-26-25) - Naive Forecast:
  - Note that the values for the thetas here tend close to 0. This wasn't the case for poorer performing competitions.



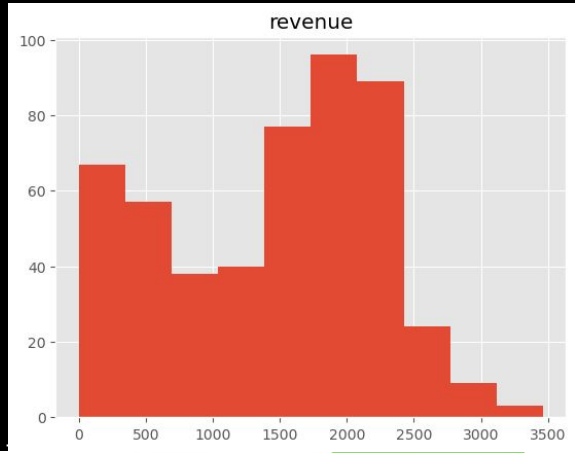


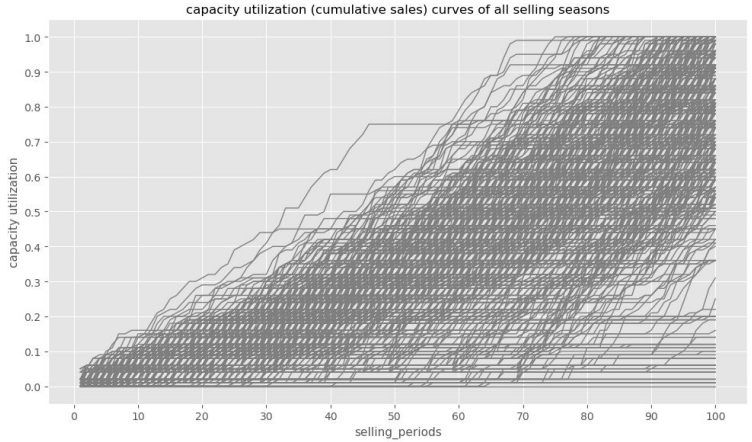
Forecasting (Nick)

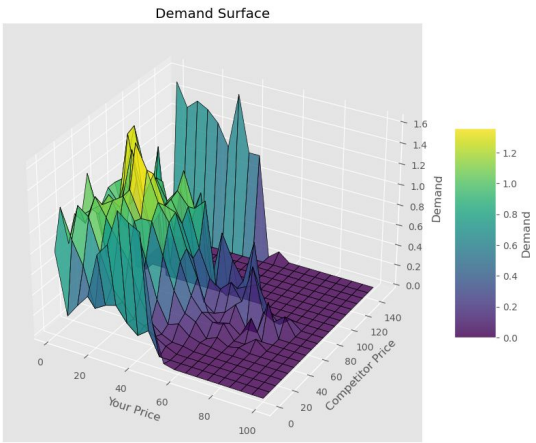
- Results (11-29-25) - Smoothing Forecast:
  - Fewer catastrophic failures
  - Overall best demand model + forecasting run




```
Competition with highest revenue: 3Swz6t (Total Revenue: 204612.4)
```



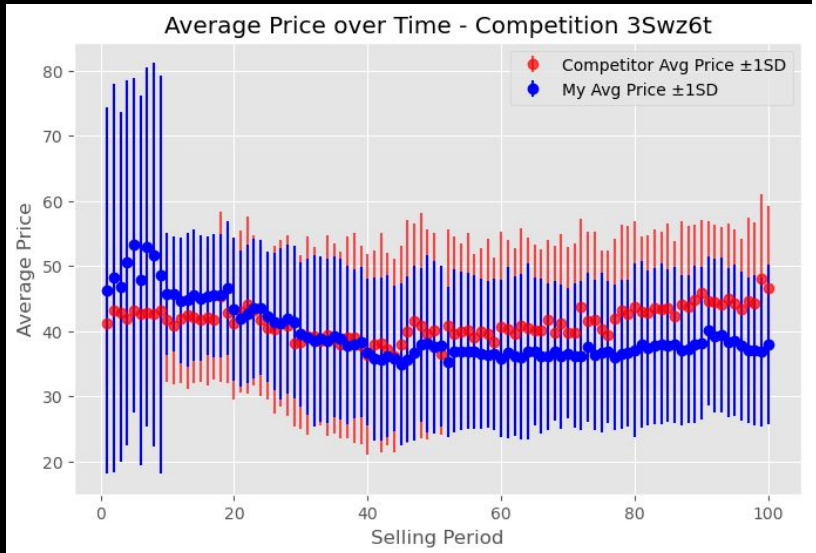


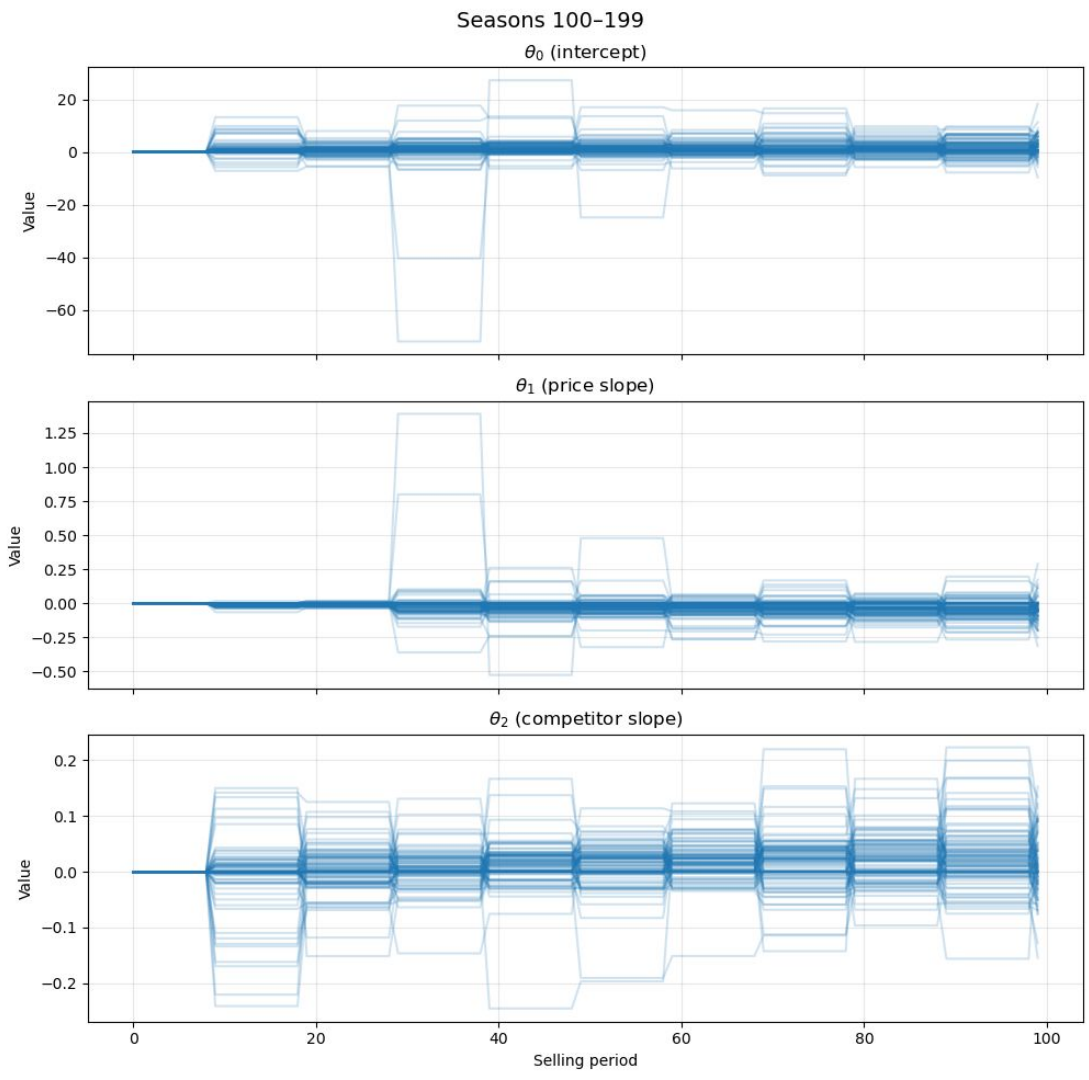


27	IntrepidBeluga	$723,422	$144,684	1.4	$1.00	$100.00	$30.88	0.552	0.192

Forecasting (Nick)

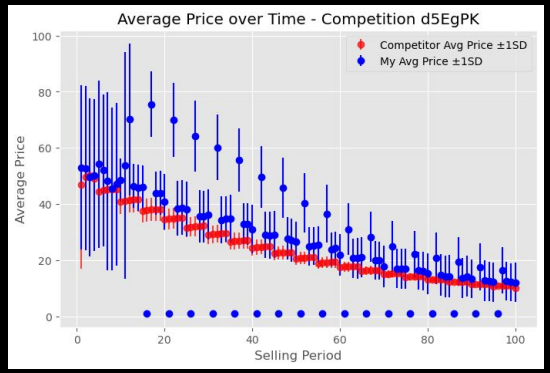
- Results (11-29-25) - Smoothing Forecast:
  - The values for thetas are tending closer to 0 than the other competitions.



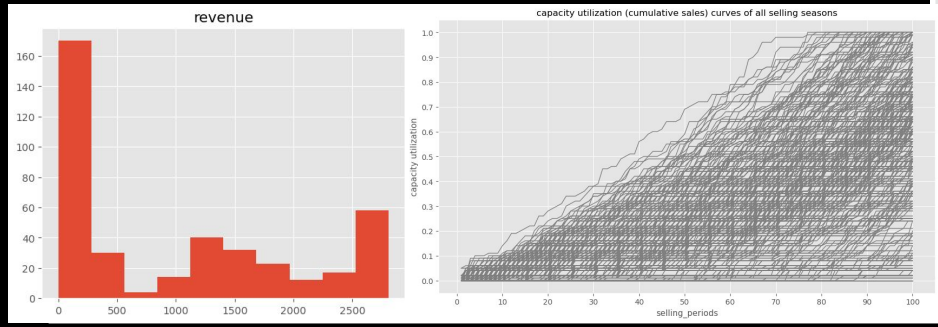


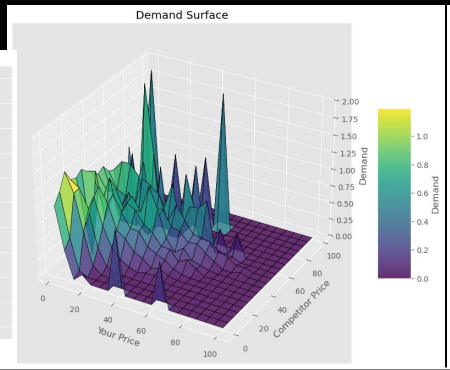
Forecasting (Nick)

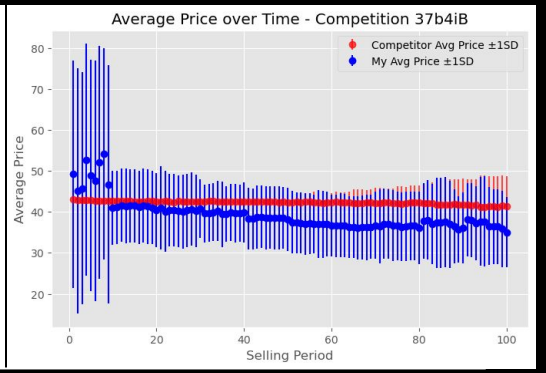
- Results (12-1-25) - AR(2) Forecast:
  - Used a lag of 2, naive forecast when AR failed or there was too little data
  - Here we had strange results (top) and my best run (bottom)




```
Competition with highest revenue: 37b4iB (Total Revenue: 239378.7)
```



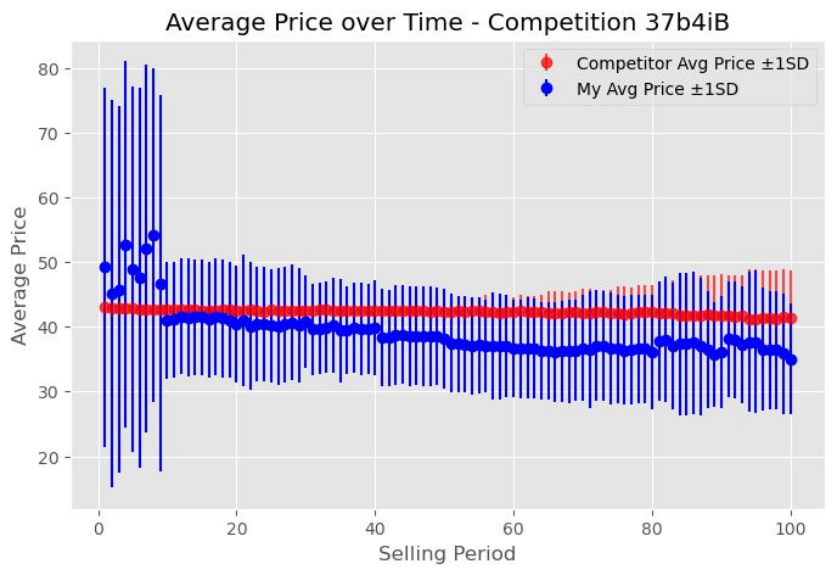


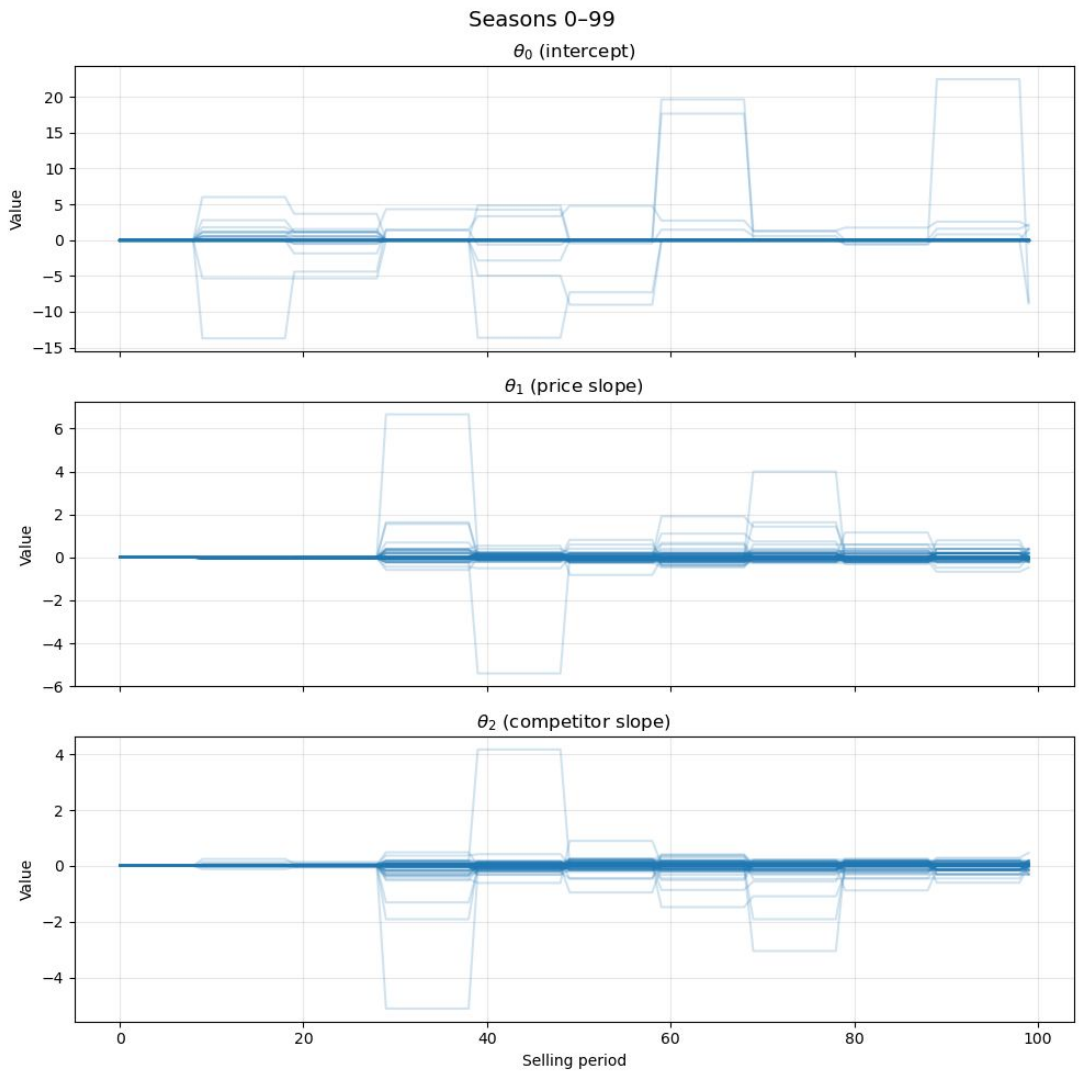


25	IntrepidBeluga	$410,441	$102,610	1.75	$1.00	$100.00	$31.25	0.247	0.113

Forecasting (Nick)

