Presentation 3 ML

Group: WiseGoose-IntrepidBeluga

Members: Nicholas Chandler, Raphael Bergner

Agenda

01 Nick’s Approaches

1. Addressing the Competitor
2. ML
3. Overall Analysis

02 Raphael

1. ML

Nick

Time Series - Adversarial Pricing

- If the competitor is in stock:
  - Price 2% lower than the predicted competitor next price.

- If they're not:
  - Do basic inventory management assuming we want to sell evenly

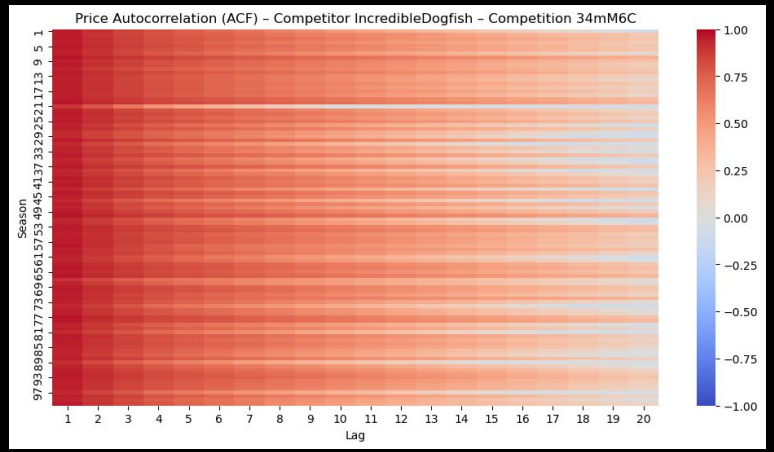
```
# Predict 2% lower than the competitor:
if competitor_has_capacity_current_period_in_current_season:
    price = 0.98 * comp_price

    # Adjust slightly based on remaining inventory vs expected inventory
    # If overstocked, reduce price more to sell faster
    if ratio > 1:
        adjustment_factor = AGGRESSIVENESS * (ratio - 1)
        price *= (1 - adjustment_factor)

else:  # Comp out of capacity - ignore their price.
    # 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))

    # smooth trend: increase slightly as the season goes on (time-based upward drift)
    time_factor = 0.1 * (selling_period_in_current_season / T_SEASON)
    price *= (1 + time_factor)*MONOPOLY_MARKUP

# Try to get the comp to go high at the beginning - in case we have to follow them to the ground
if selling_period_in_current_season < 5:
    price = 80

price = float(round(min(max(price, MIN_PRICE), MAX_PRICE), 2))
return price, new_info
```

Time Series - Autocorrelation Plots

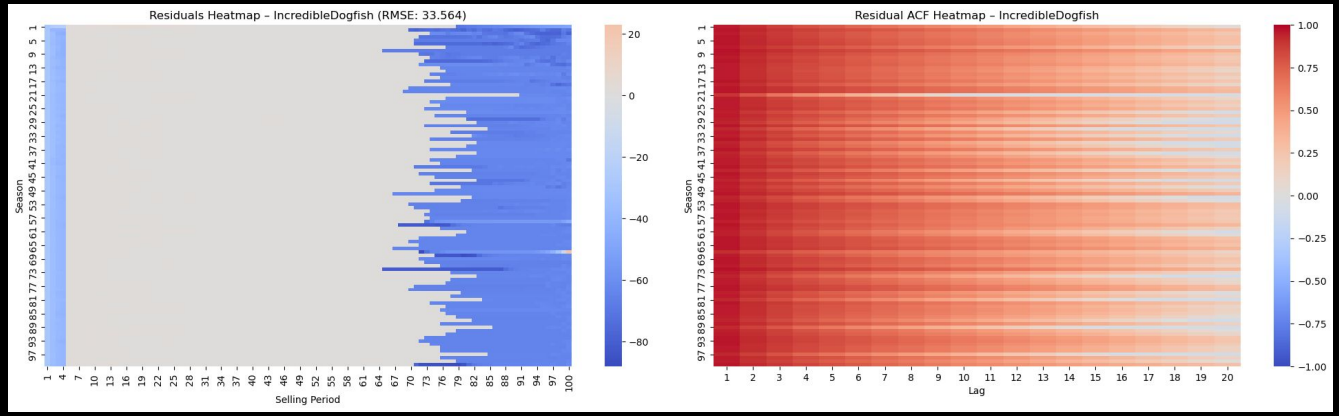
- Autocorrelation Definition:

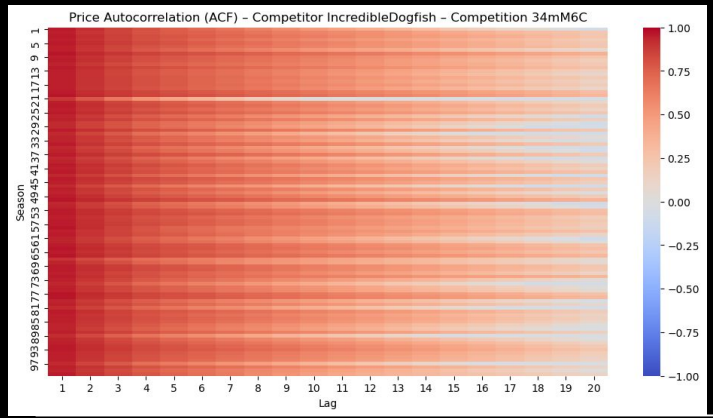
$$\rho(k) = \frac{\mathrm{Cov}(X_t, X_{t-k})}{\mathrm{Var}(X_t)} = \frac{\mathbb{E}[(X_t - \mu)(X_{t-k} - \mu)]}{\mathbb{E}[(X_t - \mu)^2]}$$

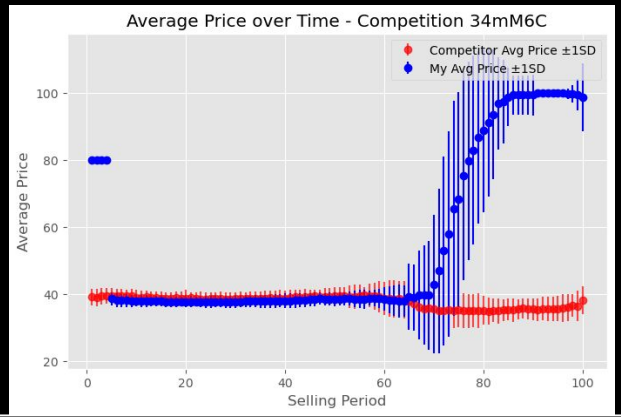
- How to read the plots:
  - Sign:
    - **Positive:** Prices continue in same direction
    - **Negative:** Oscillatory behavior
  - Magnitude:
    - **Near +1/-1:** Strong dependence between values
    - **Near 0:** Weak dependence
  - Lag Number:
    - **Short Lag:** Immediate reactions
    - **Medium Lag:** Slow adjustment patterns
  - Decay Pattern (of magnitude):
    - **Slow-decaying Positive:** Follows trend
    - **Alternating Signs:** Oscillation / Undercutting
    - **Rapid Decay:** Mostly random behavior