- Results (12-1-25) - AR(2) Forecast:
  - Here we have many outliers in parameter estimation





Dynamic Programming (Nick)

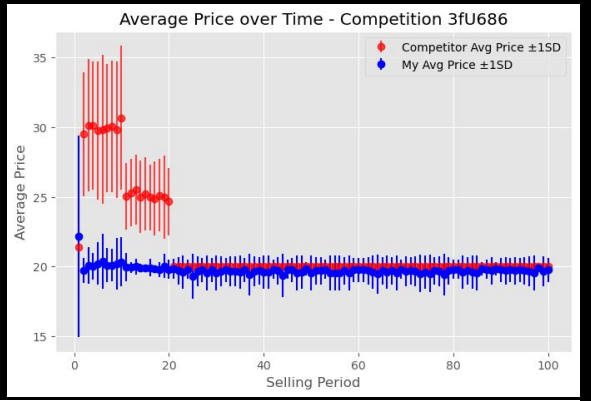
- Used the techniques introduced in class to create the value table.
- Had to optimize the code to run fast enough.

```python
def dp_table(para = [0.64237265, -0.01556792, 0.01219541]):
    T = 101        # timesteps
    C = 81         # capacity levels
    D = 12          # max demand

    # DP table: rows = t, columns = free_cap
    V = np.zeros((T, C))       # Value function
    price_table = np.zeros((T, C, D+1))  # Optional: store prices
    demand_table = np.zeros((T, C, D+1)) # Optional: store demand

    for t in range(1, T):
        for free_cap in range(C):
            max_demand = min(D, free_cap)
            for demand in range(max_demand+1):
                best_price = np.clip(get_best_price_target_demand(demand, para=para), 0.01, 100)

                # DP: remaining capacity
                remaining_cap = free_cap - demand
                best_v = V[t-1, remaining_cap] if t-1 > 0 else 0

                V[t, free_cap] = max(V[t, free_cap], best_v + demand * best_price)

                # optionally store price/demand leading to max V
                price_table[t, free_cap, demand] = best_price
                demand_table[t, free_cap, demand] = demand

    # Convert to DataFrame at the end
    rows = []
    for t in range(T):
        for free_cap in range(C):
            for demand in range(min(D, free_cap)+1):
                V_t = V[t, free_cap]
                best_price = price_table[t, free_cap, demand]
                rows.append([t, free_cap, V_t, best_price, demand])

    df_V_fast = pd.DataFrame(rows, columns=['t','free_cap','V_t','price_t','demand_t'])
    return df_V_fast
```

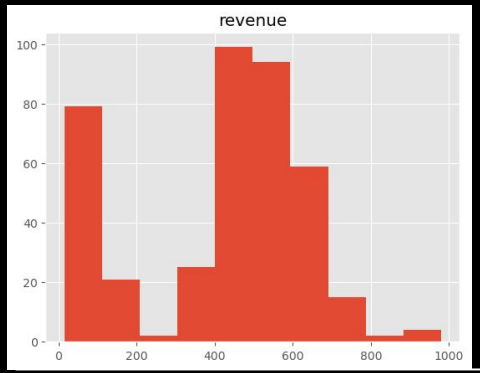
$$V_t(x, d) = V_{t-1}(x - d) + d * price(d)$$

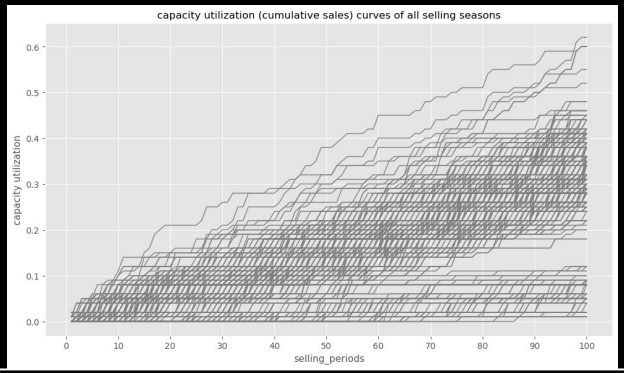
Dynamic Programming (Nick)

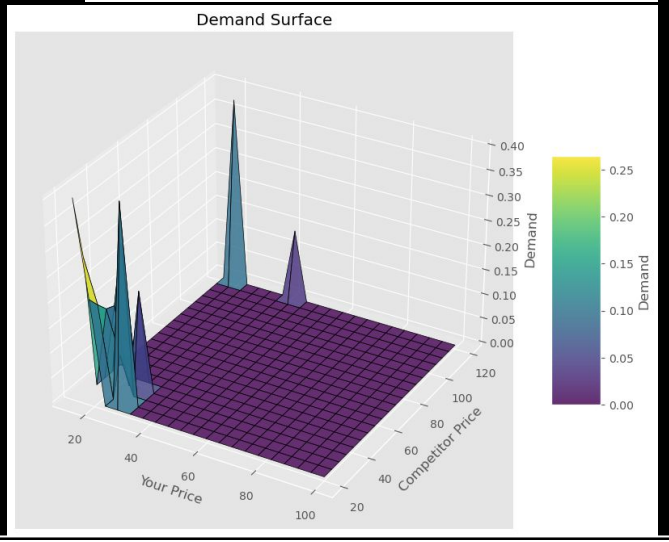
- Results (12-06-2025):
  - Used static demand model params (estimated from 23 days)
  - Maybe I should use a couple DP tables.




```
Competition with highest revenue: 3fU686 (Total Revenue: 55563.299999999996)
```



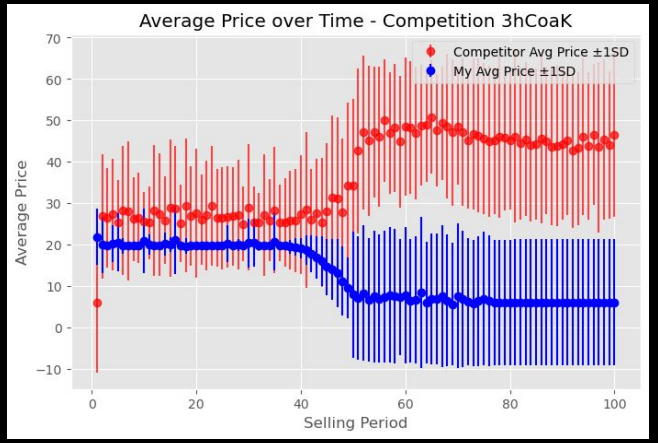




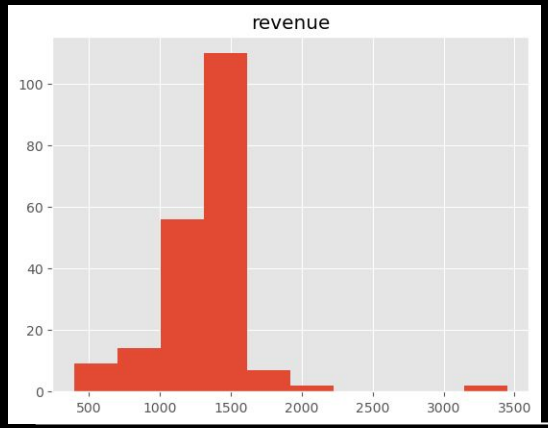
16	IntrepidBeluga	$167,672	$41,918	1	$15.00	$100.00	$19.74	0.008	0

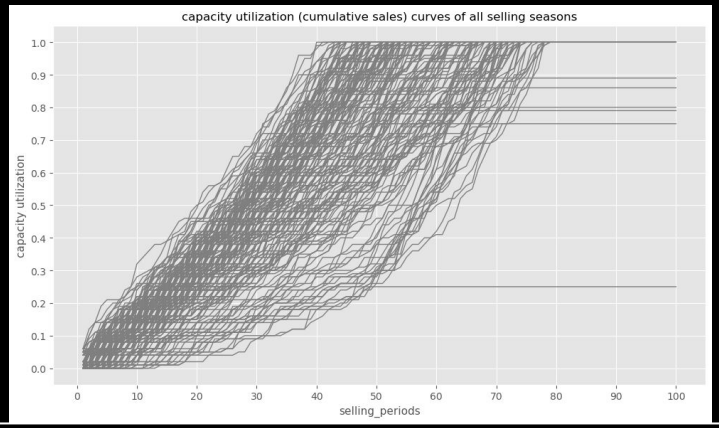
Dynamic Programming (Nick)

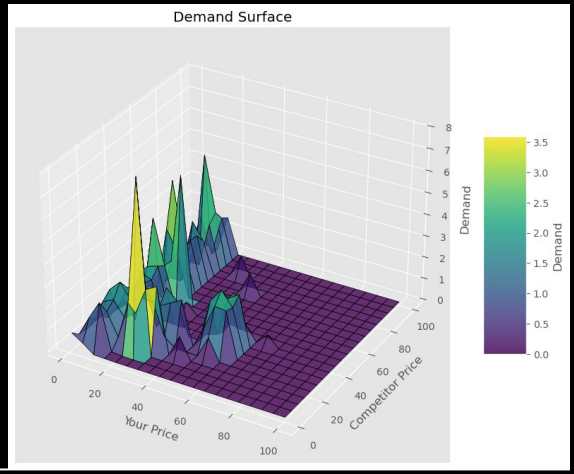
- Results (12-08-2025):
  - Used the overall parameters in the first 50 periods and estimated parameters for the second 50.
  - High standard deviation of price in the second half




```
Competition with highest revenue: 3hCoaK (Total Revenue: 147320.69999999998)
```

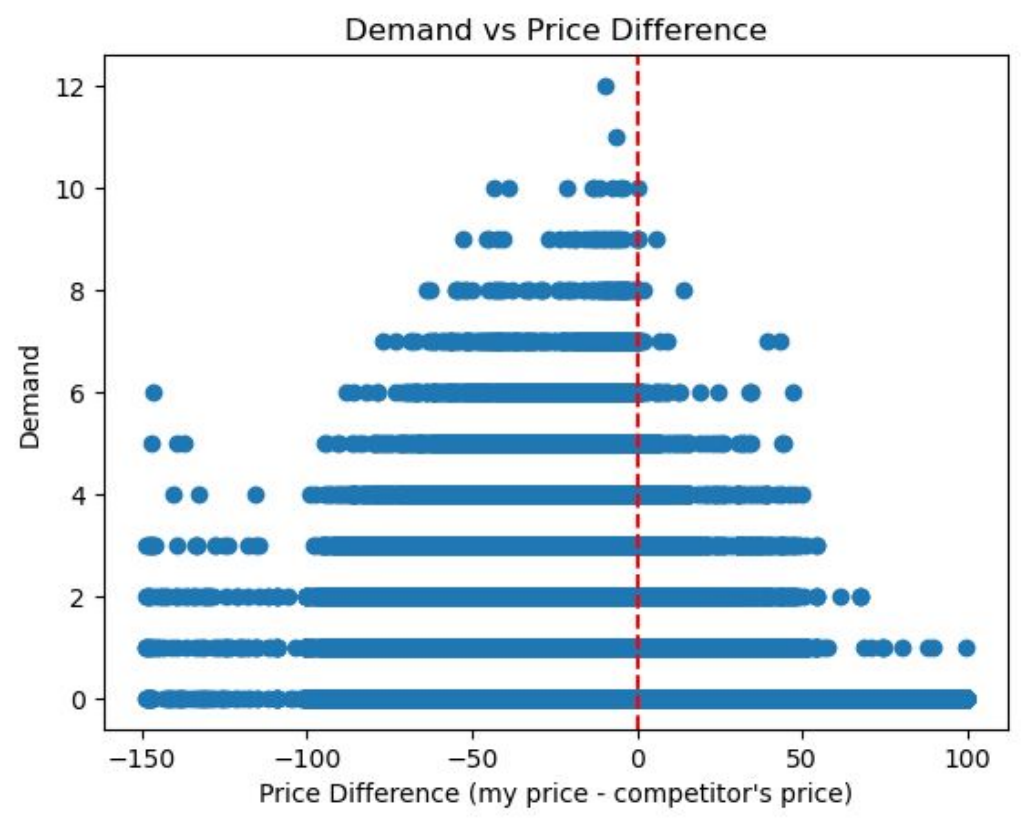






44	IntrepidBeluga	$268,055	$134,027	2	$1.00	$100.00	$19.91	0.763	0.97

Some overall statistics + visualization




```
Mean demand: 0.48915154639175257 Variance: 0.8504440125296269
```

Reflections & Next Steps

- It seems that small values of theta in the OLS models tend to work better.
    - Perhaps add regularization.
- We seem to not be stocking out enough (i.e. losing profits!). This contrasts the last competition's performance
    - We need to focus more on underpricing the competitor (i.e. better adversarial behavior).
- Computational feasibility is becoming an issue (particularly with Dynamic Programming)
    - Ensure that the models developed are actually feasible within the time, explore fast approaches
- Maybe better demand models would yield better performance

Raphael

Which Forecast Model is the best?