Time Series - Adversarial Pricing

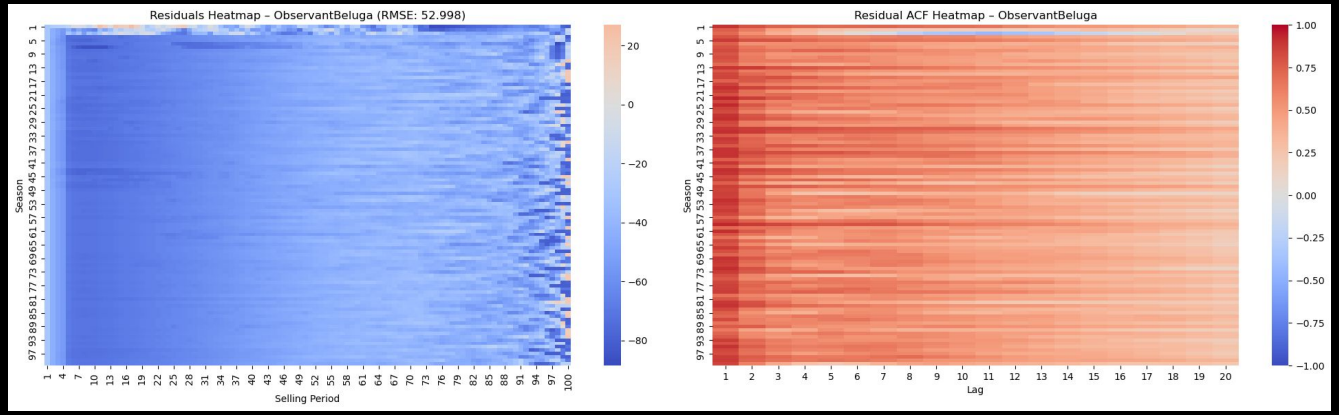


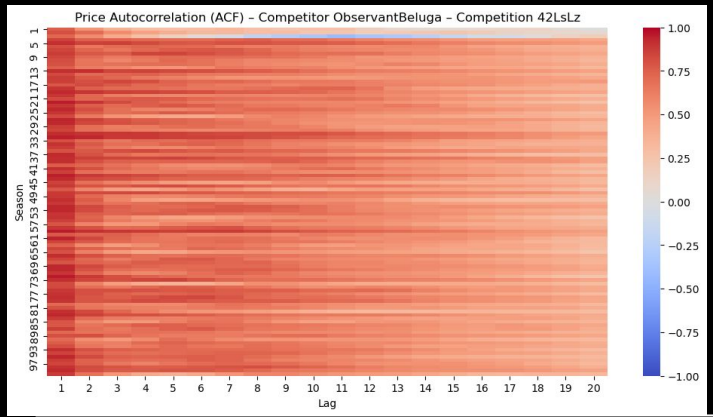


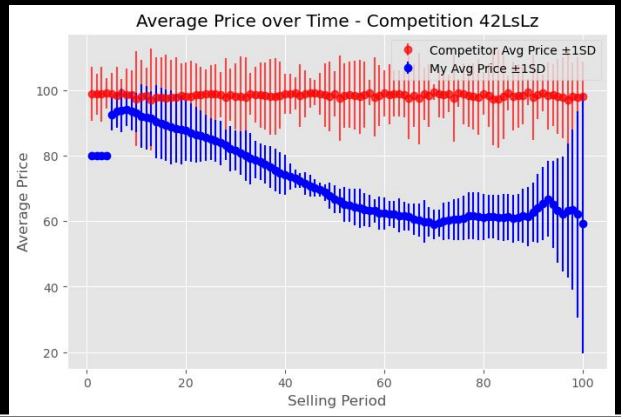


5	IntrepidBeluga	$1,953,265	$488,316	1.25	$1.00	$100.00	$69.05	0.893	0.855

Time Series - Adversarial Pricing

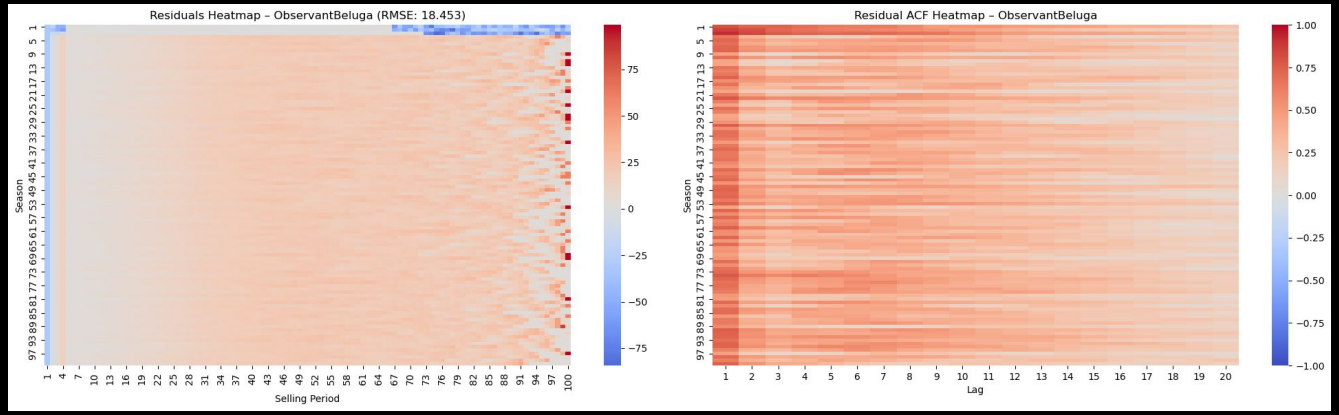


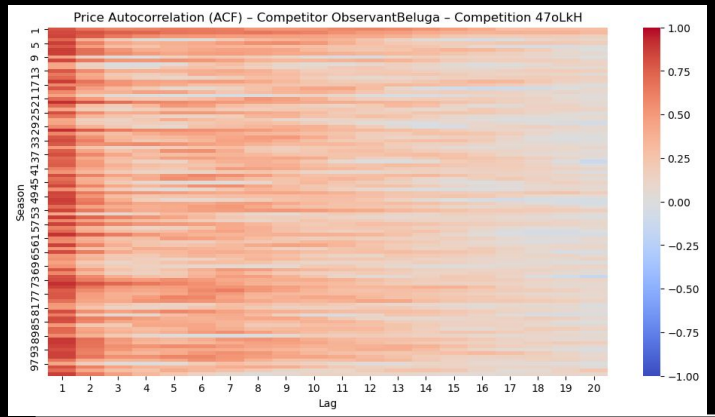


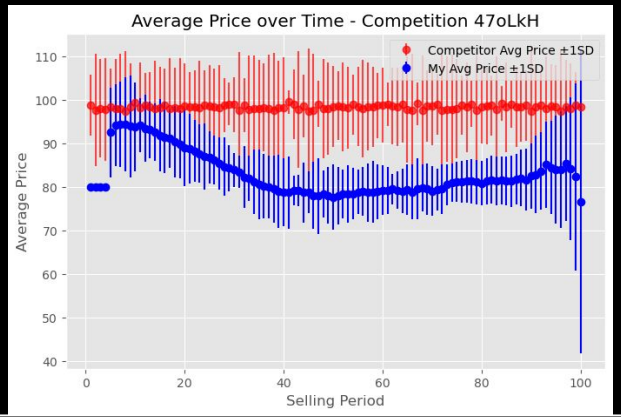


5	IntrepidBeluga	$1,953,265	$488,316	1.25	$1.00	$100.00	$69.05	0.893	0.855

Time Series - Adversarial Pricing

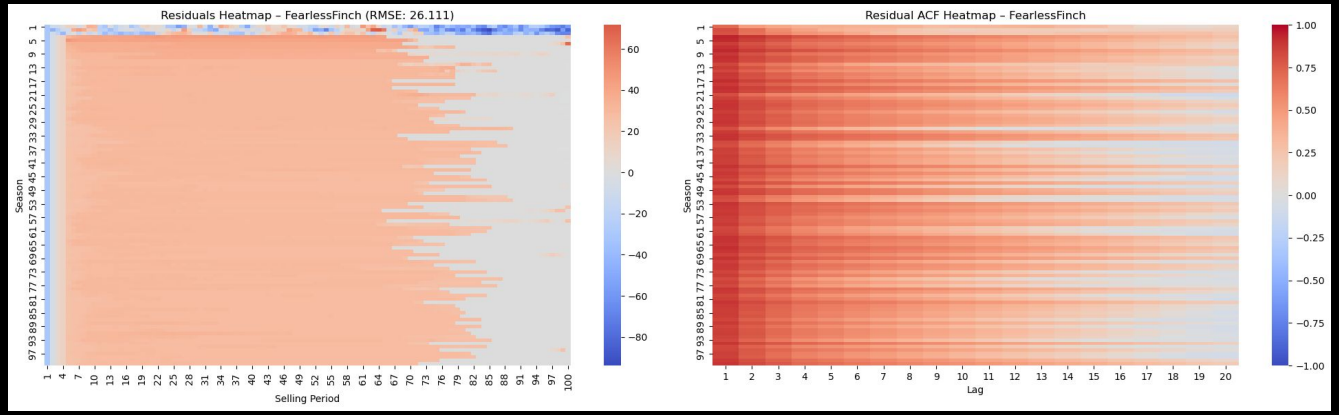


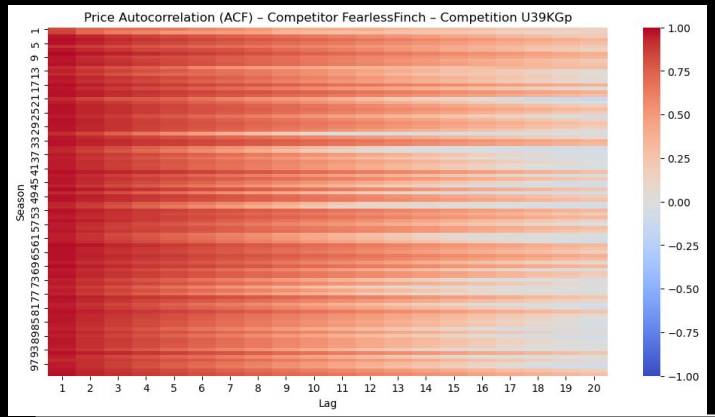


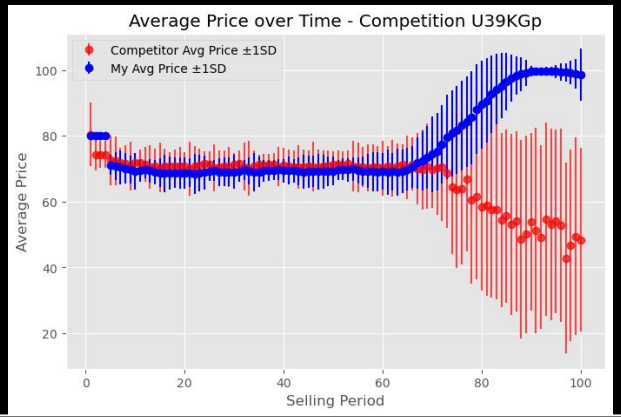


5	IntrepidBeluga	$1,953,265	$488,316	1.25	$1.00	$100.00	$69.05	0.893	0.855

Time Series - Adversarial Pricing







5	IntrepidBeluga	$1,953,265	$488,316	1.25	$1.00	$100.00	$69.05	0.893	0.855

XGBoost

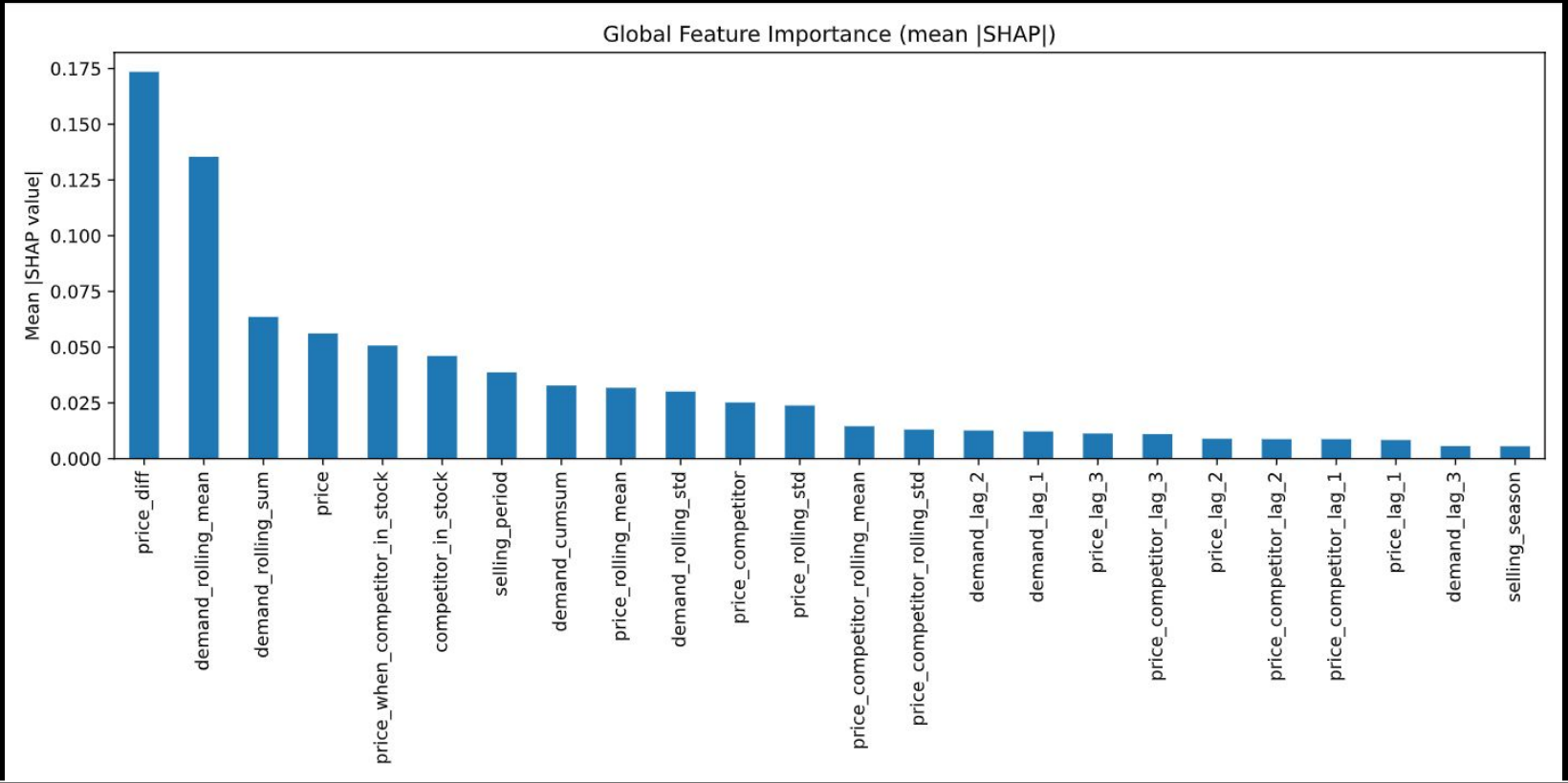
- **Data (Same total set as Raph):**
  - 1890000 periods of competitions from Raph and I.
  - Last 20 competitions are held out for the test set.