- Accuracy
- Computation Time
- Parameter Settings

Competitor Types

- Variance of Prices
- Changes towards my prices

	Low Var	Med Var	High Var
Low Attach			
Med Attach			
High Attach			

Competitor Types

- Variance of Prices
- Changes towards my prices

	Low Var	Med Var	High Var
Low Attach			
Med Attach			
High Attach	Copy Cat	Copy Cat	

Competitor Types

- Variance of Prices
- Changes towards my prices

	Low Var	Med Var	High Var
Low Attach			Random
Med Attach			Random
High Attach	Copy Cat	Copy Cat	

Competitor Types

- Variance of Prices
- Changes towards my prices

	Low Var	Med Var	High Var
Low Attach	Constant		Random
Med Attach			Random
High Attach	Copy Cat	Copy Cat	I am Random

Competitor Types

- Variance of Prices
- Changes towards my prices

	Low Var	Med Var	High Var
Low Attach	Constant		Random
Med Attach			Random
High Attach	Copy Cat	Copy Cat	I am Random

Competitor Types

- Variance of Prices
- Changes towards my prices

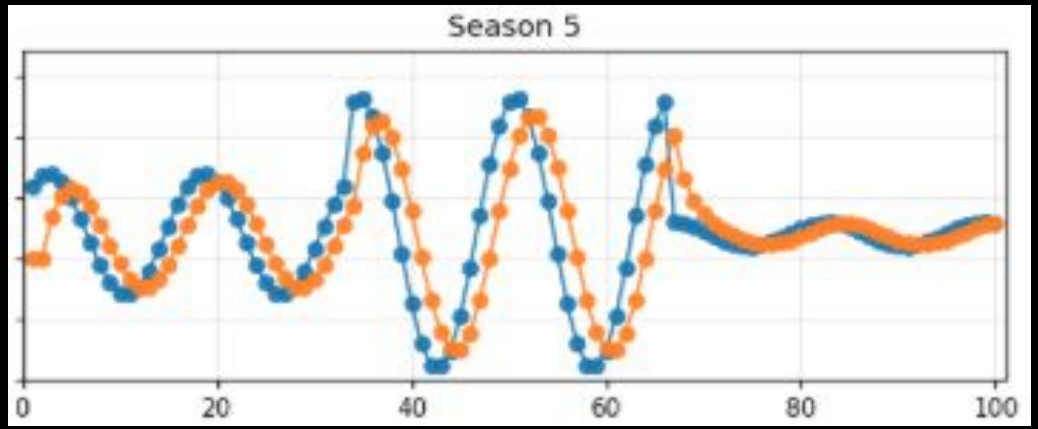
	Low Var	Med Var	High Var
Low Attach	Constant	Price-Adapt	Random
Med Attach	Comp-Adapt	Complex	Random
High Attach	Copy Cat	Copy Cat	I am Random

Competitor Types

- ~~Constant~~
- ~~Random~~
- ~~Copy Cat~~
- Adapt opp
- Adapt price
- Adapt complex
- Unexpected Changes

Competitor Types

- ~~Constant~~
- ~~Random~~
- ~~Copy Cat~~
- Adapt opp
- Adapt price
- Adapt complex
- Unexpected Changes

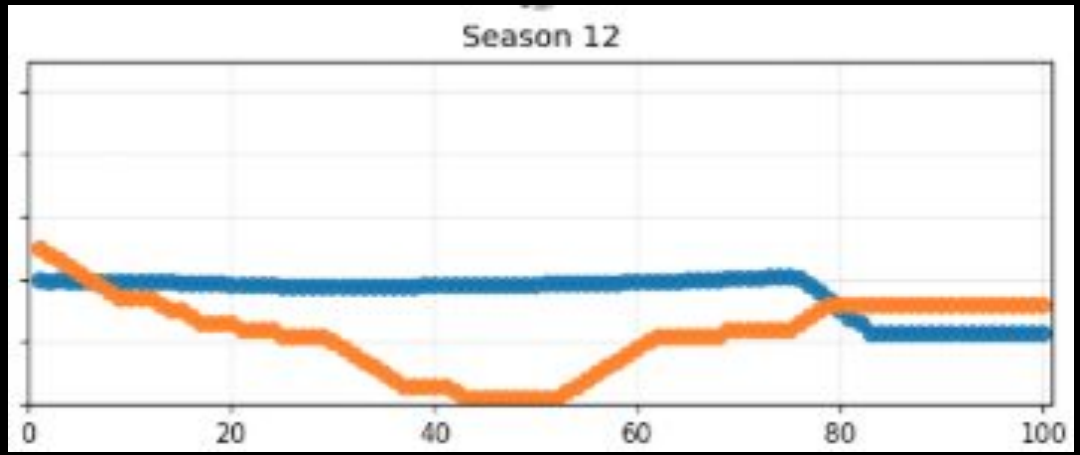


Competitor Types

- ~~Constant~~
- ~~Random~~
- ~~Copy Cat~~
- ~~Adapt opp~~
- Adapt price
- Adapt complex
- Unexpected Changes

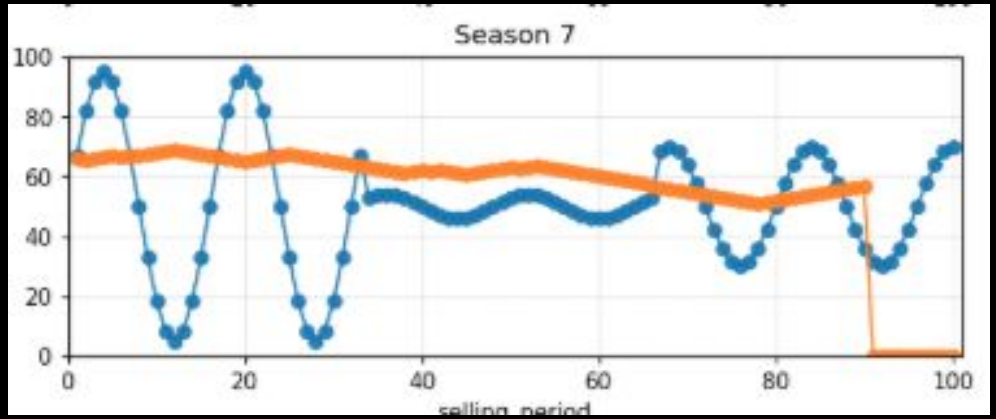
Competitor Types

- ~~Constant~~
- ~~Random~~
- ~~Copy Cat~~
- ~~Adapt opp~~
- Adapt price
- Adapt complex
- Unexpected Changes



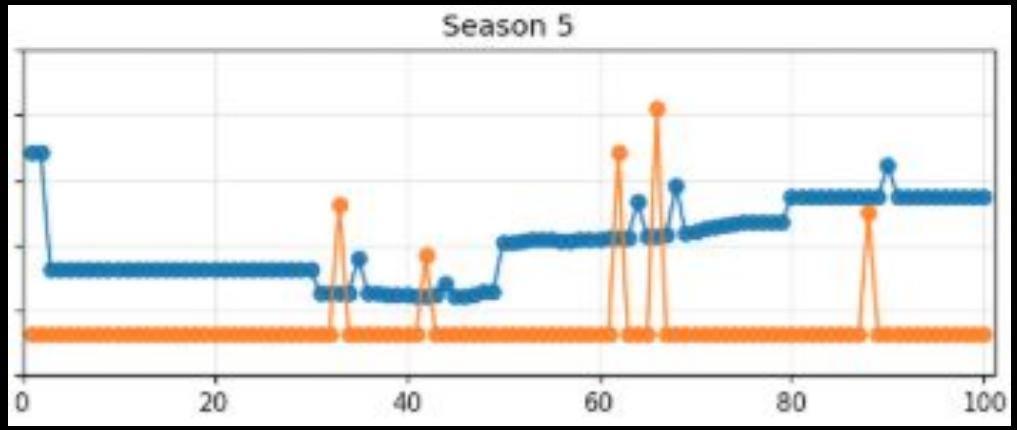
Competitor Types

- ~~Constant~~
- ~~Random~~
- ~~Copy Cat~~
- ~~Adapt opp~~
- Adapt price
- Adapt complex
- Unexpected Changes



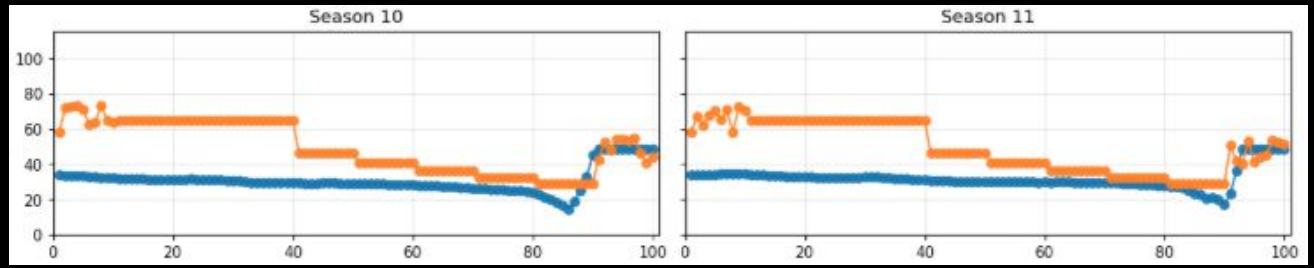
Competitor Types

- ~~Constant~~
- ~~Random~~
- ~~Copy Cat~~
- ~~Adapt opp~~
- Adapt price
- Adapt complex
- Unexpected Changes



Competitor Types

- ~~Constant~~
- ~~Random~~
- ~~Copy Cat~~
- ~~Adapt opp~~
- Adapt price
- Adapt complex
- Unexpected Changes
- Reoccuring pattern



Forecast Models

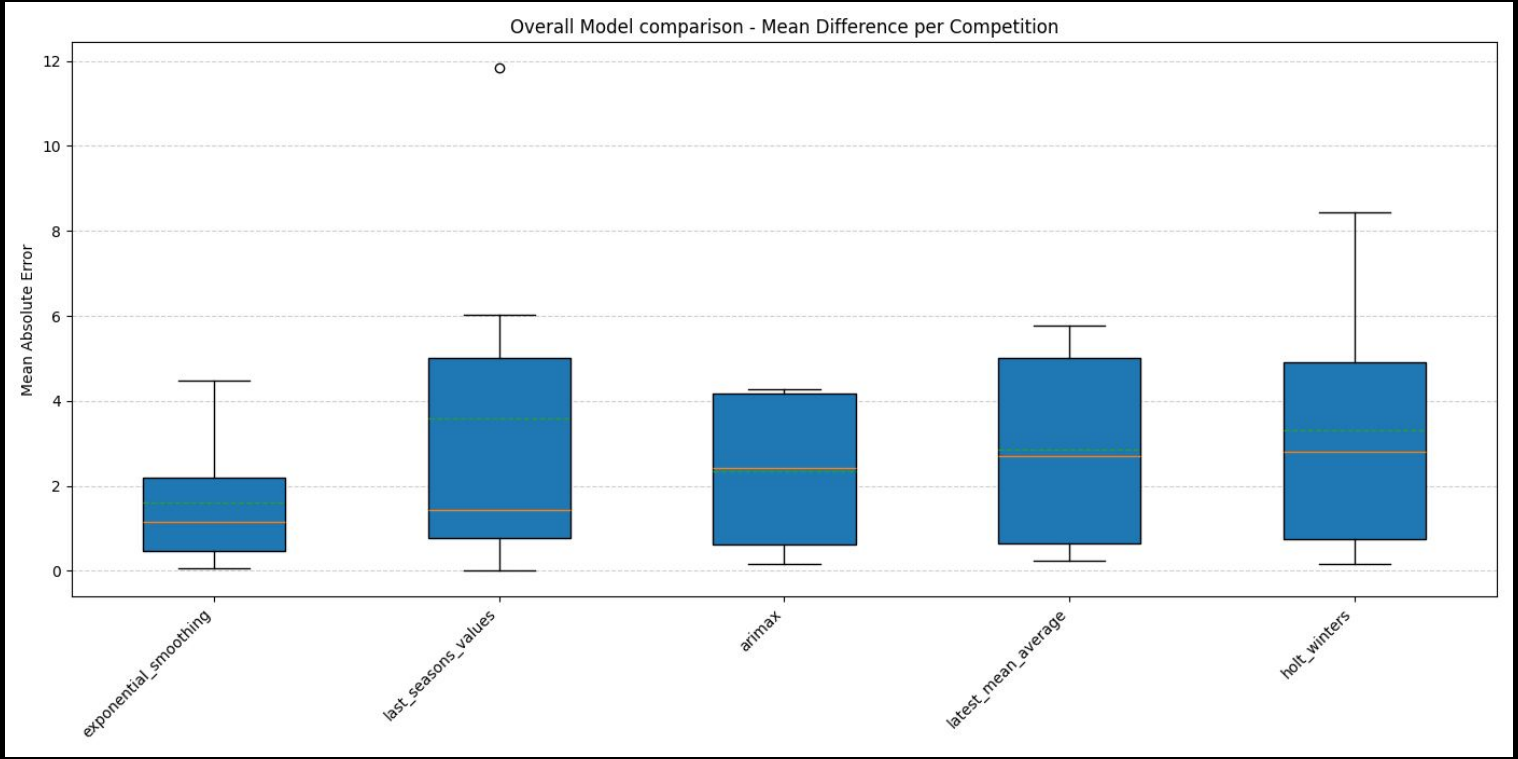
```
from statsforecast.models import AutoETS, AutoARIMA, HoltWinters
HYPERPARAMETERS = {
   "latest_mean_average": {"use_this_many_latest_steps": [1,2, 5, 10, 50]},
   "last_seasons_values":{"sigma": [0.5, 1, 2, 4]},
   "exponential_smoothing": {
       "model": ["ZZZ", "AZZ", "AZN", "MZN", "MZZ"],
       "damped": [True, False]
   },
   "arimax": {
       "p": [0, 1, 2],
       "d": [0, 1, 2],
       "q": [0, 1, 2],
   },
   "holt_winters": {
       "season_length": [12, 25, 100],
       "mode": ["Add", "Multi"],
```

Forecast Models

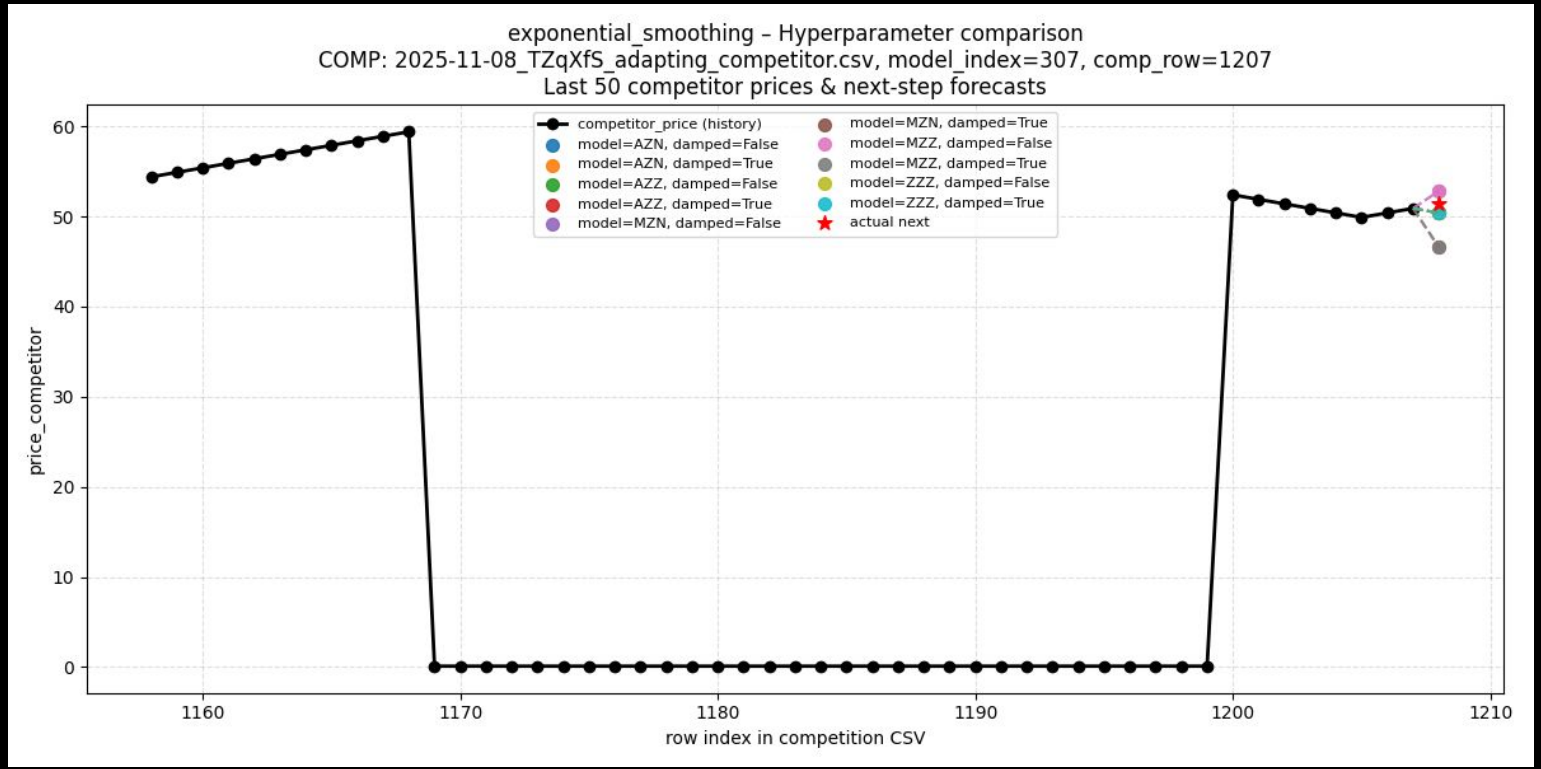
Setup:

- For Each Competition type:
  - Sample 50 random points
  - For Each Model Parameter Setup:
    - Predict next price
    - Calculate MAE

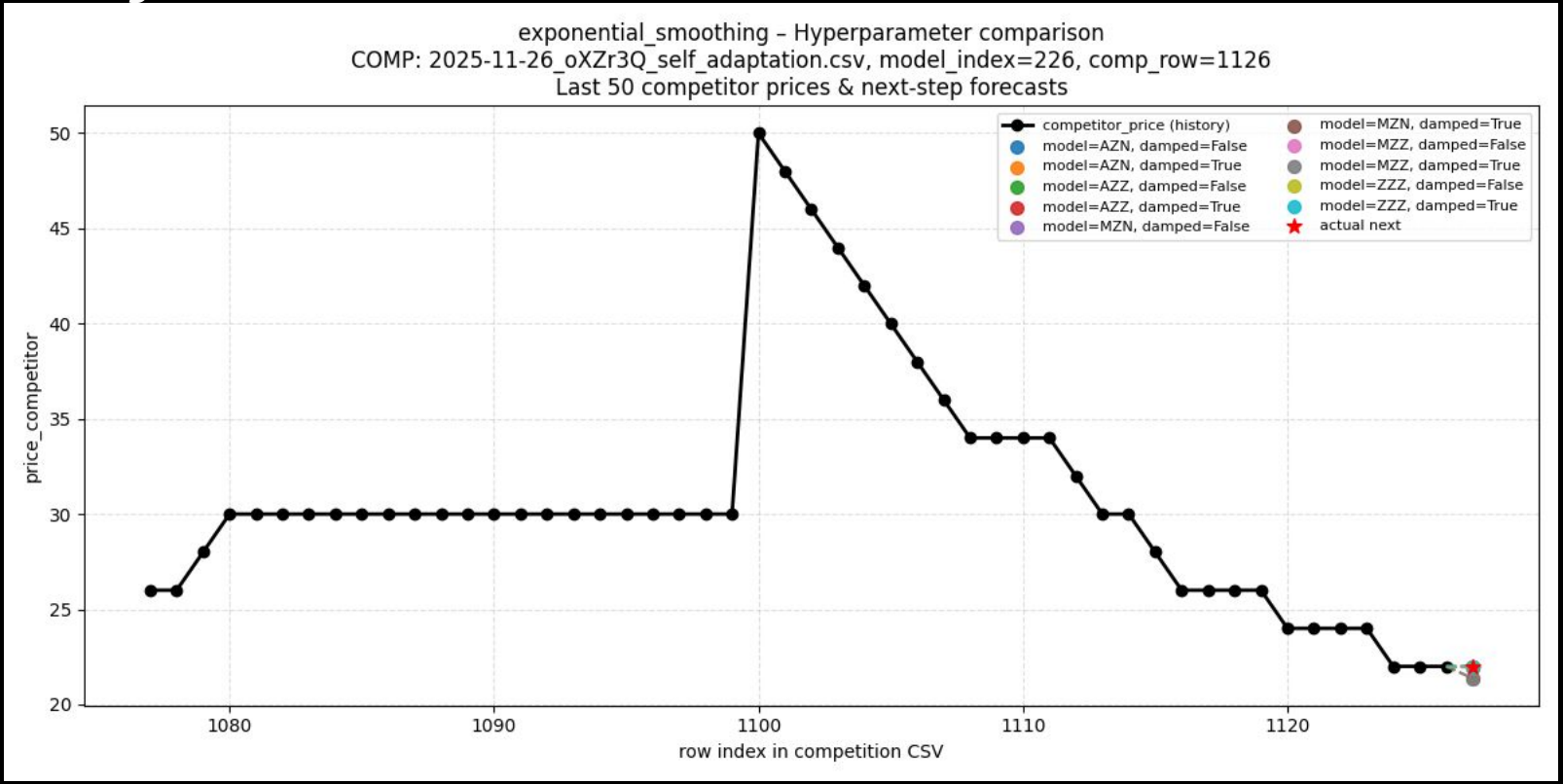
Analysis


Overall Model comparison - Mean Difference per Competition
Mean Absolute Error
0
2
4
6
8
10
12
exponential_smoothing
last_seasons_values
arimax
latest_mean_average
holt_winters