- **Features:**
  - 29 features including
    - Competitor stock
    - Lagged/Rolling means and std. devs of columns from competition details
    - Rolling correlations between prices
    - Price difference

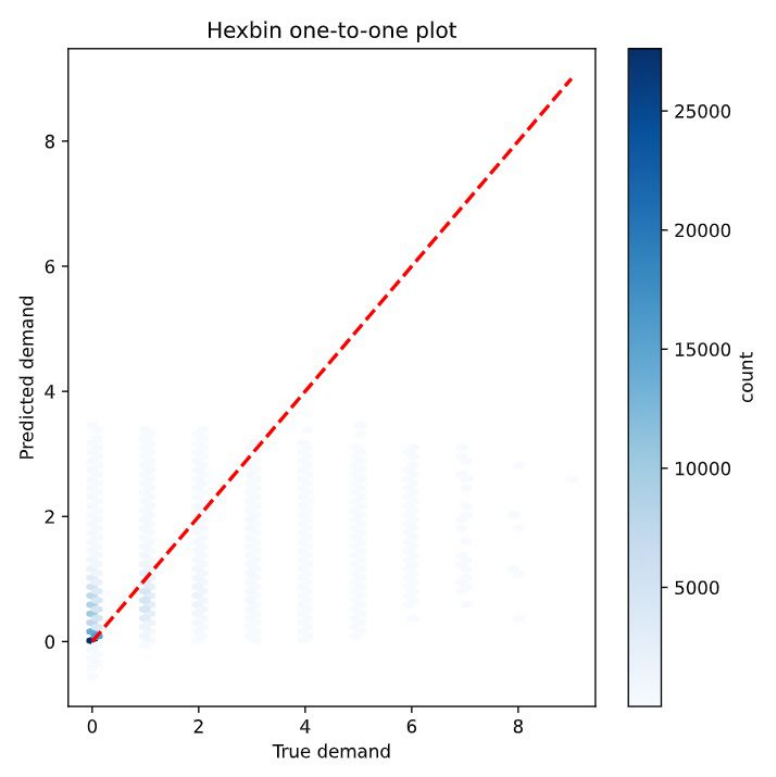
- **Training Pipeline:**
  - 5-Fold Cross Validated Random Search for 30 trials

- **Best Model:**
  - Best RMSE: 0.7373
  - Best parameters: {'subsample': 0.6, 'reg_lambda': 1.5, 'reg_alpha': 0, 'n_estimators': 300, 'min_child_weight': 3, 'max_depth': 9, 'learning_rate': 0.1, 'colsample_bytree': 0.8}

XGBoost - Feature Importances



XGBoost - Model Performance

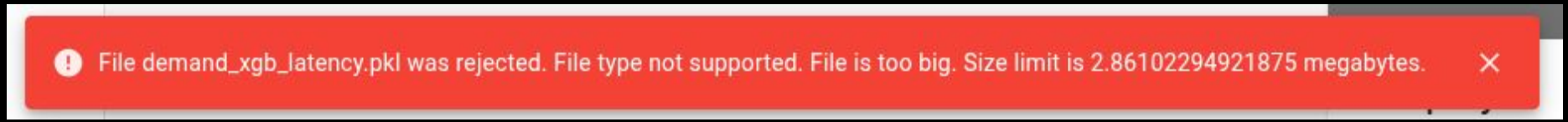


- **Test Metrics:**
  - **MSE**: 0.6348
  - **MAE**: 0.5463
  - **R-Squared**: 0.1515
  - **RMSE:** 0.7967
- **Interpretation:** The model mispredicts demand by approximately 0.7967 on average.
- **Consensus:** Not very good... But wait!

XGBoost

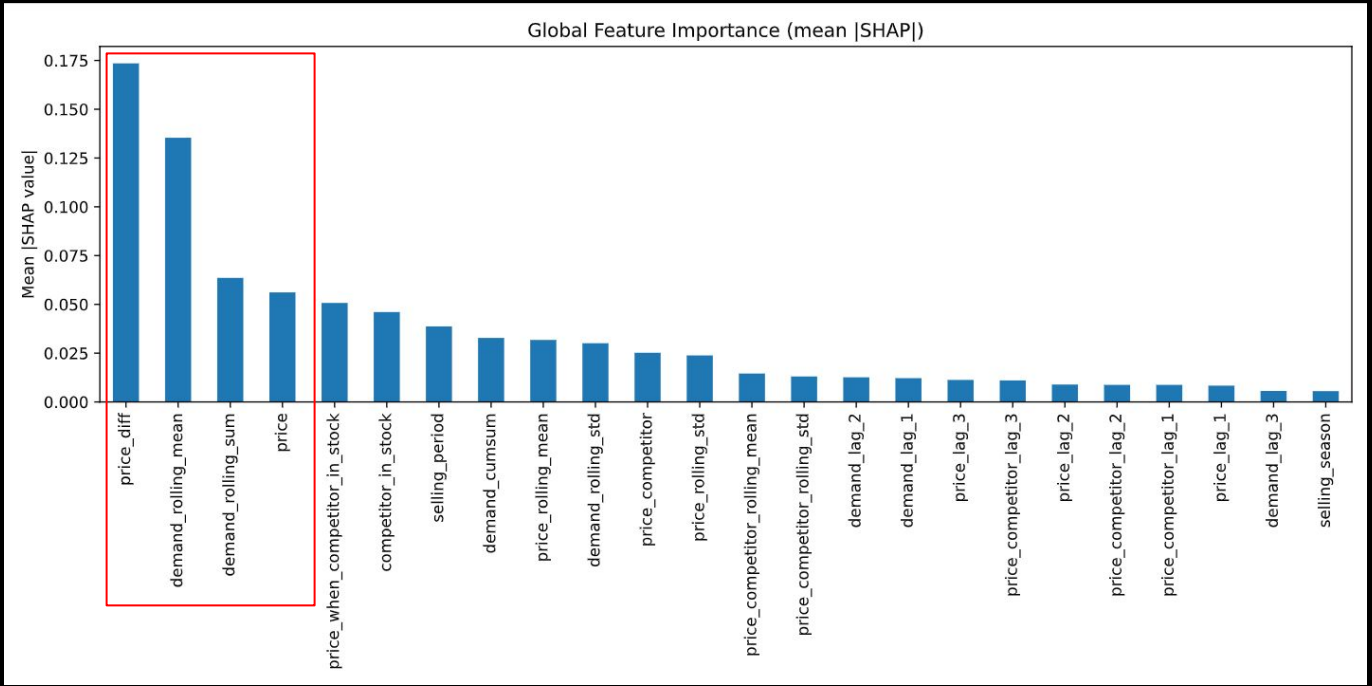
The model was far too big at 48Mb...

So we need another approach.



We also switch to JSON at this point too.

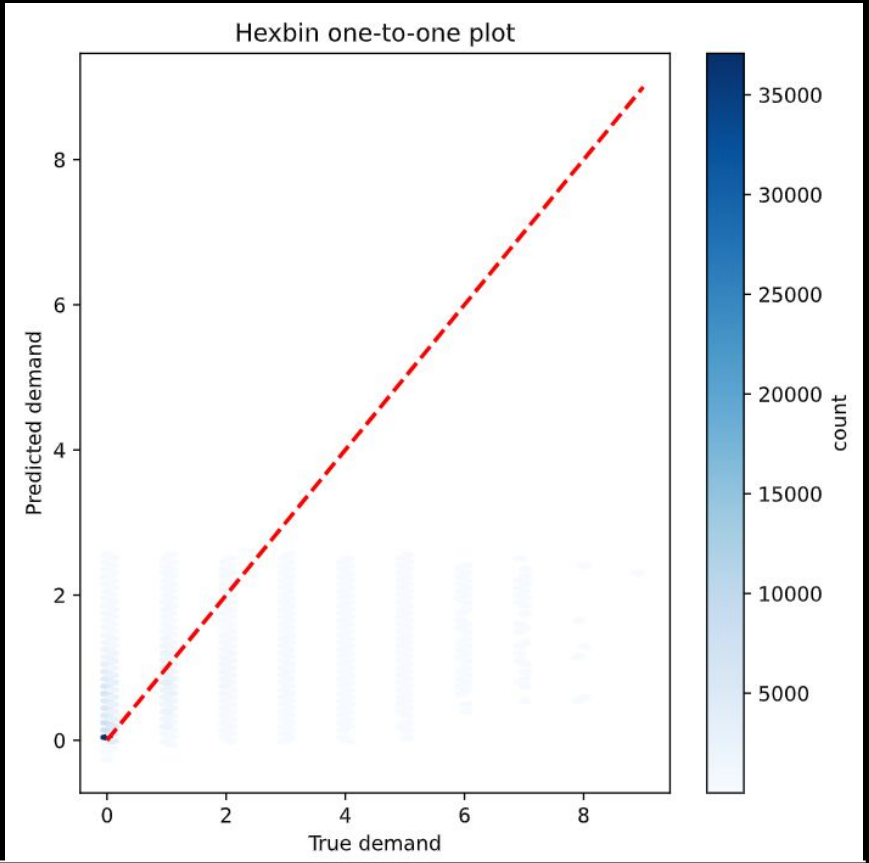
XGBoost



- **Selected features:**
  - Price difference
  - Rolling mean/sum of demand
  - My price
- **Size Limits:**
  - Fewer estimators
  - Fewer splits

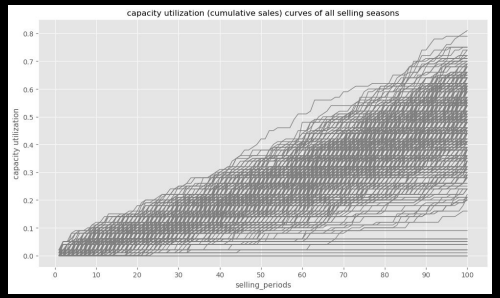
XGBoost

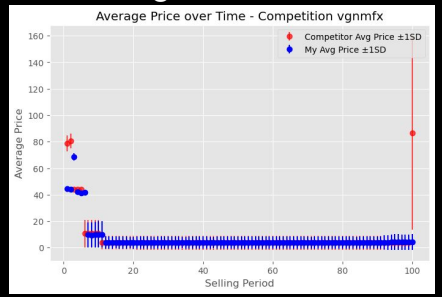
- XGB with fewer features
- Still CV HPO using random strategy for 10 trials.
- Training Results:
  - **Test RMSE:** 0.7931
  - **Test R-Squared:** 15.92%
- Still not great in terms of performance
- Feature engineering and a stronger model didn't make a massive difference in my setup
- Interestingly, the SHAP analysis seemed to be correct since the elimination of ~20 features didn't bring down performance.

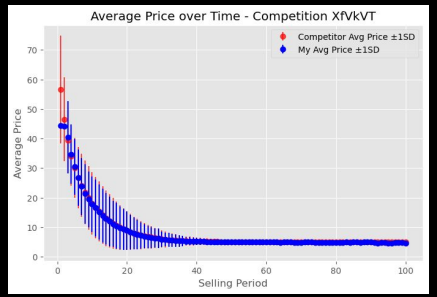


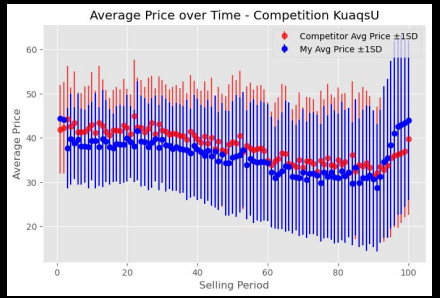
XGBoost

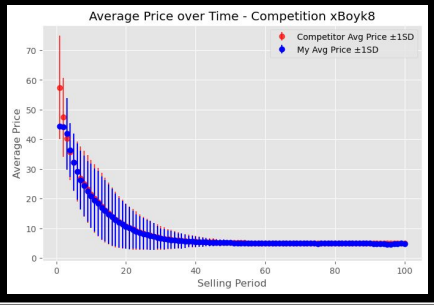
- In this, there was a predetermined selling model where a target demand was aimed for.
- We tried 15 different prices in the ML model for demand to see which had the highest price at the target demand
- On a low demand day we did alright

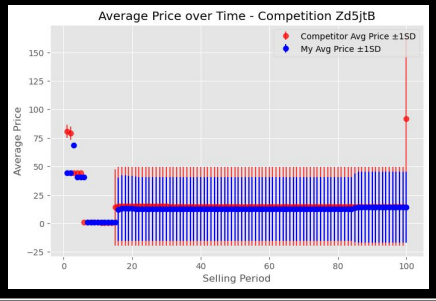


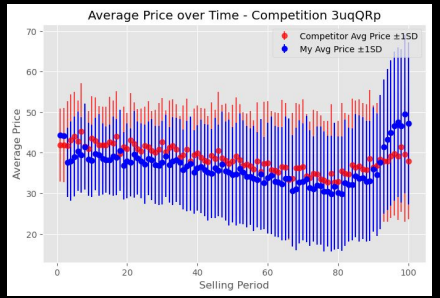












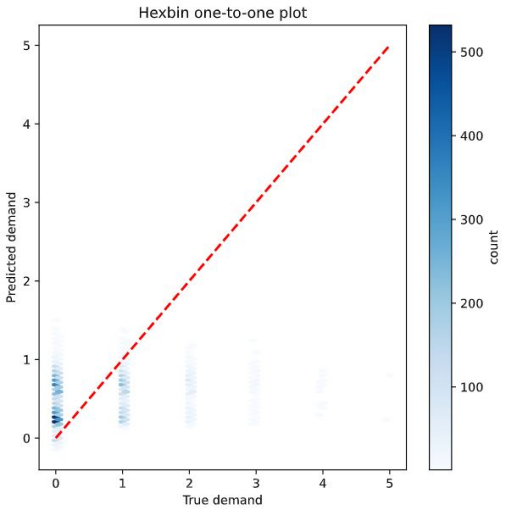
22	IntrepidBeluga	$380,280	$63,380	1	$1.00	$94.80	$18.42	0.763	0

NN

- Made a small NN, done with less data to speed up training, same pipeline as XGB
- Parameters: {'hidden_dim': 20, 'n_layers': 2, 'dropout': 0.0772, 'lr': 0.00595, 'batch_size': 512}
- Performance (offline data):
  - Test RMSE: 0.6916
  - Test MAE: 0.5518
  - Test R-Squared: 0.0054

Better RMSE but far worse R-Squared...

The test set was also smaller so this could be part of the issue

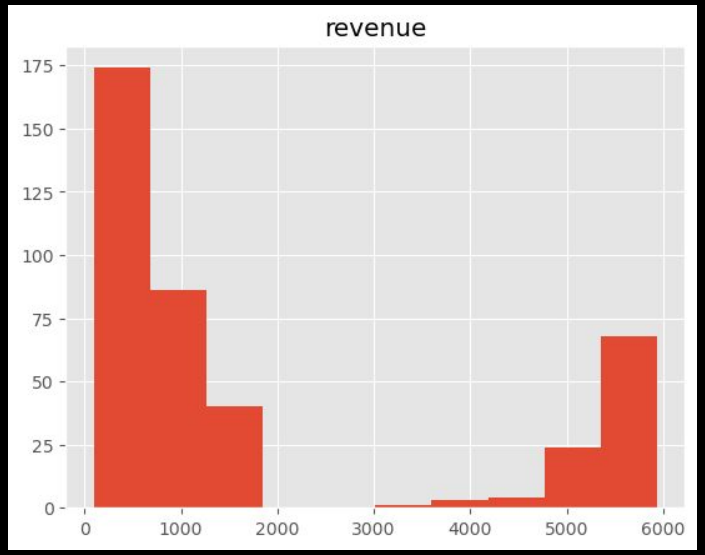


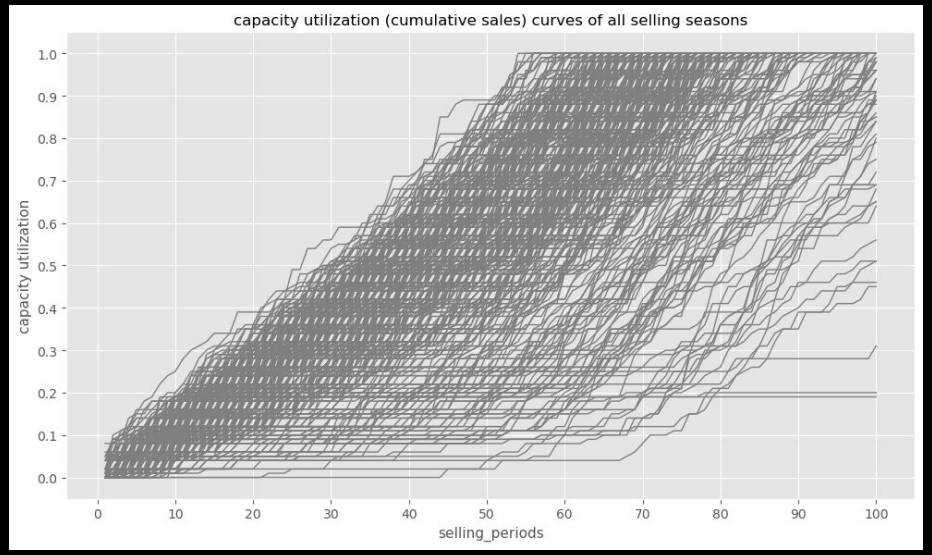
Metric computation:

```
mse = mean_squared_error(y_test, y_pred)
mae = mean_absolute_error(y_test, y_pred)
rmse = np.sqrt(mse)
r2 = r2_score(y_test, y_pred)
```

NN

- Interesting revenue distribution
- There was also decent selling out
- The good competitions are kind of curious given the poor fit of the NN