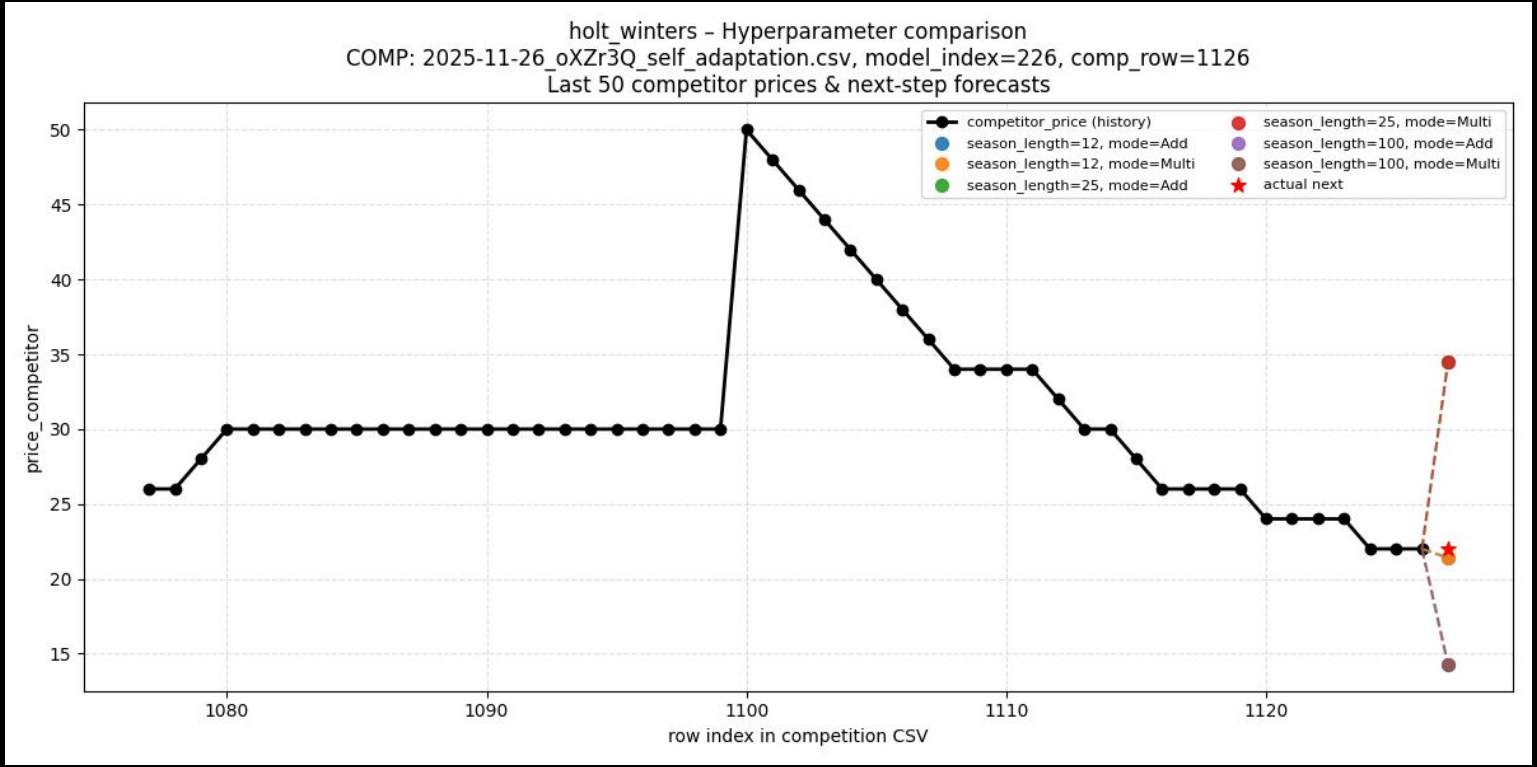
Analysis - Difficult Predictions



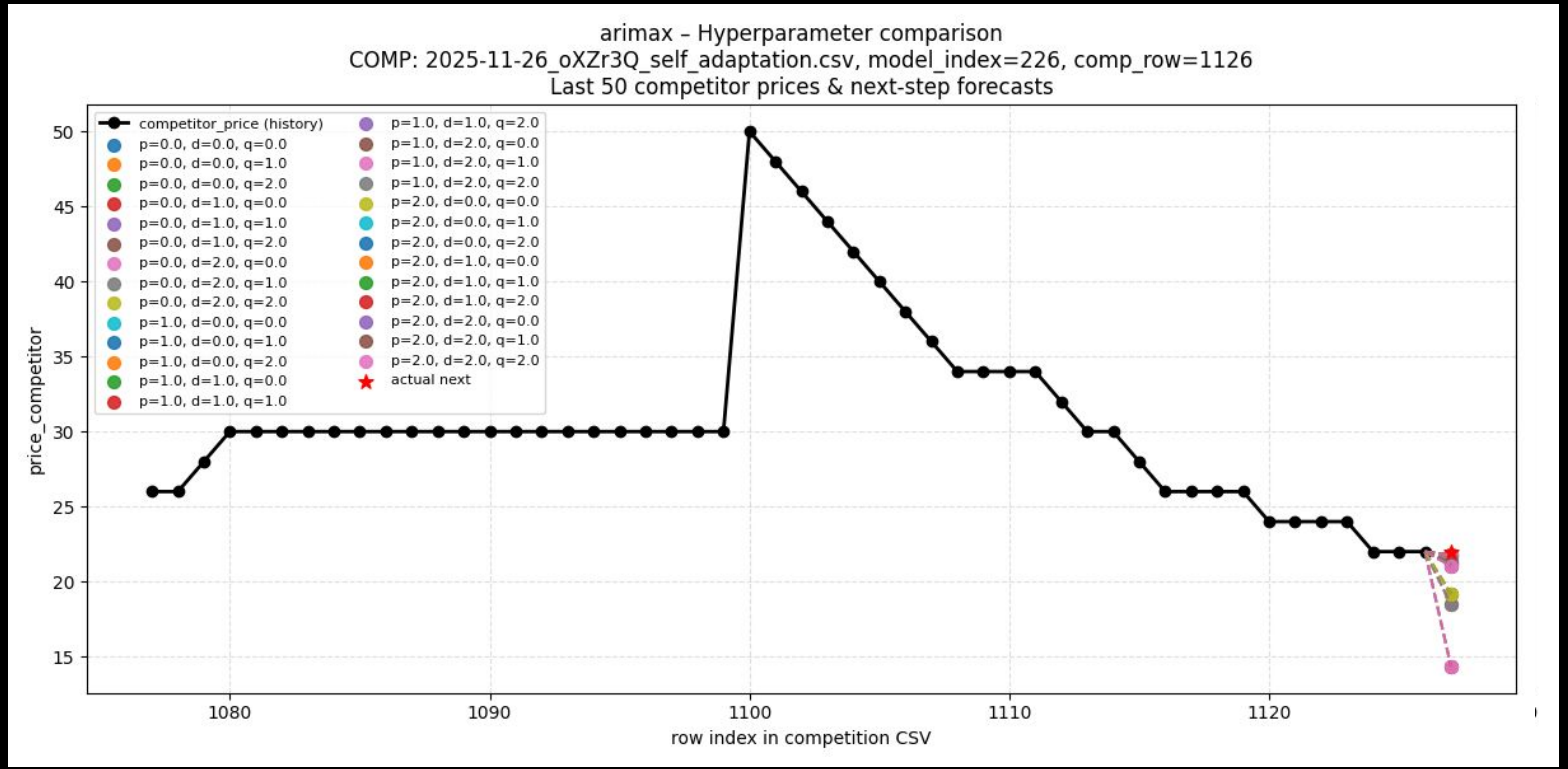
Analysis - Difficult Predictions



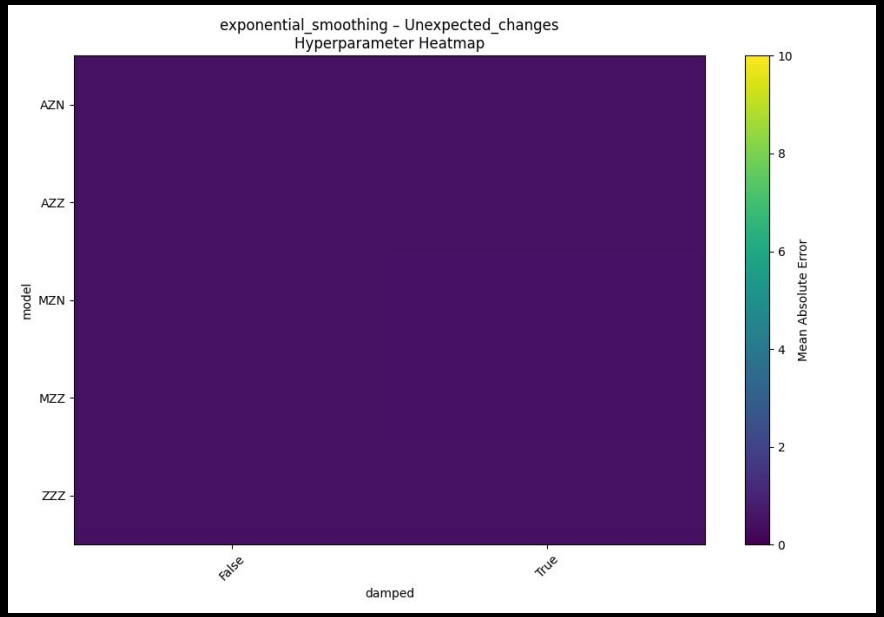
Analysis - Difficult Predictions

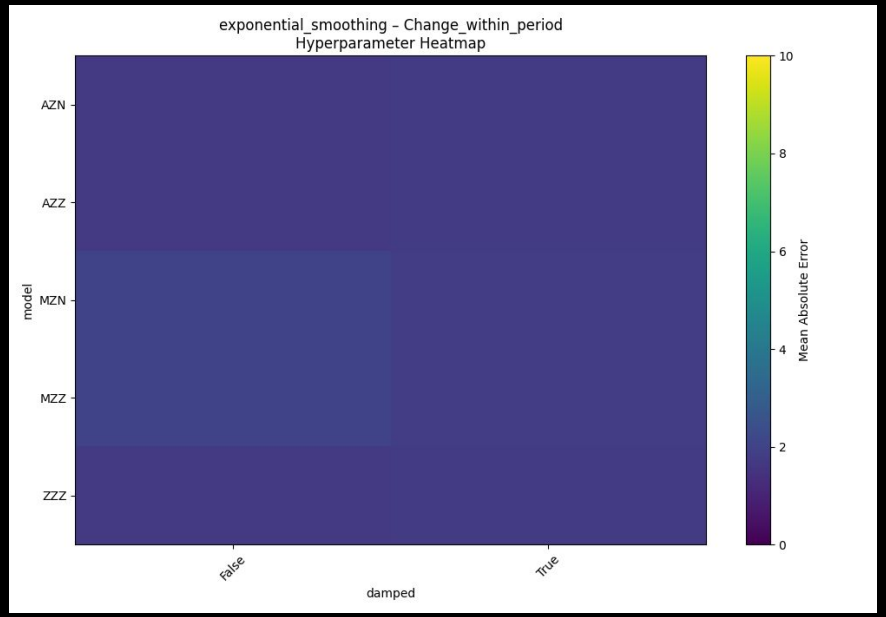


Analysis - Difficult Predictions

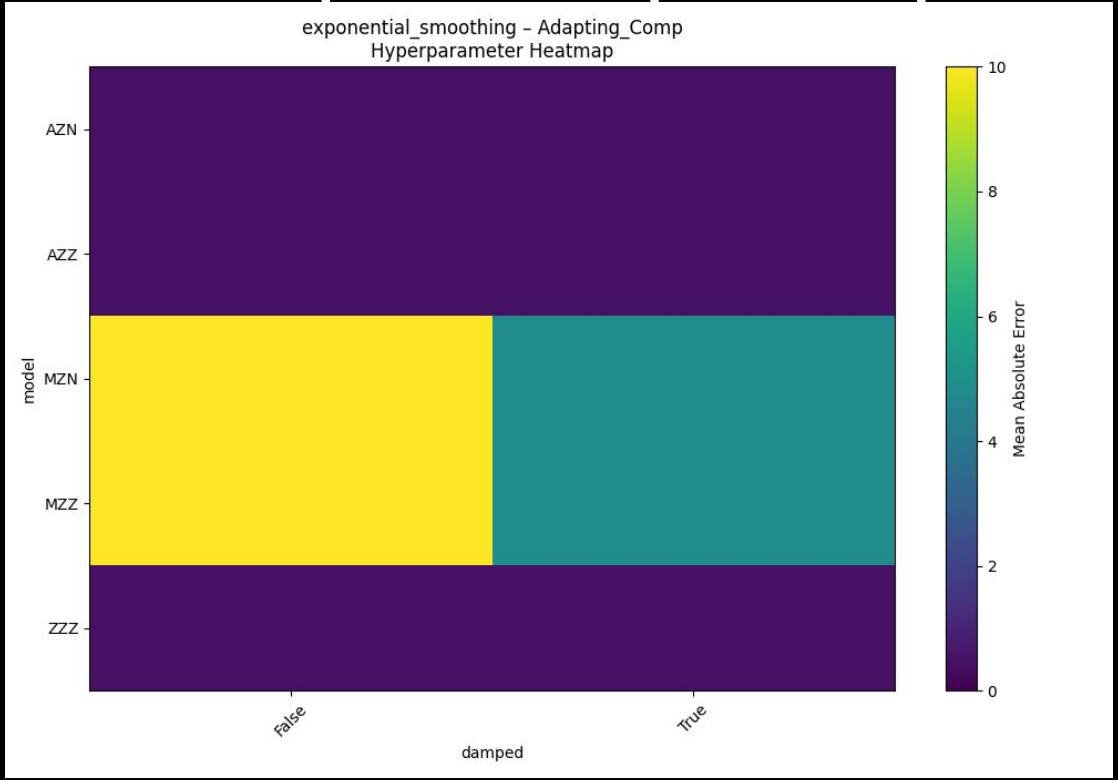


Analysis - Parameters

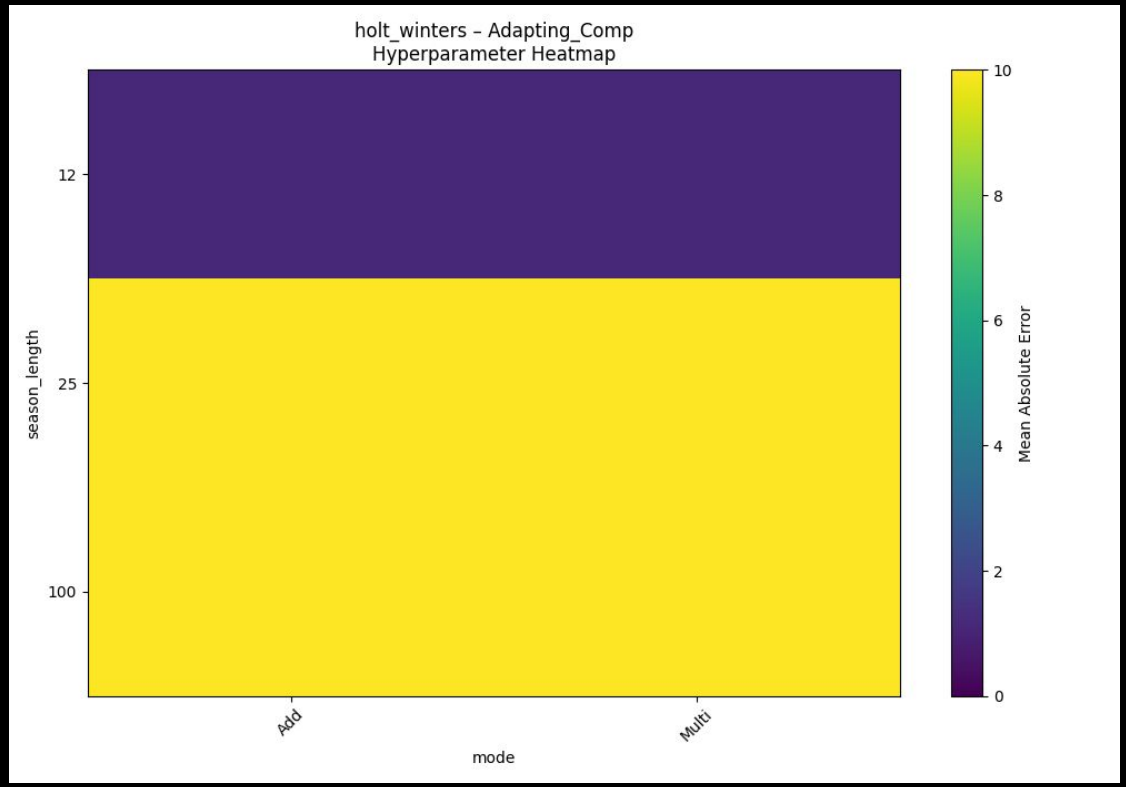




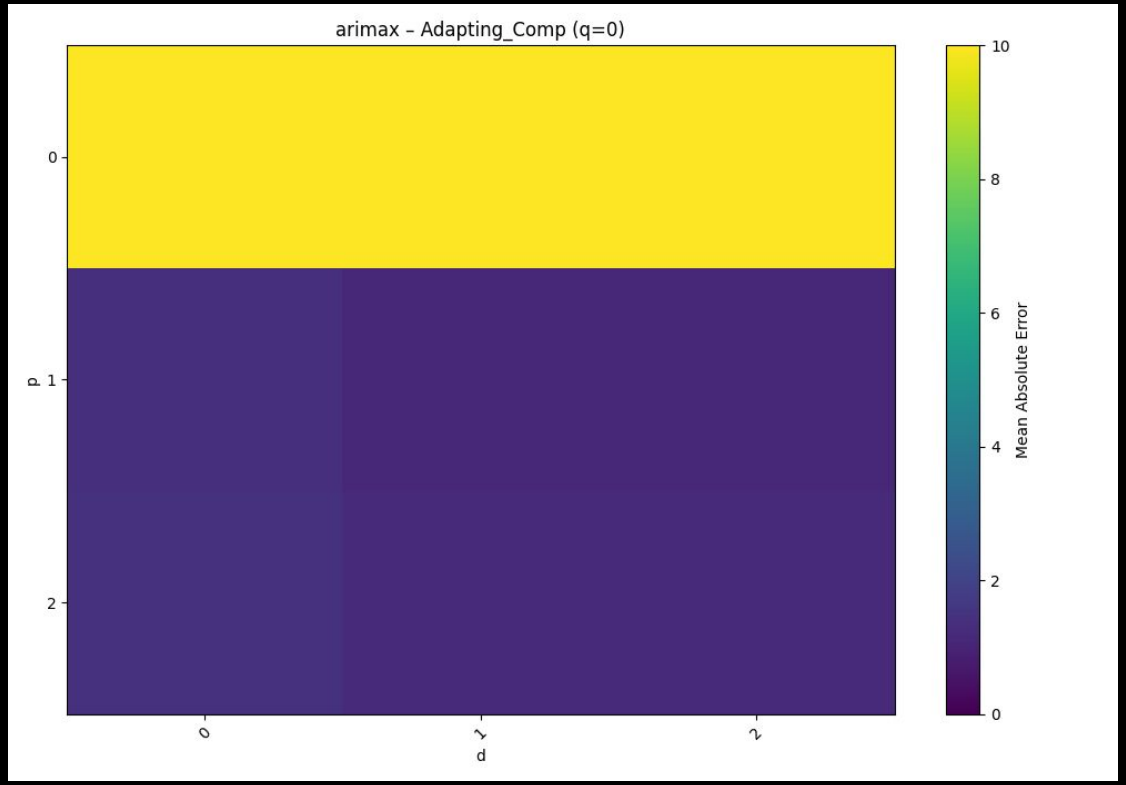
Analysis - Adapt Comp - Expo. Smooth.

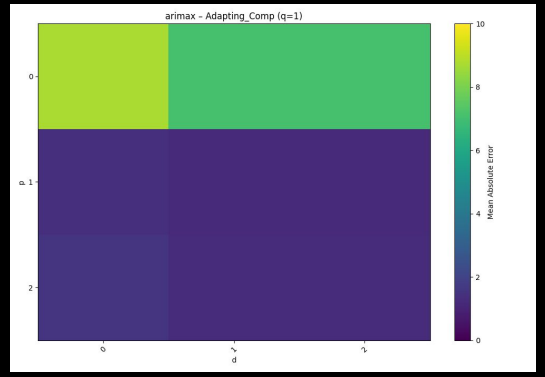


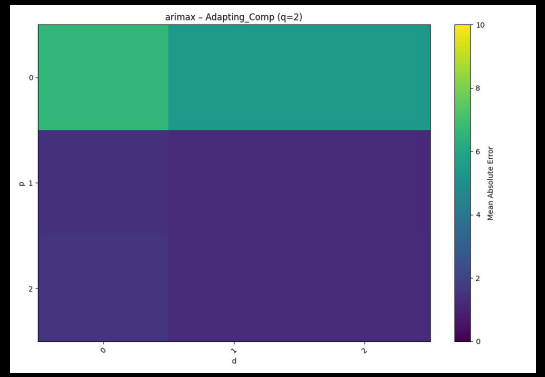
Analysis - Adapt Comp - Holt-Winter



Analysis - Adapt Comp - Arimax

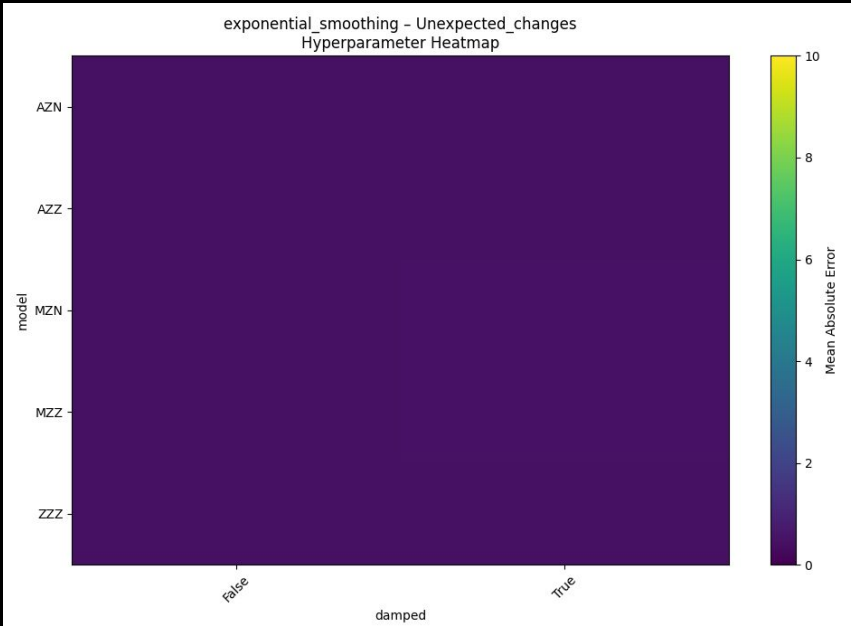




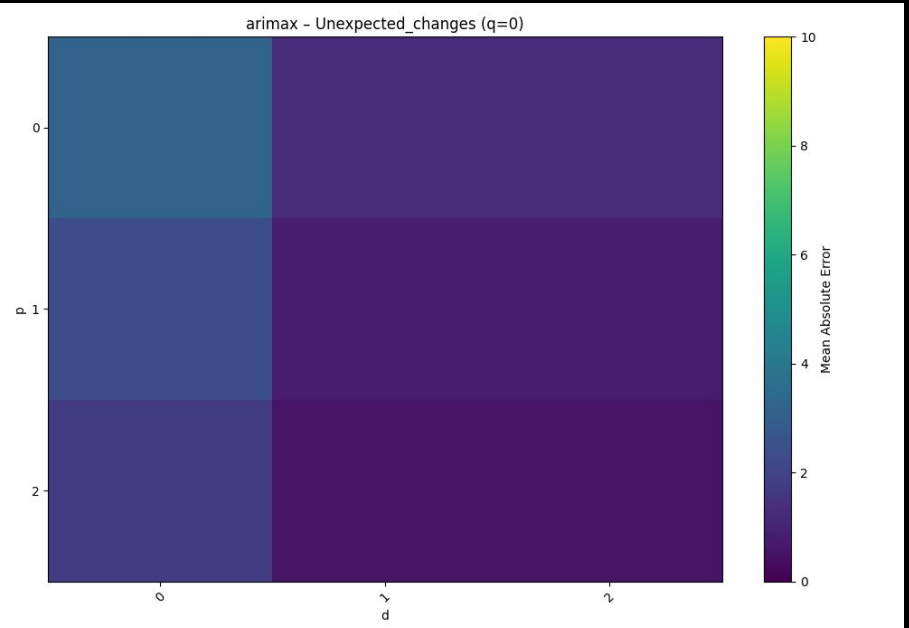


Analysis - Unexpected Changes

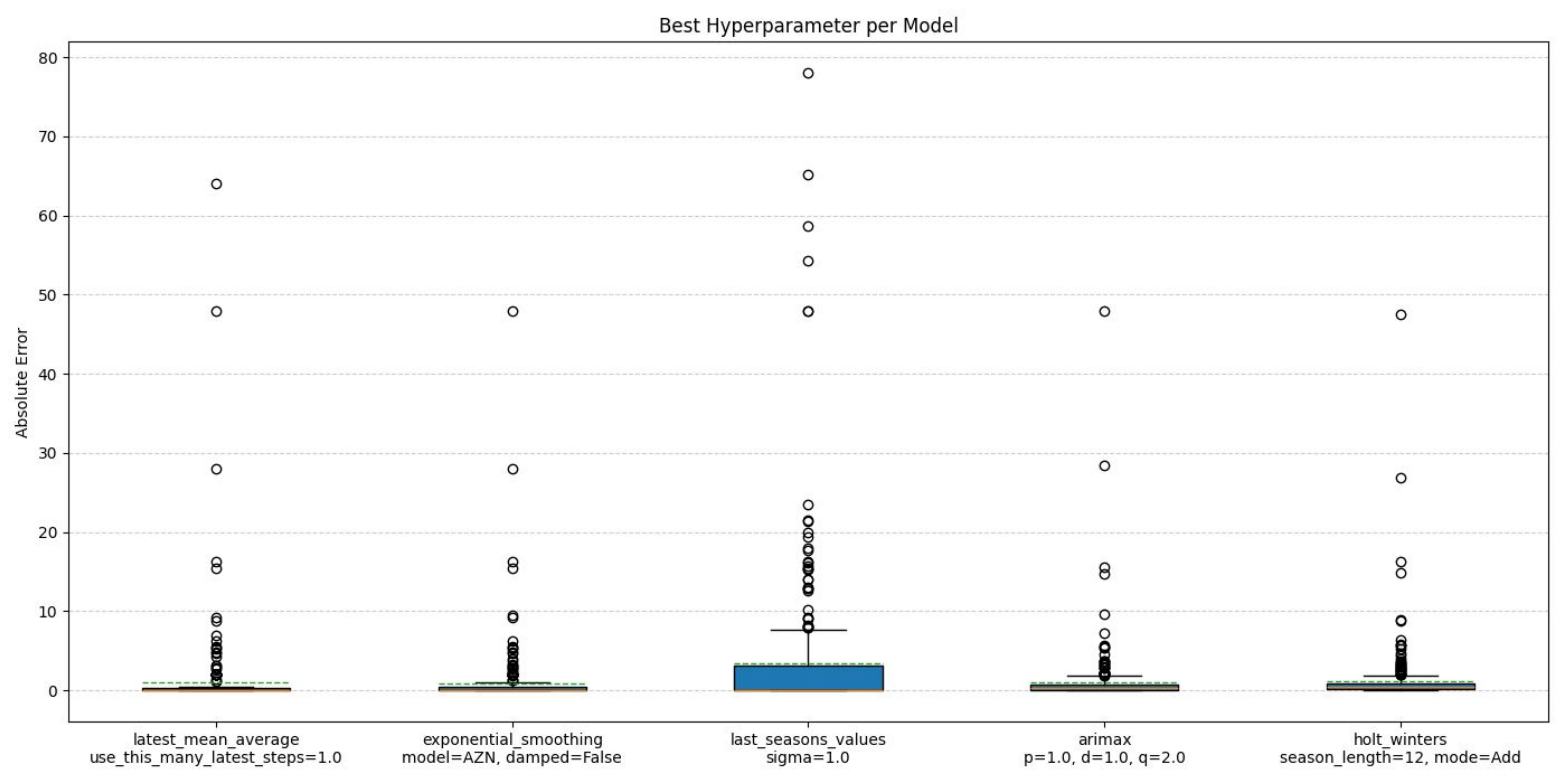
Exponential Smoothing



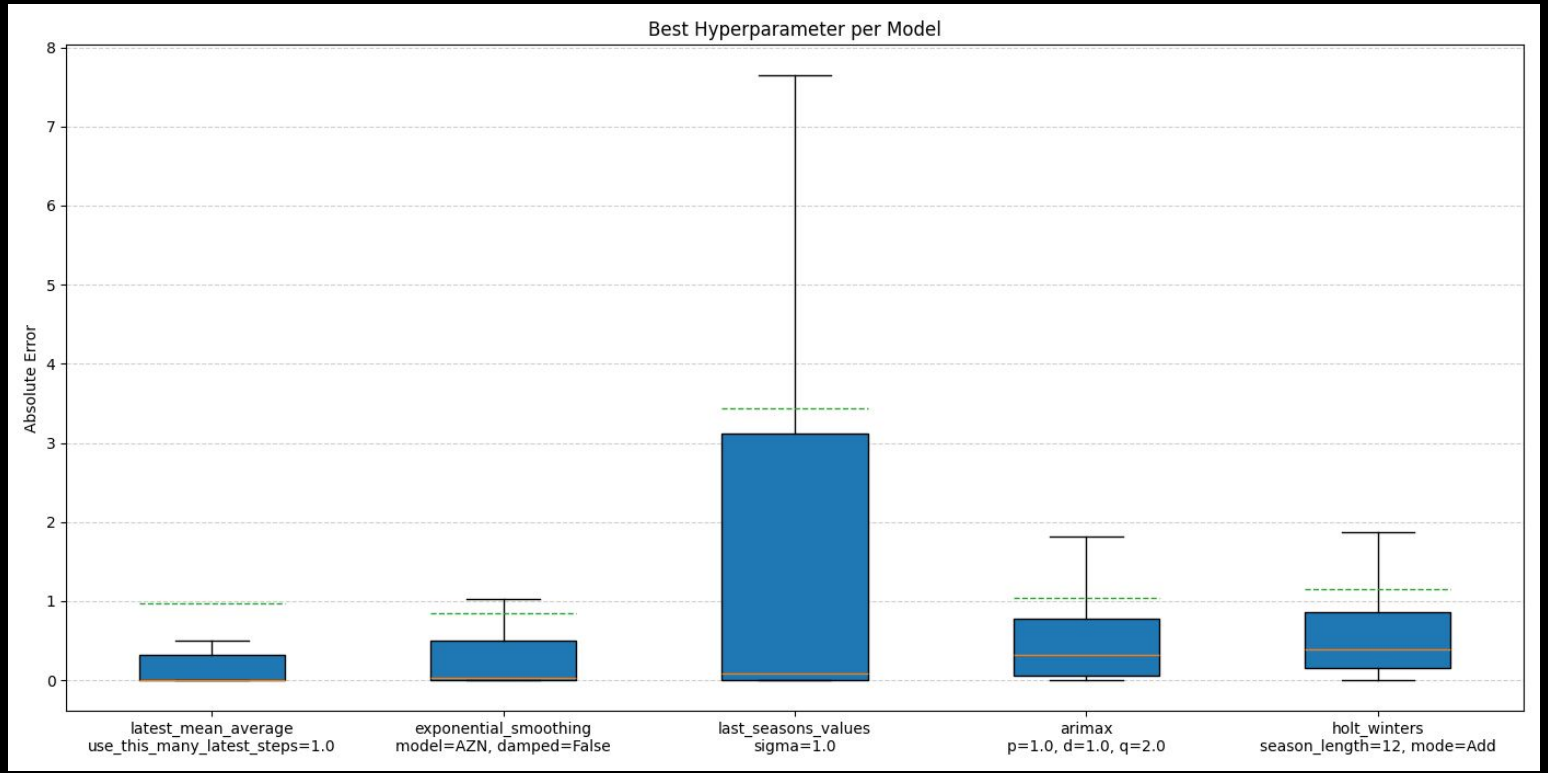
Arimax



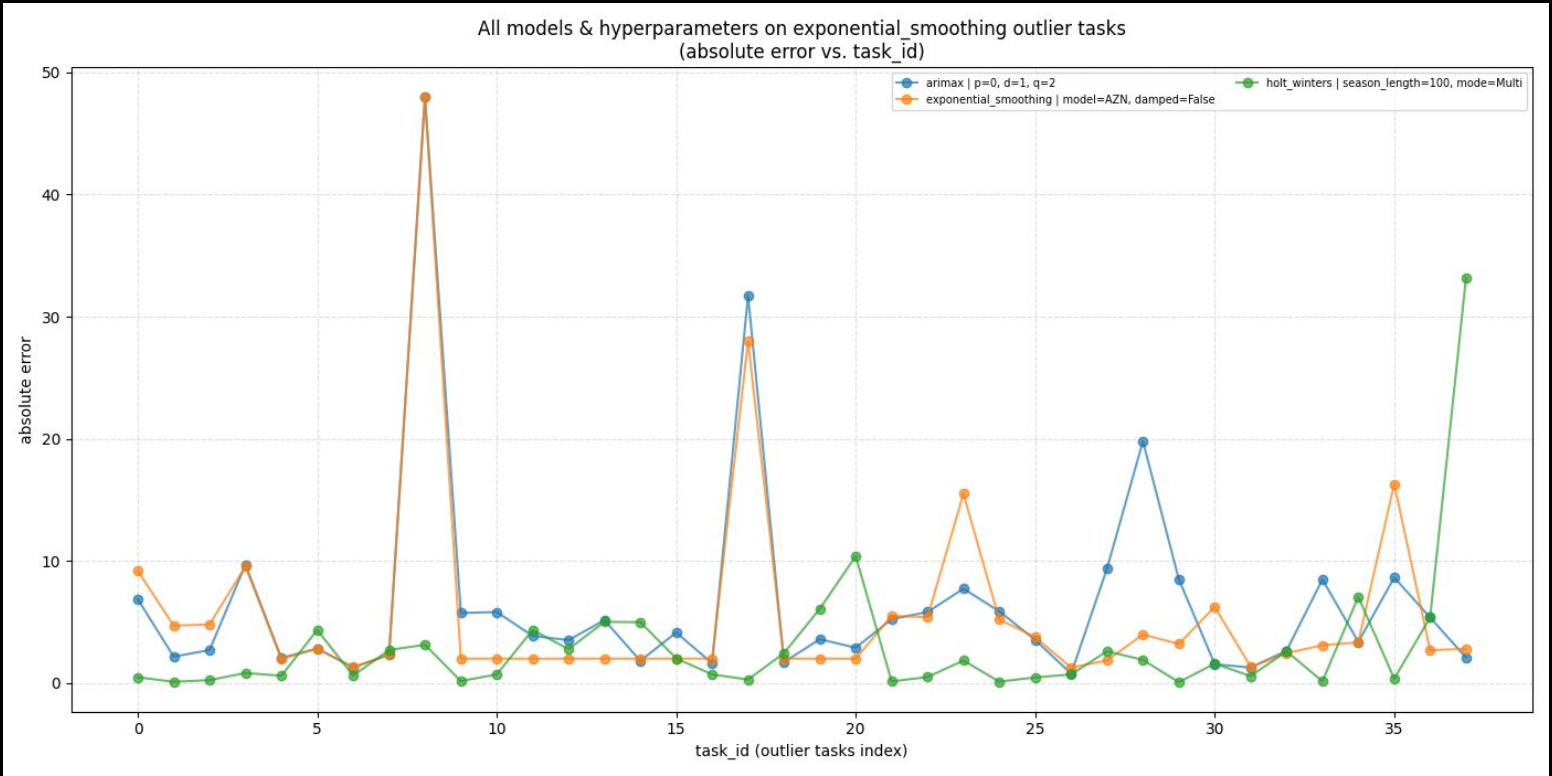
Analysis - Only Best Hyperparameters



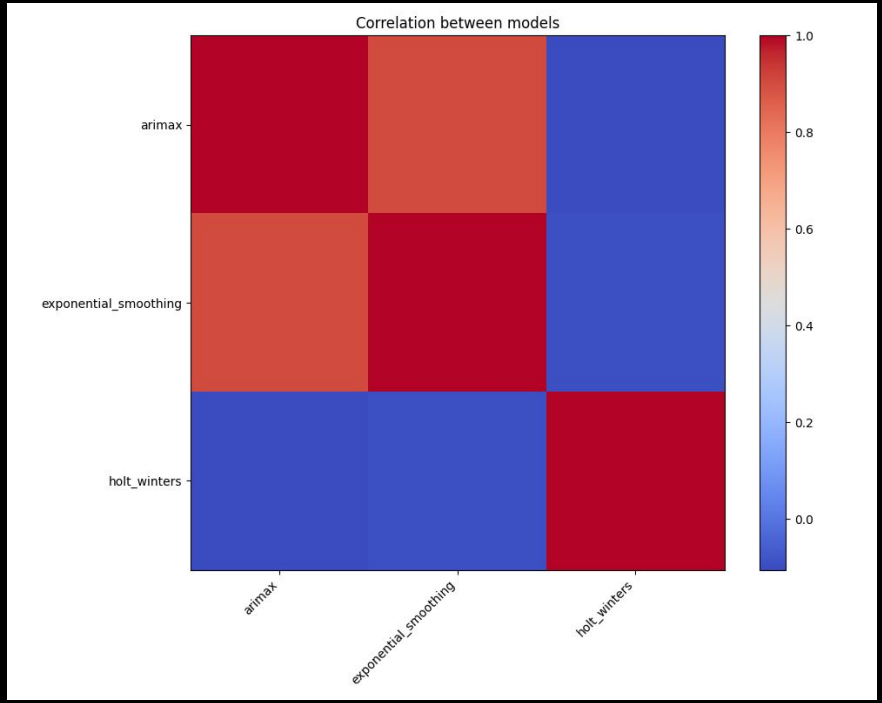
Analysis - Only Best Hyperparameters



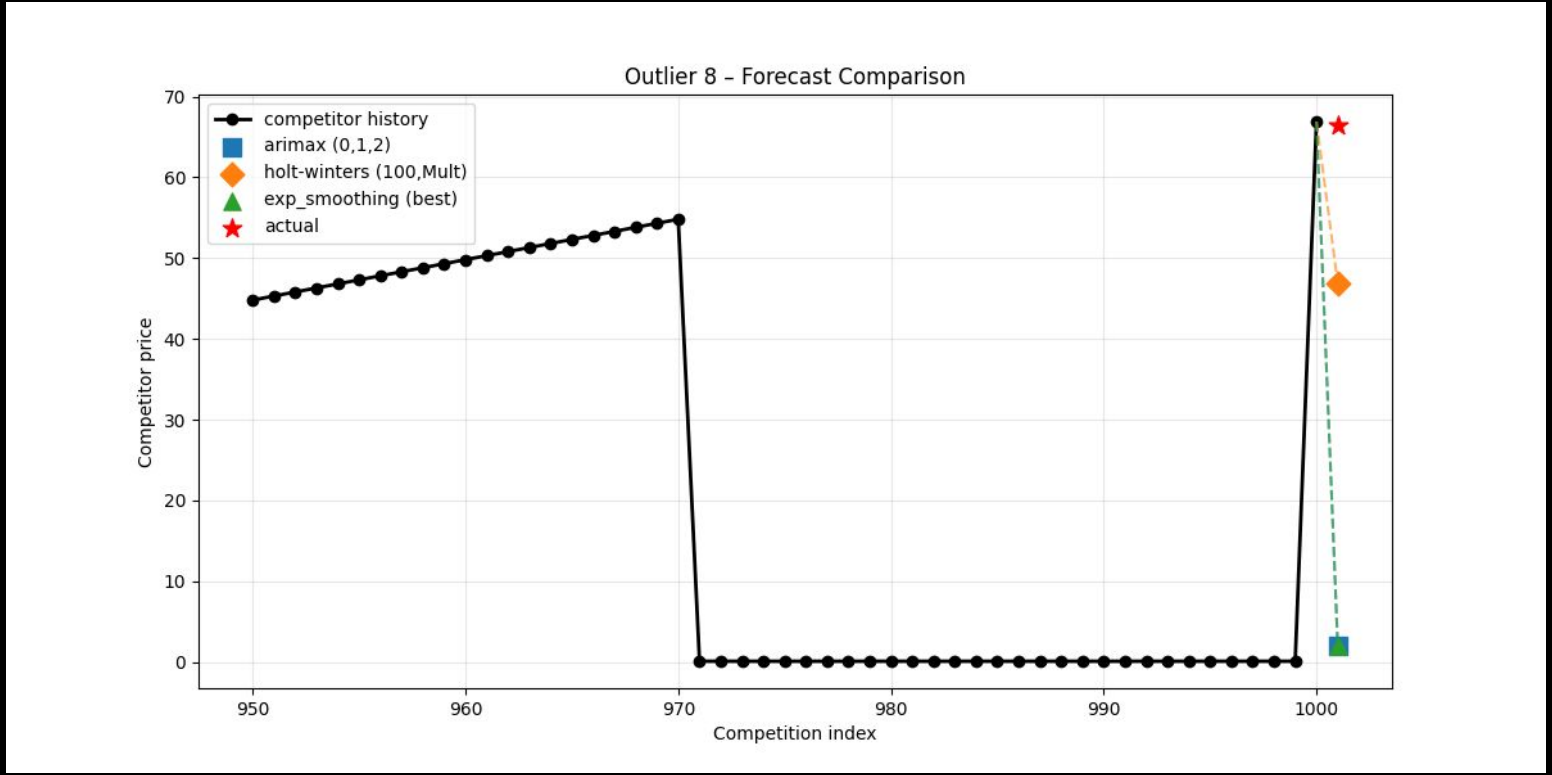
Analysis - Outliers of exp. smoothing



Analysis - Outlier Model Correlations



Analysis - Outliers