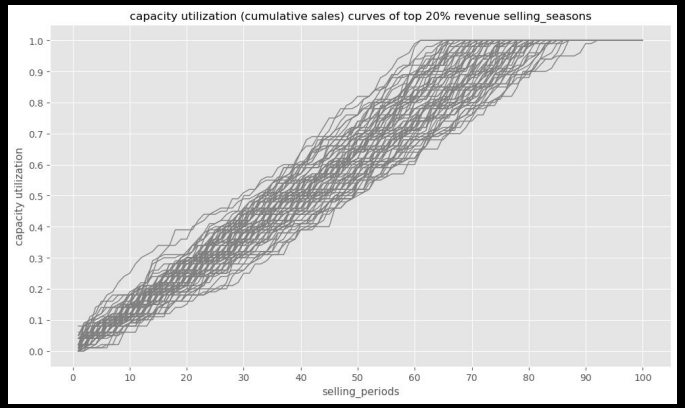


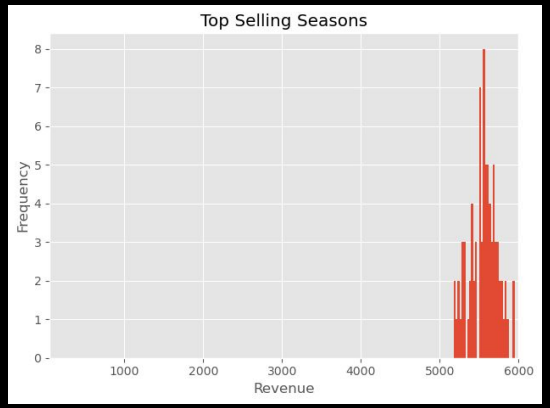


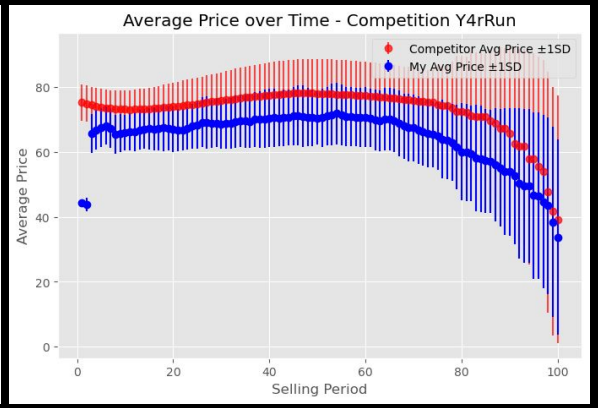
47	IntrepidBeluga	$733,328	$183,332	1.25	$1.00	$100.00	$27.90	0.455	0.885

NN

- Let's look at the scenarios where the model performed well
- Top 20% were only from this competition id: **Y4rRun**
- Notice the mean + sdev is (mostly) below the mean of the competitor price
    - From our SHAP analysis, price difference is the most important characteristic of the features we tried
- Both averages are relatively high above $50



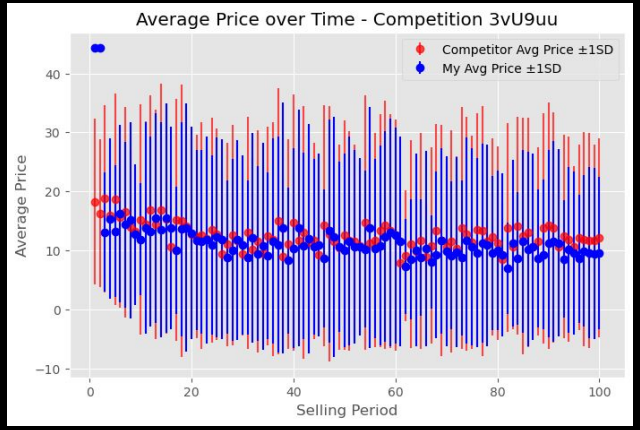


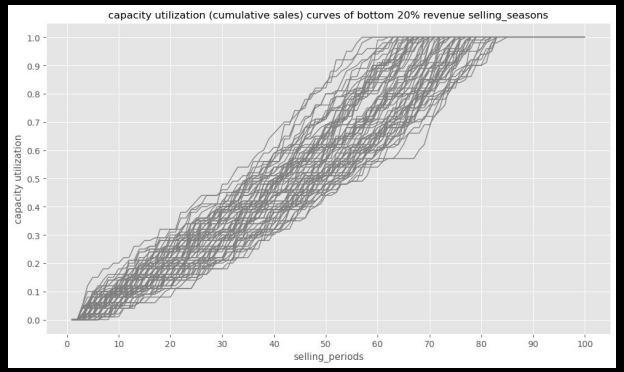


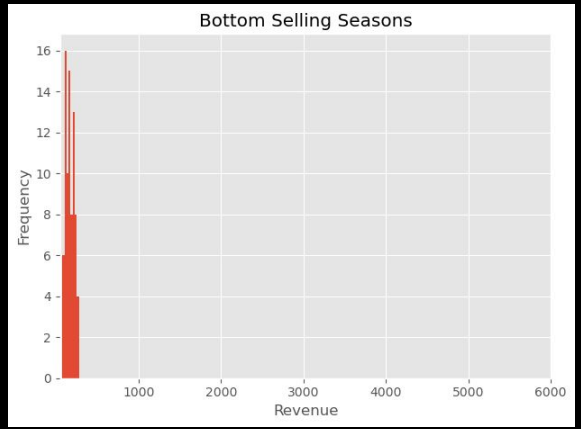
47	IntrepidBeluga	$733,328	$183,332	1.25	$1.00	$100.00	$27.90	0.455	0.885

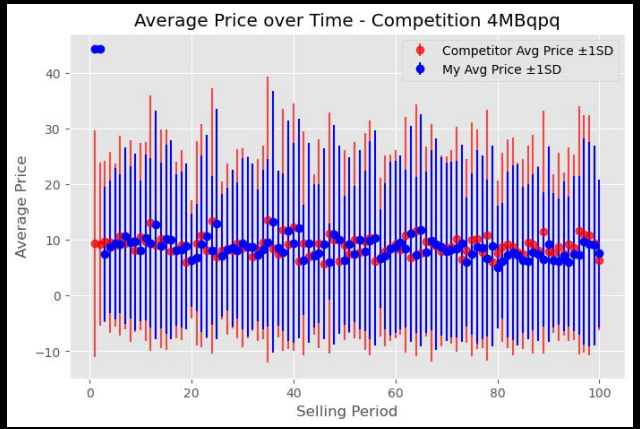
NN

- Let’s look at the scenarios where the model performed poorly
- Bottom 20% were only from these competition ids: **3vU9uu**, **4MBqpq**
- Doesn’t clearly undershoot competitor’s price
- Overall price is lower with averages around $10





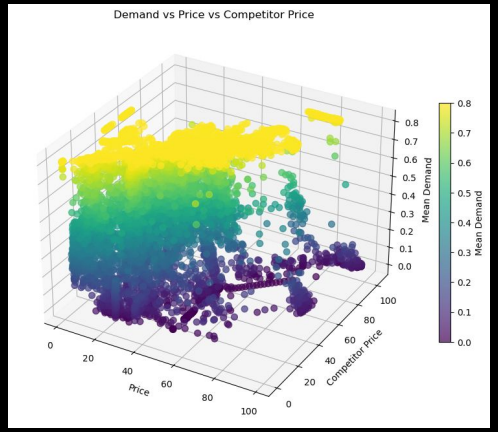


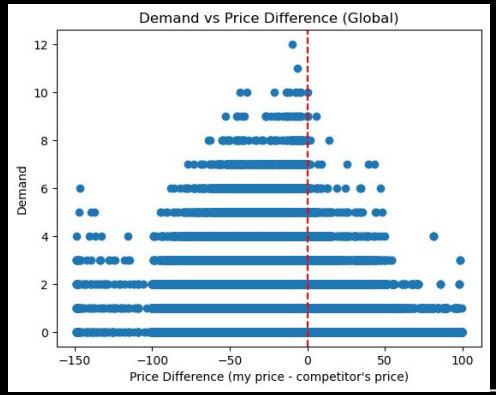


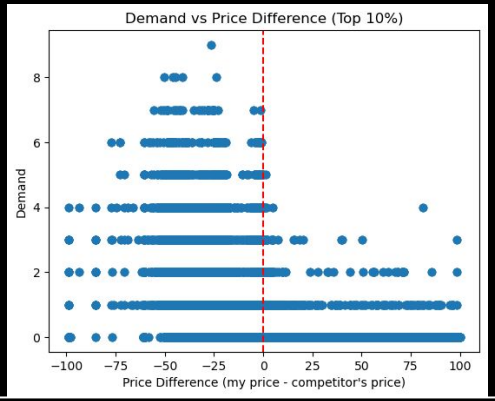
47	IntrepidBeluga	$733,328	$183,332	1.25	$1.00	$100.00	$27.90	0.455	0.885

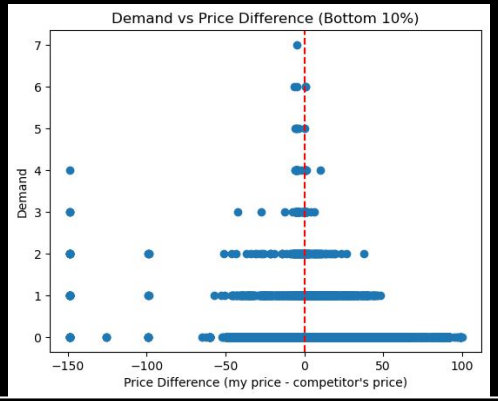
Global Data Analysis

- Conclusions:
  - **Want a high average with enough discount on competitor**
  - **Even with wrong models the pricing can still respond decently**
- Mean Demand:
  - Overall: 0.525
  - 90th percentile: 0.799
  - 10th percentile: 0.0604



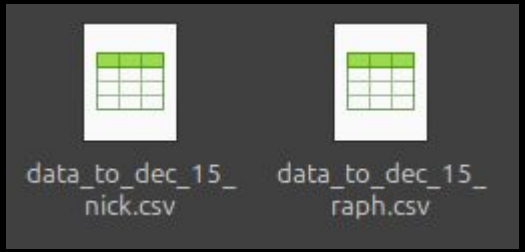






Raphael

XGBoost - Data



XGBoost - Feature Selection

- Basic

```
    new_df["abs_diff"] = abs(new_df["price_competitor"] - new_df["price"])
    new_df["price_ratio"] = new_df["price"] / new_df["price_competitor"]
    new_df["price_change"] = (
        new_df.groupby(["competition_id", "selling_season"])["price"].diff()
    )
    new_df["comp_price_change"] = (
        new_df.groupby(["competition_id", "selling_season"])["price_competitor"].diff()
    )
```

XGBoost - Feature Selection

- Statistic

```
new_df["price_roll_mean"] = (
        new_df
        .groupby(["competition_id", "selling_season"])["price"]
        .transform(lambda s: s.rolling(N, min_periods=1).mean())
    )
    new_df["comp_price_roll_mean"] = (
        new_df
        .groupby(["competition_id", "selling_season"])["price_competitor"]
        .transform(lambda s: s.rolling(N, min_periods=1).mean())
    )
```

XGBoost - Feature Selection

- Forecast

```
    new_df["comp_price_pred"] = (
        new_df
        .groupby(["competition_id", "selling_season"])["price_competitor"]
        .transform(ses_forecast)
    )
    new_df["comp_price_pred_gap"] = new_df["price_competitor"] -
new_df["comp_price_pred"]
```