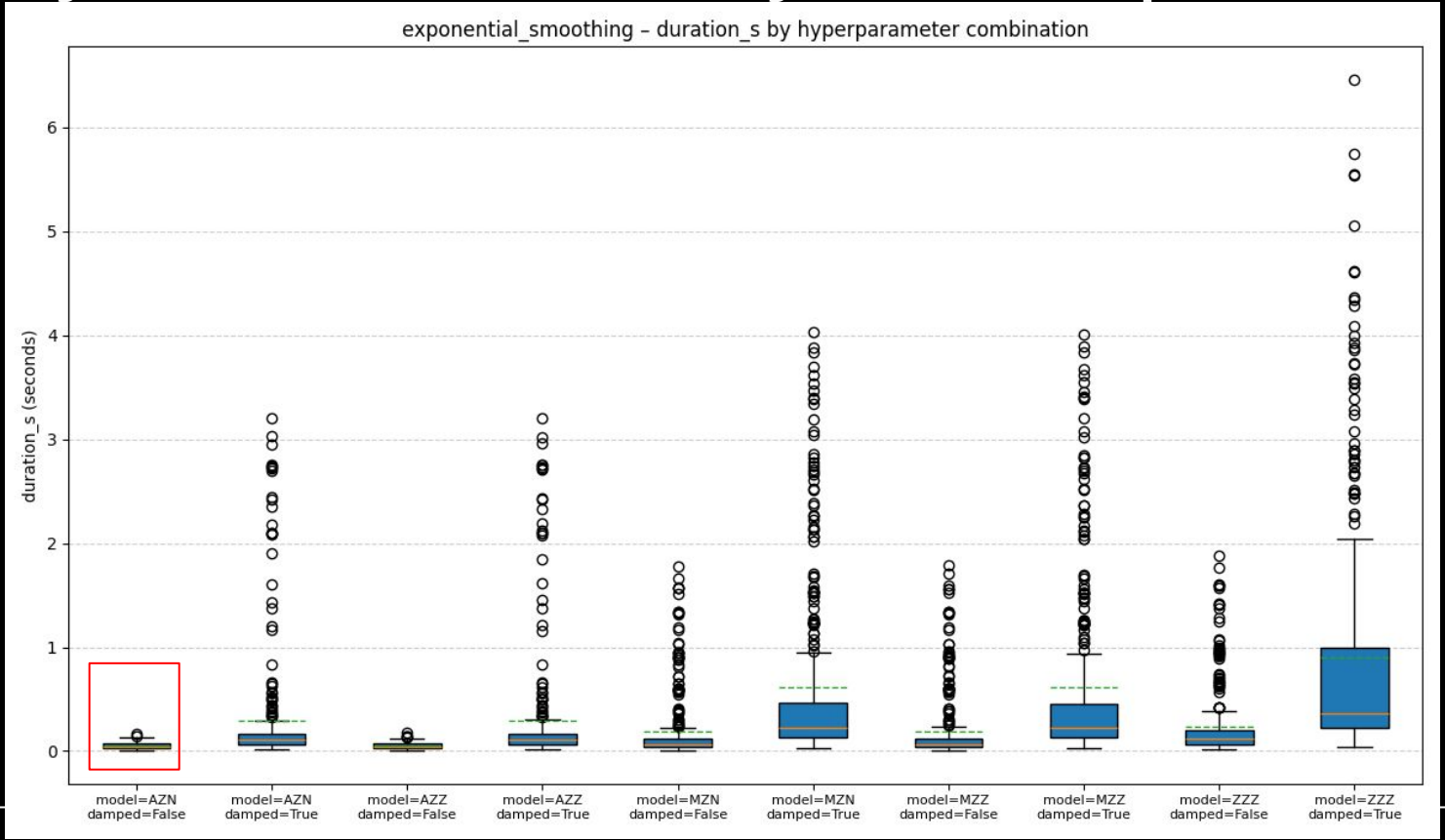


Analysis - Breakdown

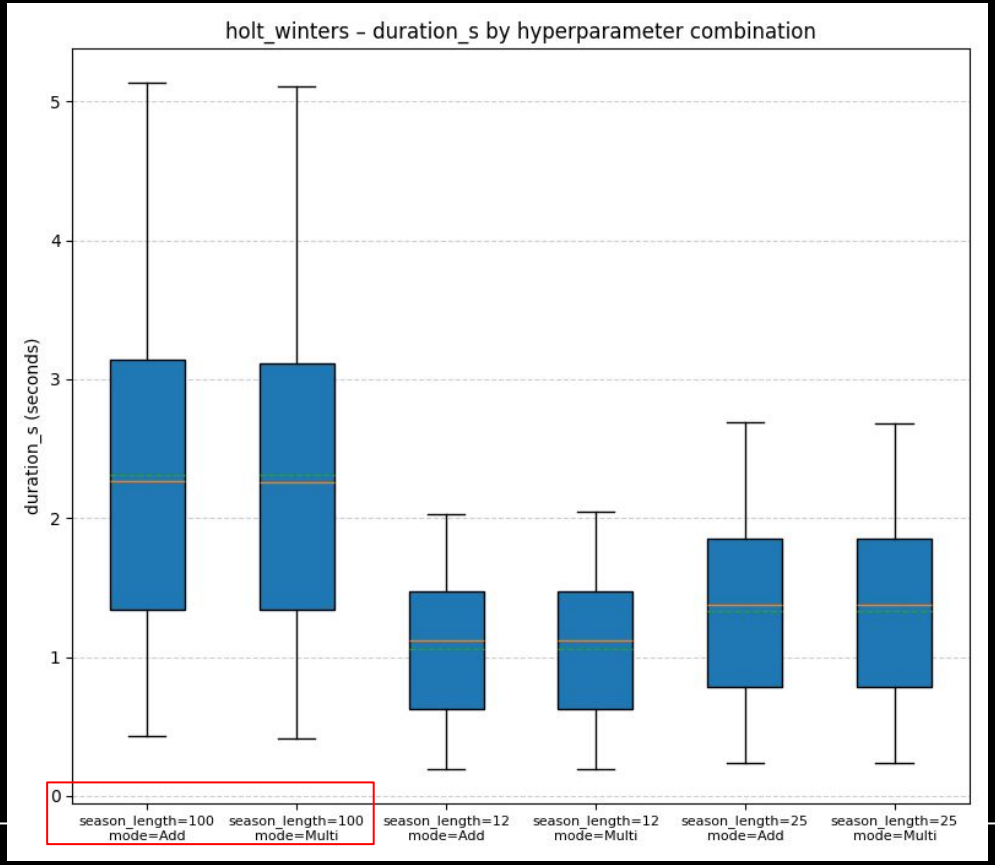
- On average exponential smoothing predicted best on given competitions.
- At some tasks, holt-winter *(Mult & 100 seasons)* might be better

What about the computation time?

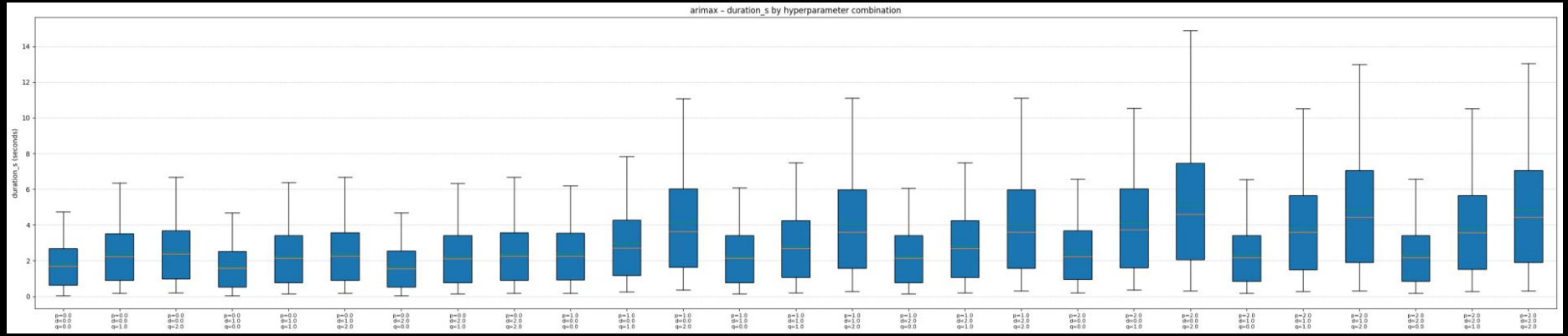
Analysis - Time Analysis - expo smooth



Analysis - Time Analysis - holt winter



Analysis - Time Analysis - holt winter

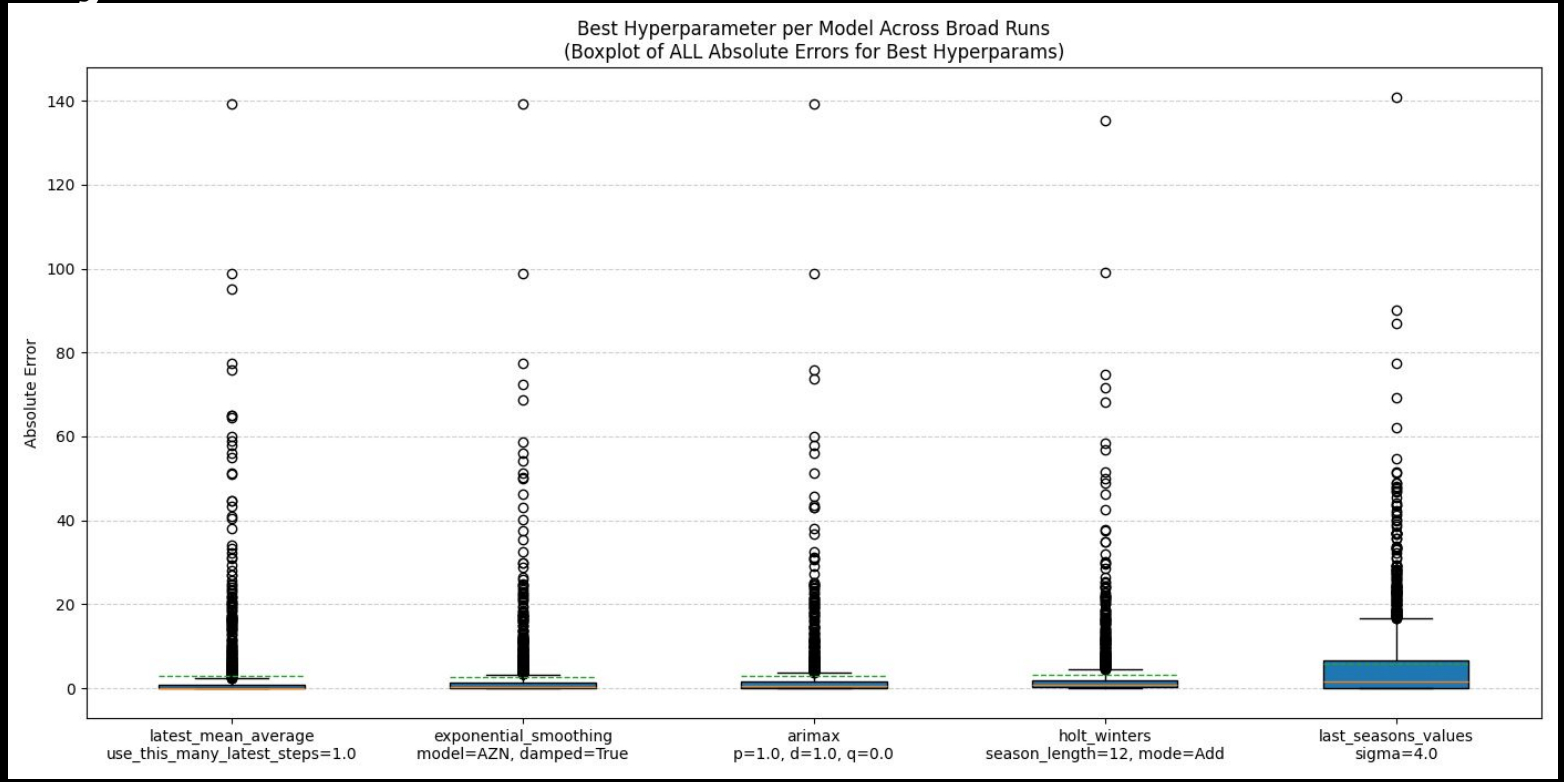


Forecast Models - Changed Setup

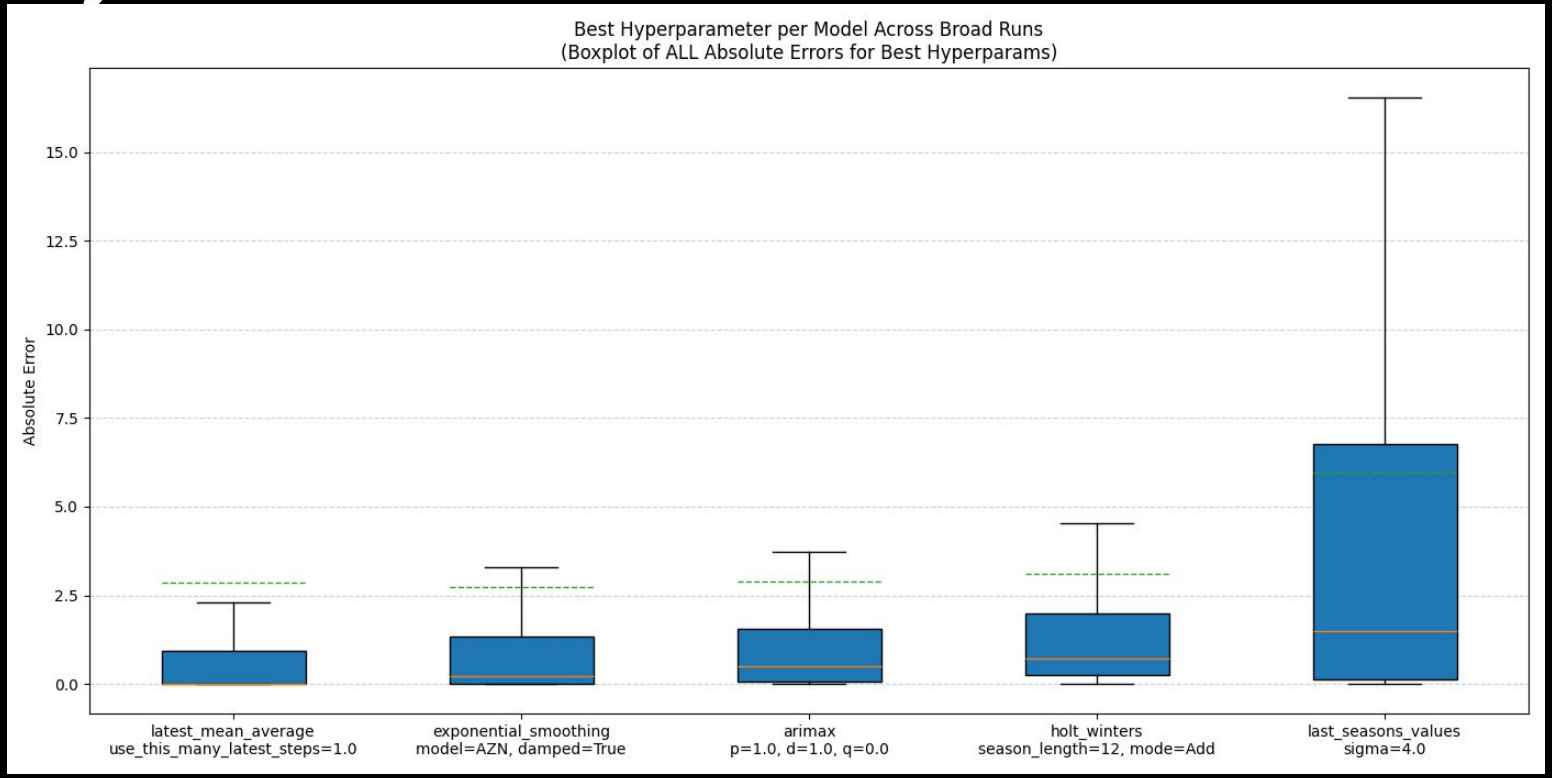
Setup:

- For Each Competition:
  - Sample 12 random points
  - For Each Model Parameter Setup:
    - Predict next price
    - Calculate MAE

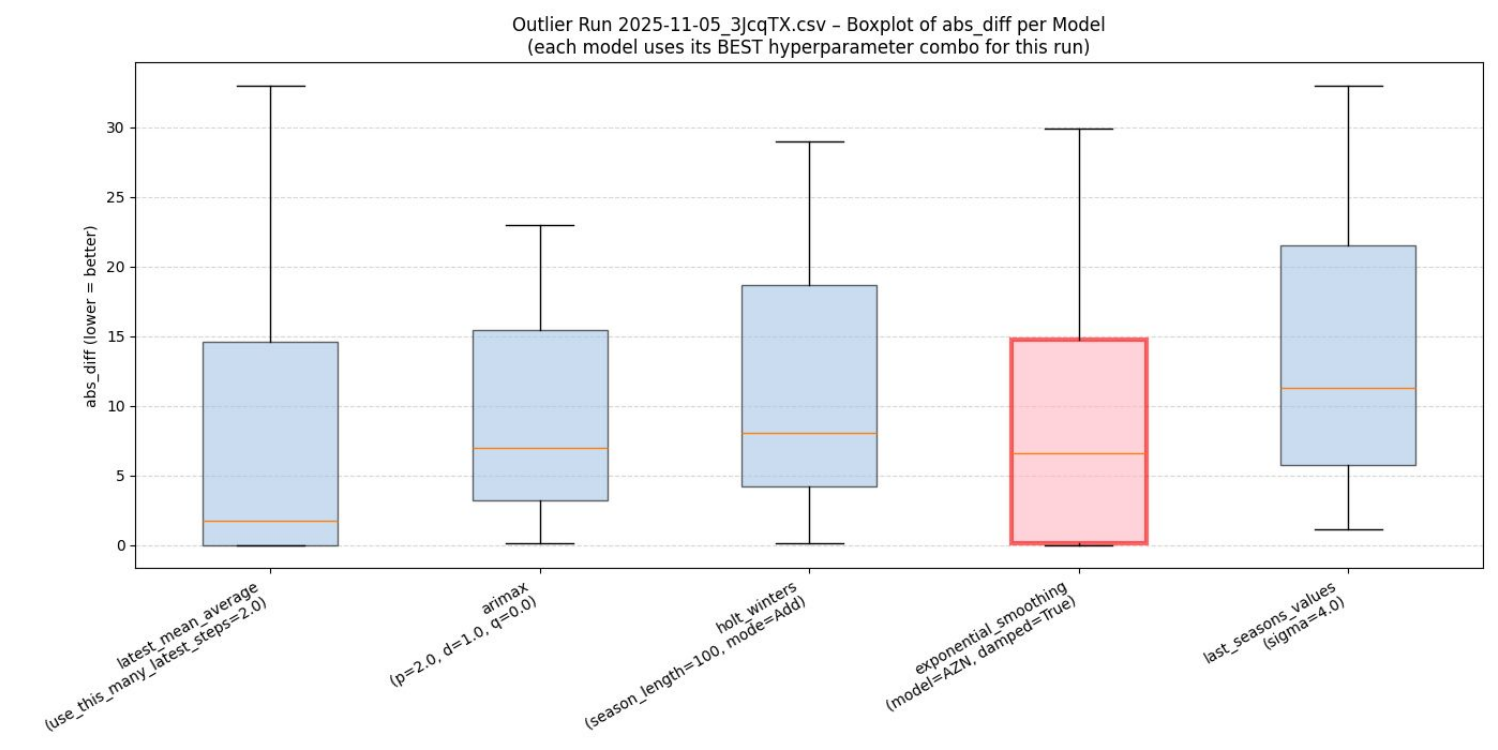
Analysis - All Data



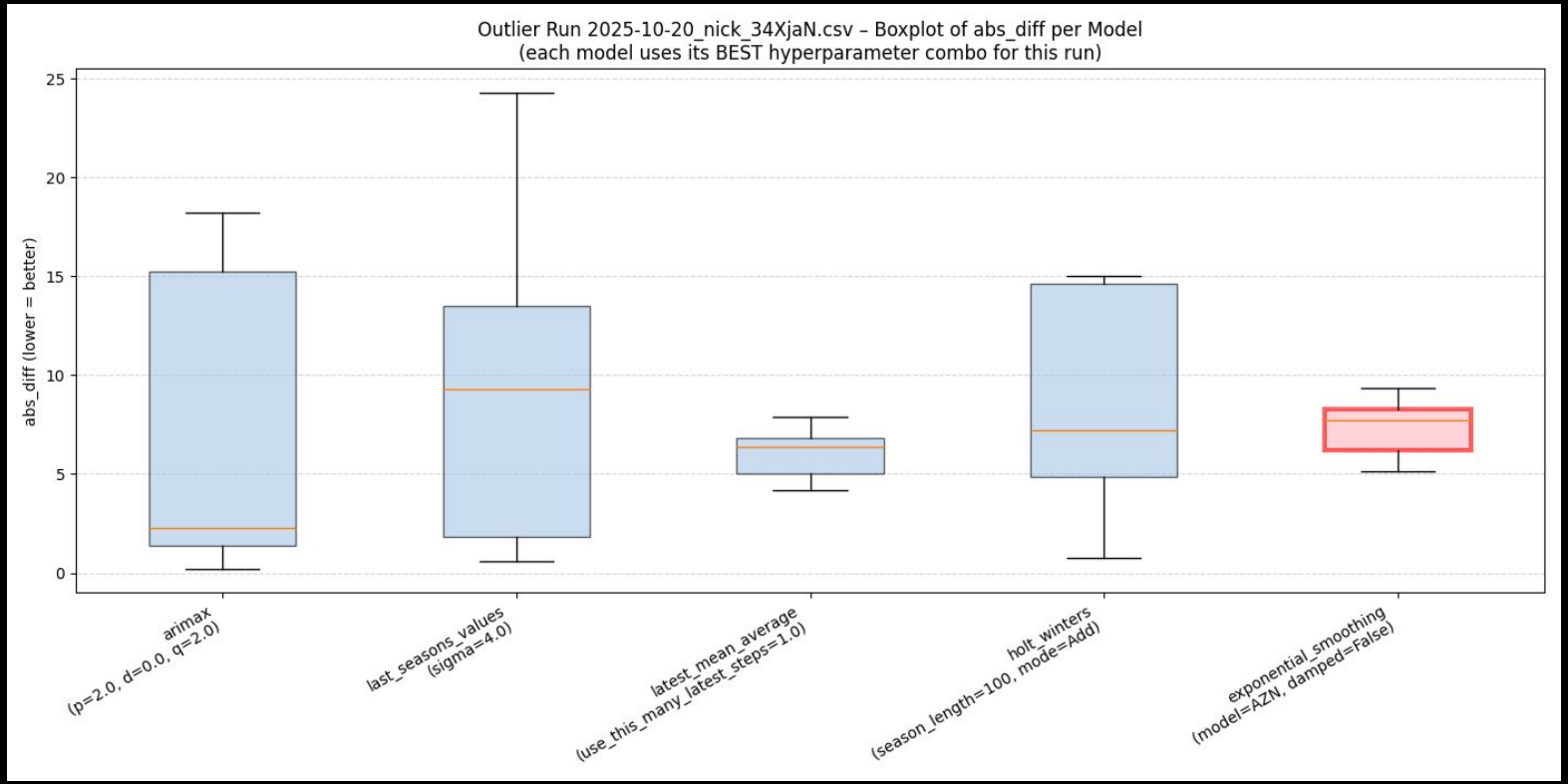
Analysis - All Data



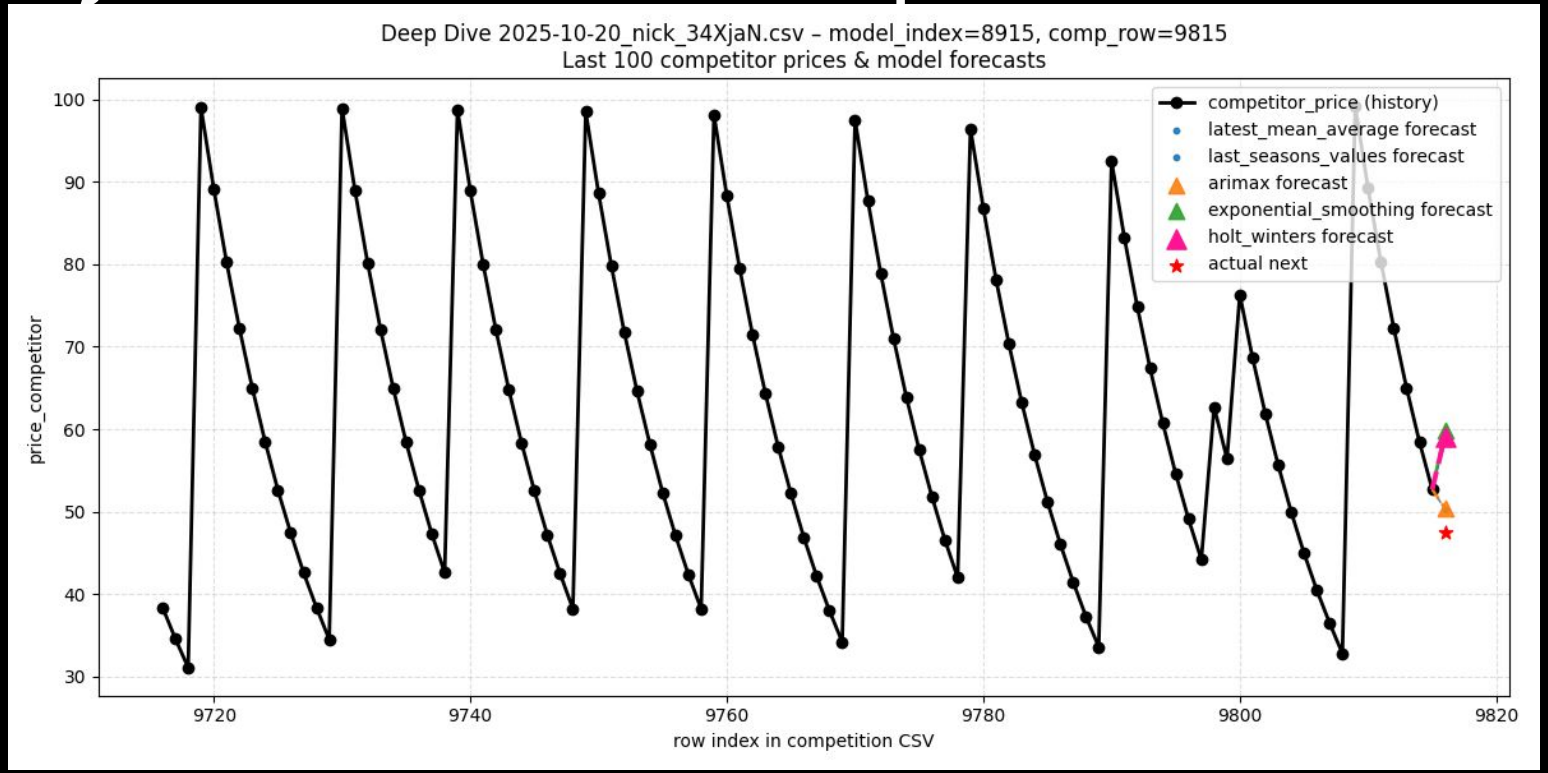
Analysis - Outlier Comparison



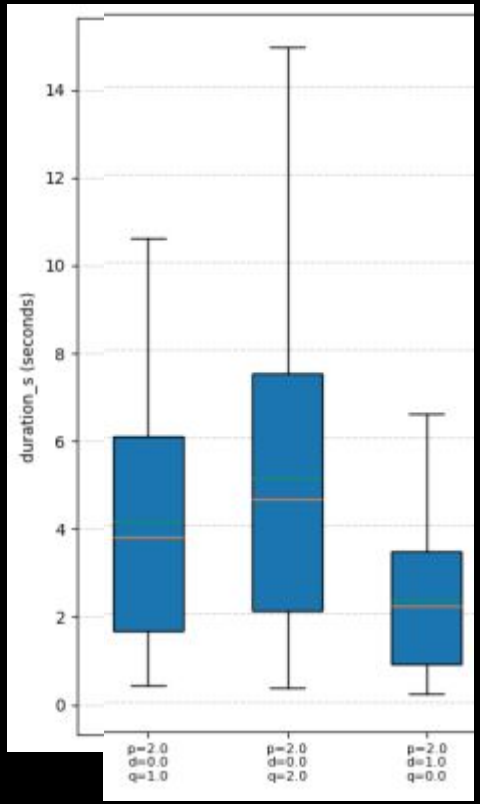
Analysis - Outlier Comparison



Analysis - Outlier Comparison

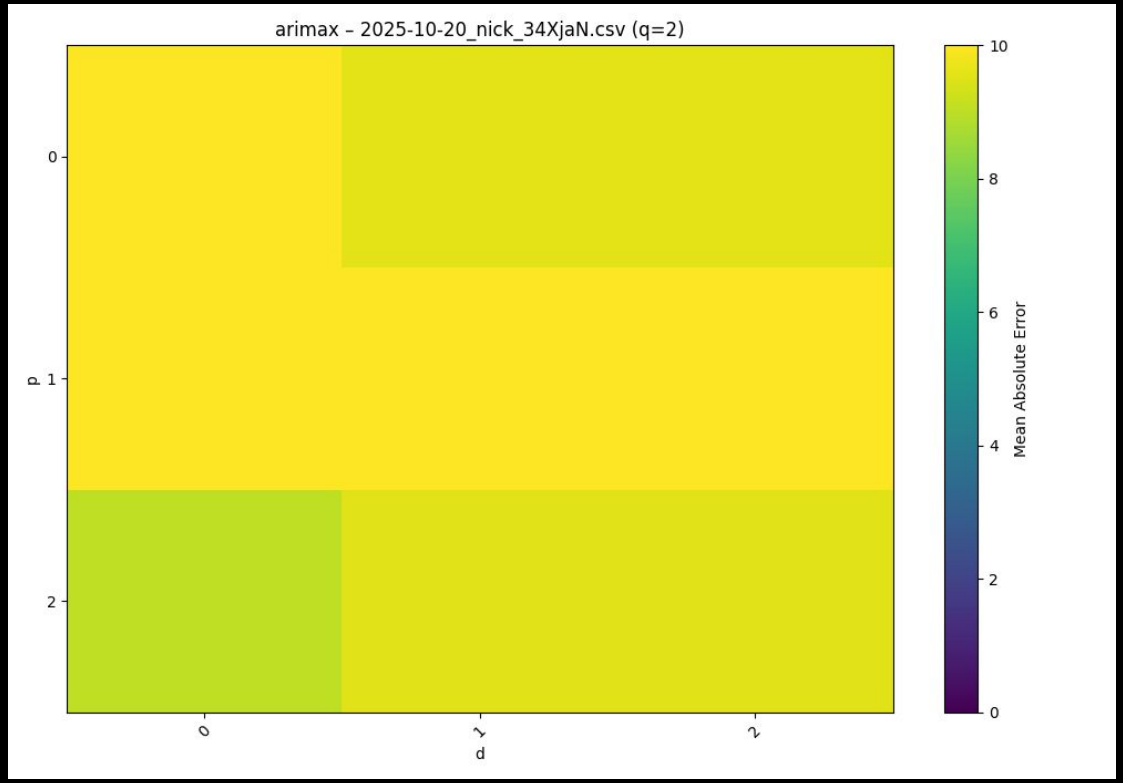


Analysis - Outlier Comparison



Sadly it is expensive

Analysis - Outlier Comparison



Conclusion (Raph)

- Exponential Smoothing works well in this duopoly
  - Still analyse the competitions, where exponential smoothing fails
  - Maybe there is a better algorithm out there
- Rather spend computation time on a better demand model

All models are wrong, but some are useful

1976, George Box