XGBoost - Feature Selection

- Forecast

```
    new_df["comp_price_pred"] = (
        new_df
        .groupby(["competition_id", "selling_season"])["price_competitor"]
        .transform(ses_forecast)
    )
    new_df["comp_price_pred_gap"] = new_df["price_competitor"] -
new_df["comp_price_pred"]
```

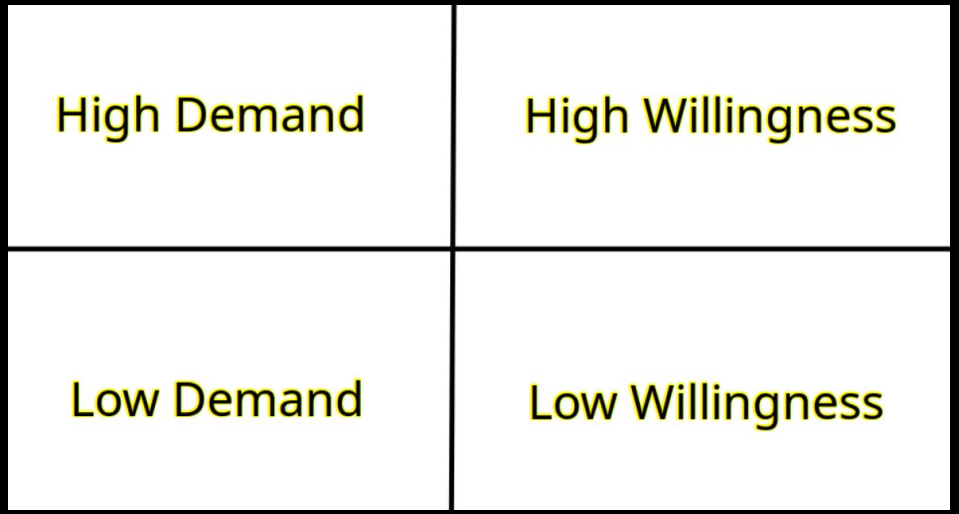
XGBoost - Feature Selection

- Seasonal

```
    new_df["season_block"] = (
        new_df
        .groupby(["competition_id", "selling_season"])["selling_period"]
        .transform(lambda s: pd.qcut(s.rank(method="first"), 5, labels=False))
    )
```

competition_id	selling_season	selling_period	season_block
A	1	1	0
A	1	20	0
A	1	21	1
A	1	40	1
A	1	41	2
...	...	...	...
A	1	100	4

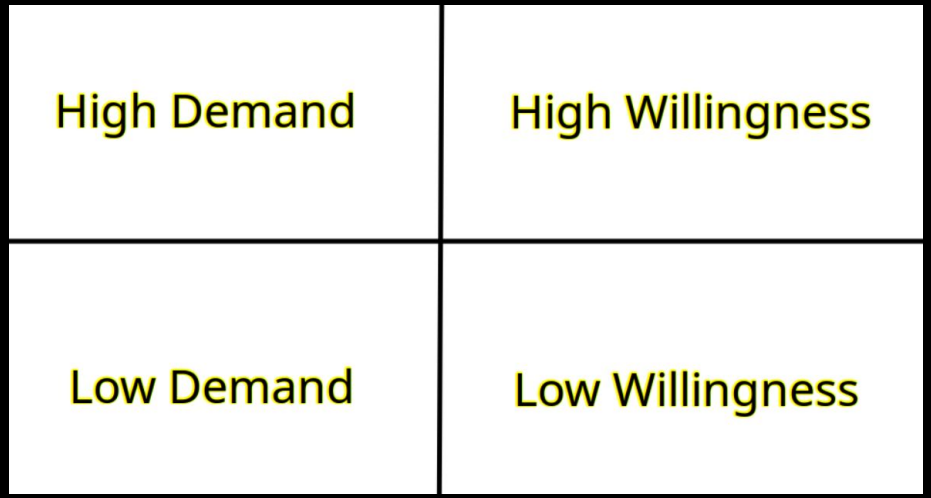
XGBoost - Competition Splits



- Demand Decision

```
season_own["own_soldout_season"] = (season_own["season_demand_sum"] >= OWN_STOCK_CAP).astype(int)

own_soldout_by_comp = (
    season_own.groupby(COMP_COL, as_index=False).agg(own_soldout_freq=("own_soldout_season", "mean"))
)
```

XGBoost - Competition Splits



- Willingness Decision

```
dfx["selling_price"] = np.where(dfx["demand"] > 0, dfx["price"], np.nan

willing_by_comp = (
   dfx.groupby(COMP_COL, as_index=False).agg(mean_selling_price=("selling_price", "mean"))
)
```

XGBoost - Competition Splits

XGBoost - Feature Selection

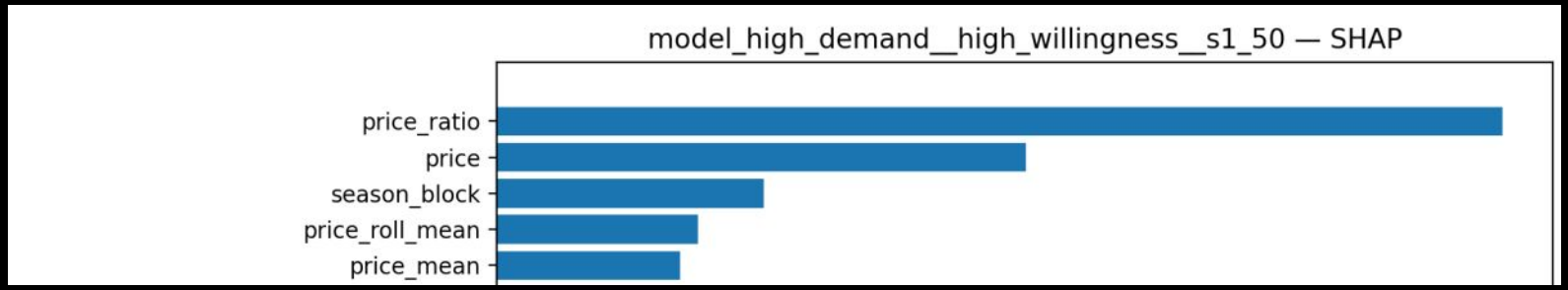
- Late Season Selection

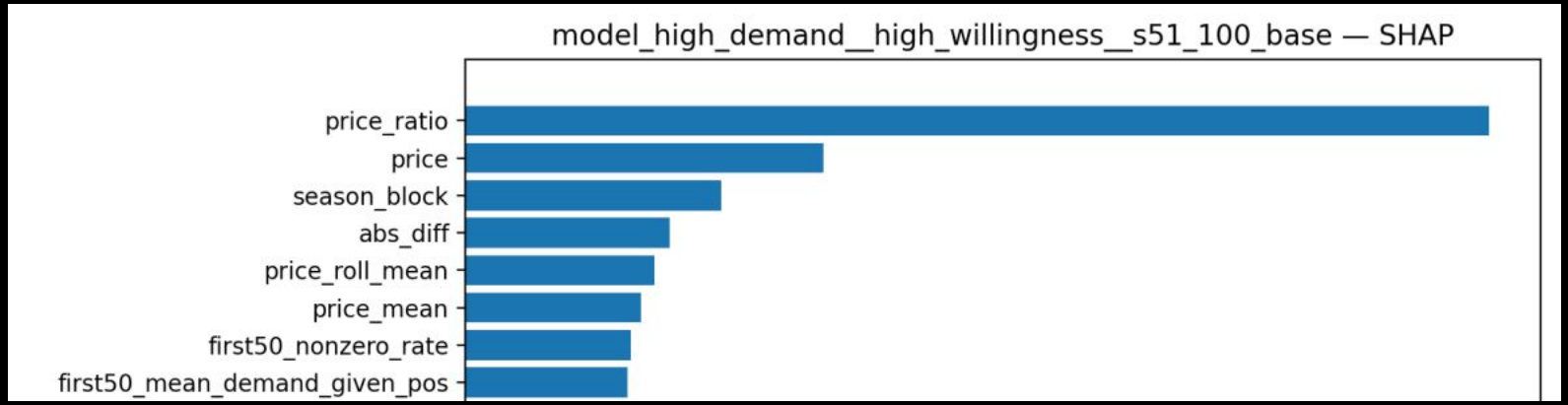
```
first50 = new_df.loc[new_df["selling_season"] <= SEASON_CUTOFF].copy()
first50["d_pos"]   = (first50["demand"] > 0).astype(int)
first50["d_spike"] = (first50["demand"] >= 2).astype(int)
comp_sparse = (
    first50
    .groupby("competition_id", as_index=False)
    .agg(
        first50_nonzero_rate=("d_pos", "mean"),
        first50_spike_rate=("d_spike", "mean"),
        first50_mean_demand=("demand", "mean"),
        first50_max_demand=("demand", "max"),
    )
)
```

	A	B	C	D	E
1	competition_id	selling_season	selling_period	demand	first50_nonzero_rate
2	2GxWsg	51	2	0	0.663844199830652
3	2GxWsg	51	3	0	0.663844199830652
4	2GxWsg	51	4	1	0.663844199830652
5	2GxWsg	51	5	1	0.663844199830652

XGBoost - Feature Selection - Result

- Feature selection



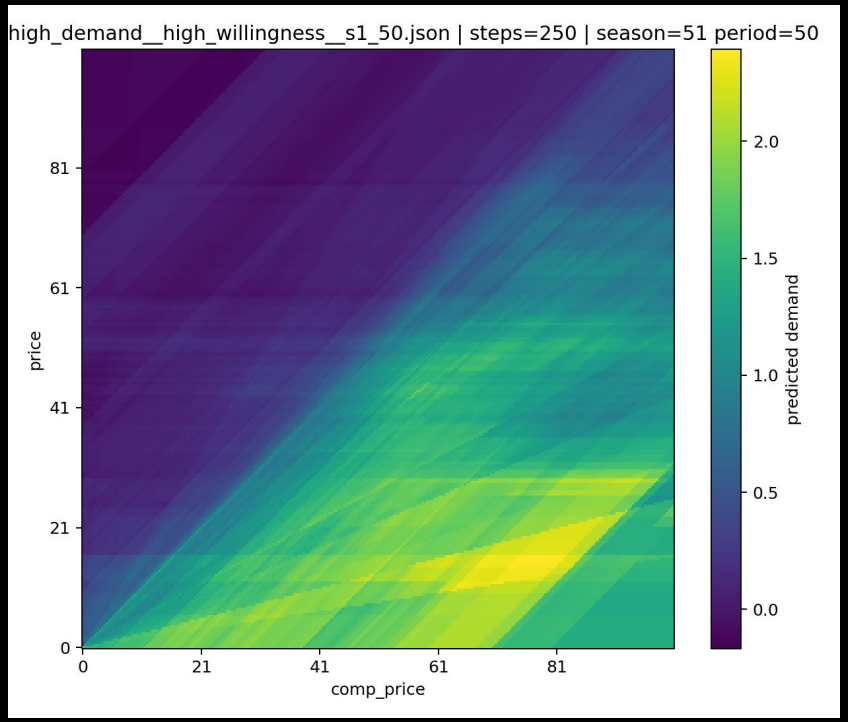


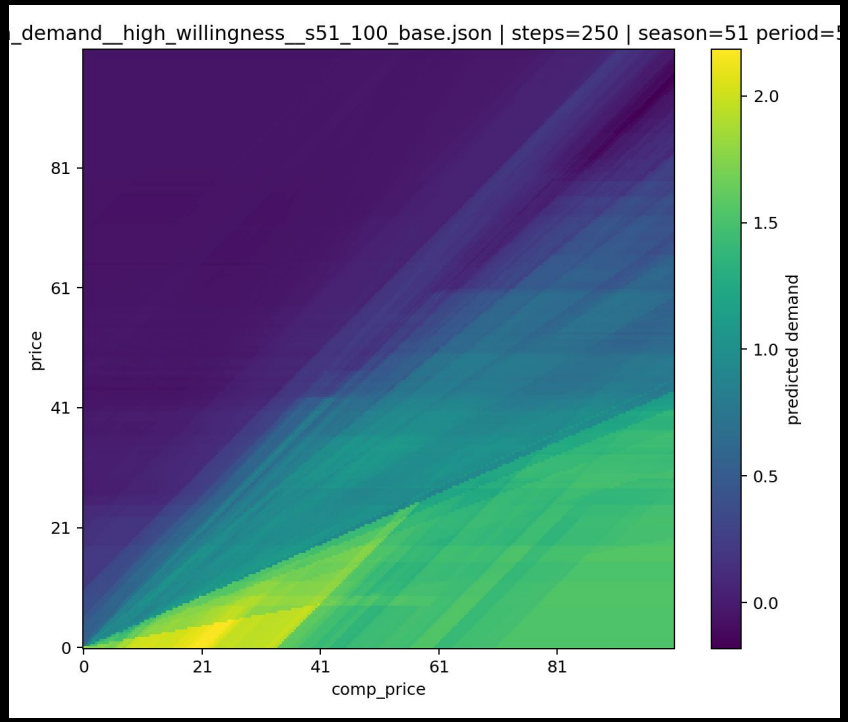
XGBoost - Feature Selection - Used

- Engineered Features Used

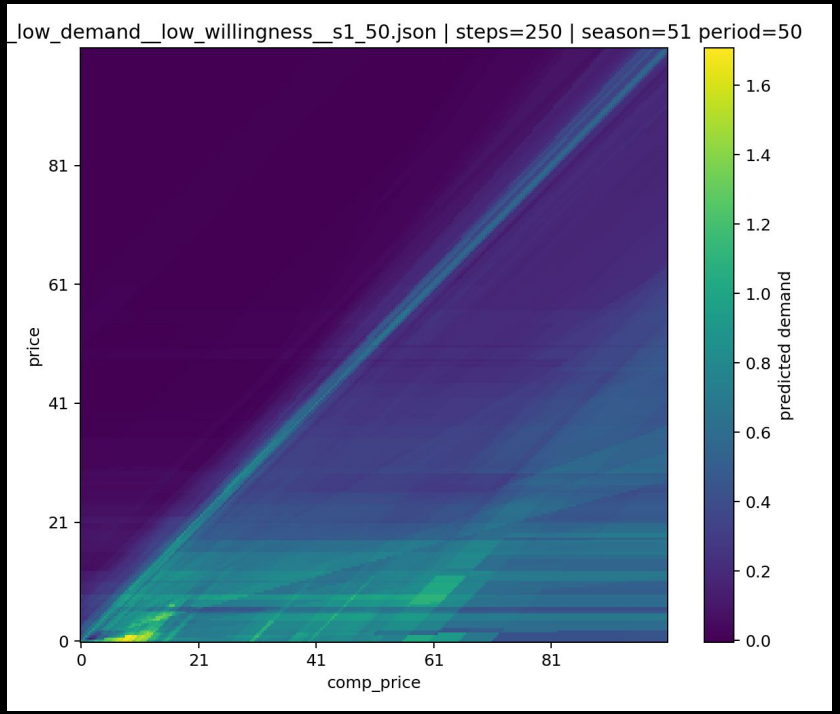
```
FEATURES_ALL = ["price_ratio", "price", "price_roll_mean", "season_block", "abs_diff"]
FEATURES_FIRST50 = FEATURES_ALL
FEATURES_51_100 = FEATURES_ALL + ["first50_mean_demand_given_pos", "first50_nonzero_rate"]
```

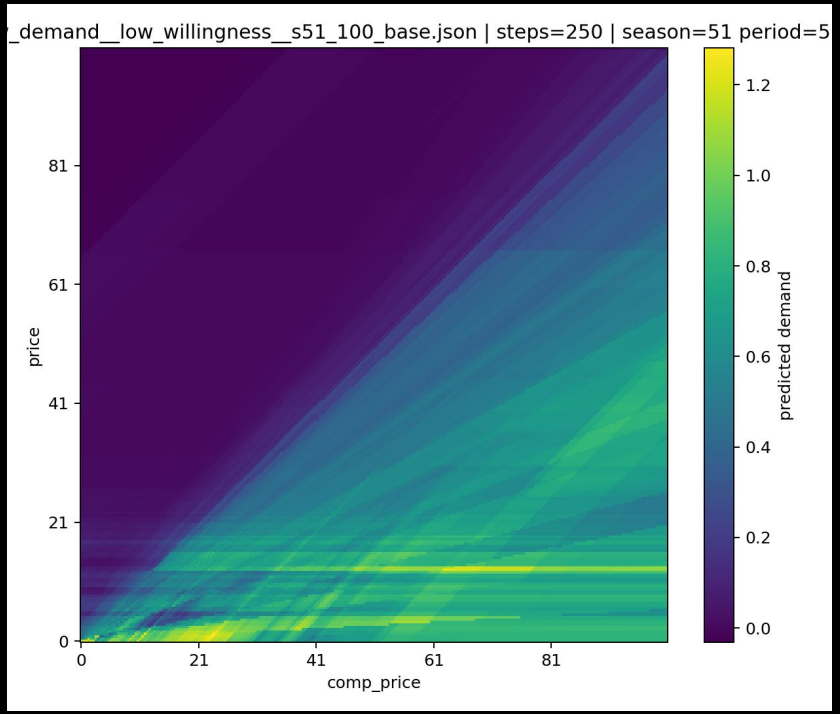
Model - Prices to Use



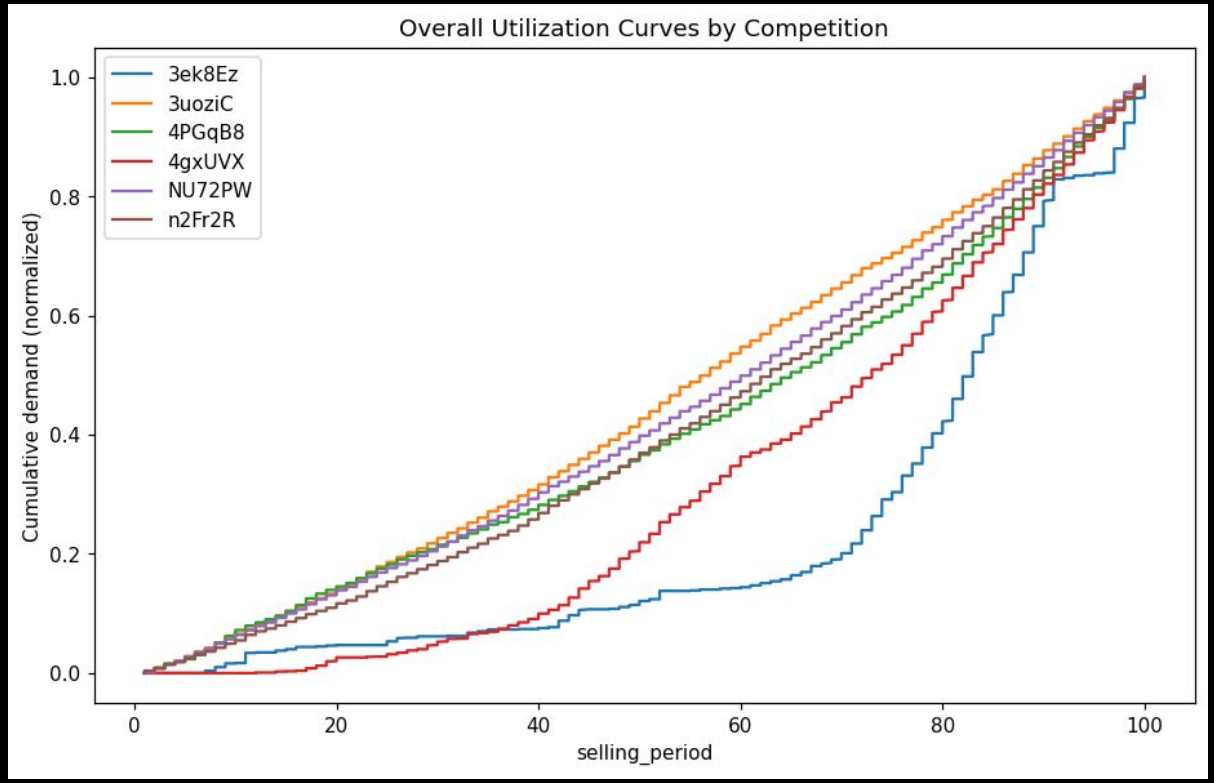


Model - Prices to Use

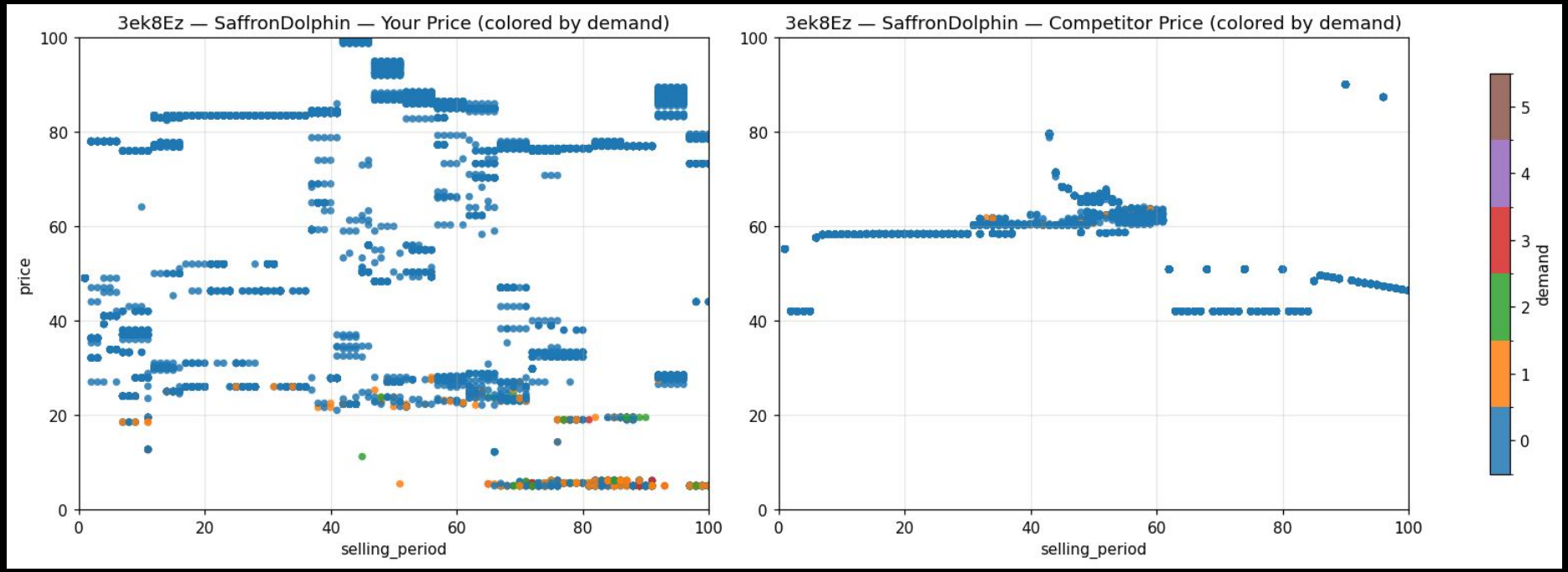




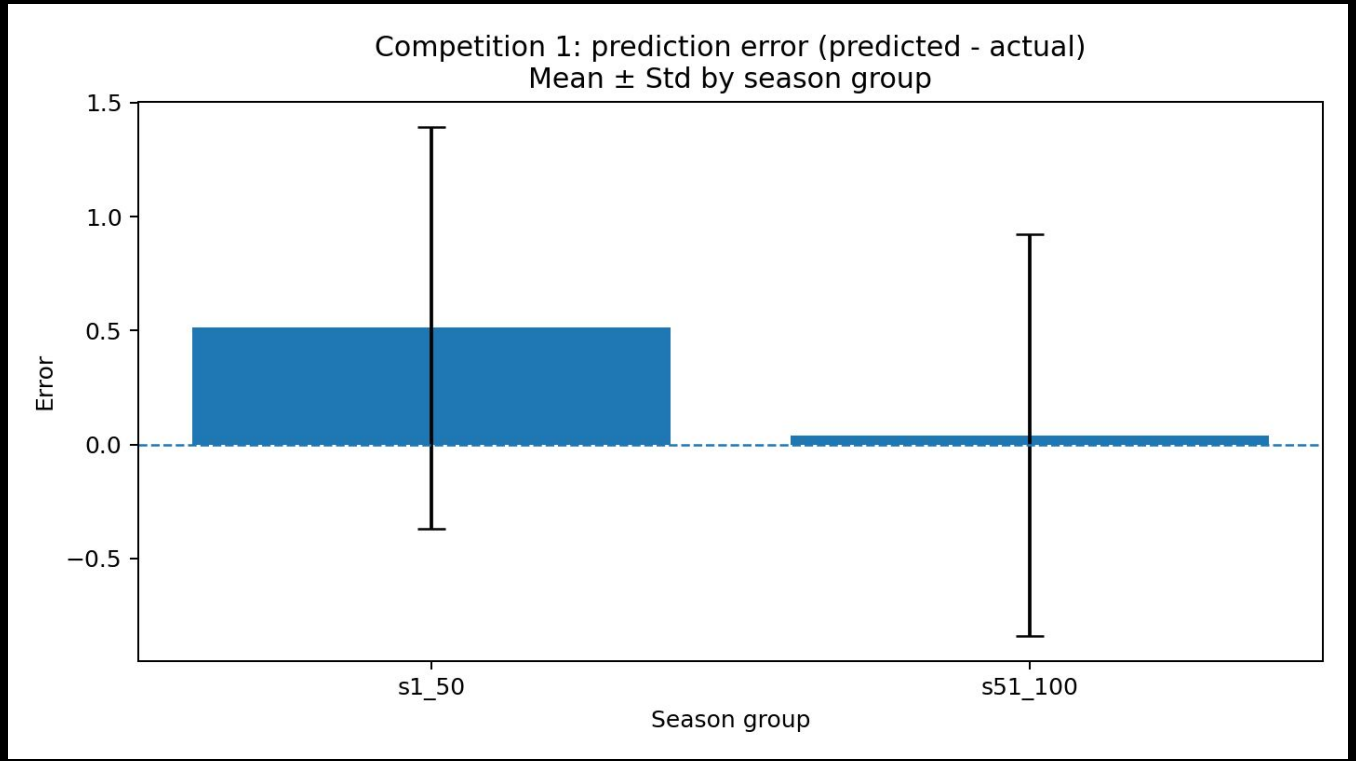
Model - In Action - Overall



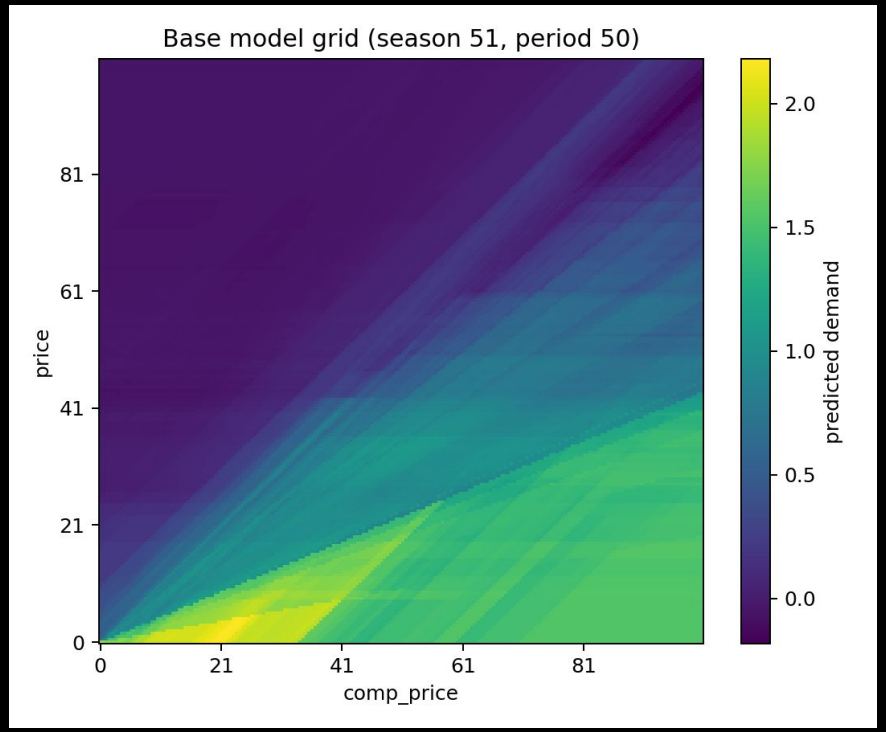
Model - In Action - All Prices used

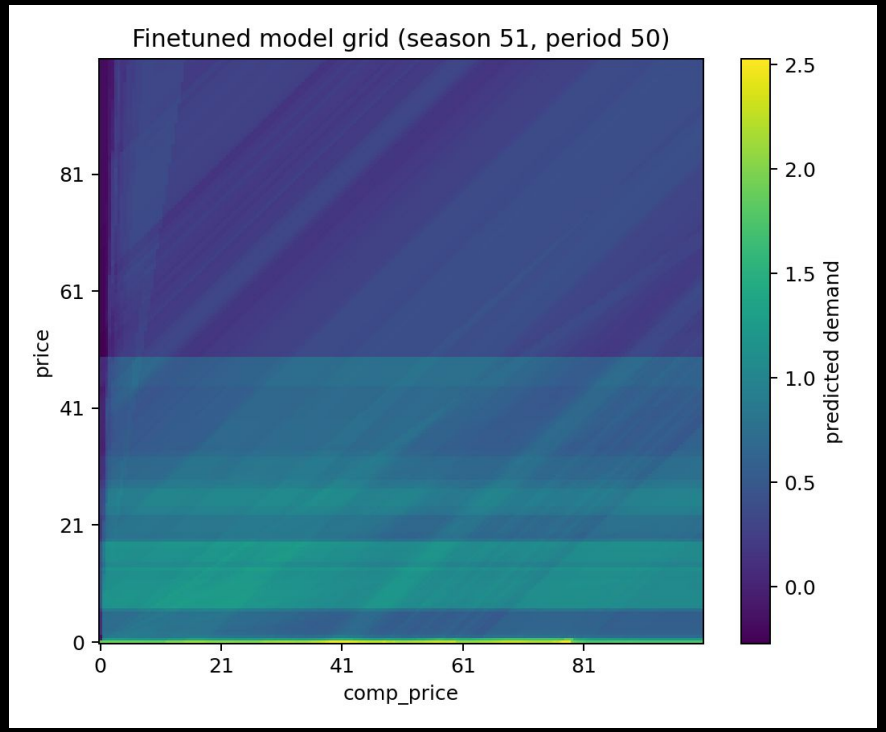


Model - Accuracy



Model - Accuracy





All models are wrong, but some are useful

1976, George Box