ChatGPT

Predictive Model for Theoretical Food Cost & Usage in Restaurants

Data Inputs and Schema

A robust predictive model needs well-structured data on recipes, sales, ingredients, and purchasing. Key inputs include:

- **Menu Item Recipes & Yields:** Each menu item is mapped to a Bill of Materials (BOM) listing its ingredient components and quantities. Include yield factors (trim loss, cooking shrinkage) for each ingredient to translate **edible portion (EP)** to **as-purchased (AP)** quantity [1]. For example, if a soup uses 1 kg of peeled potatoes (EP) and peeling yields 95%, the recipe should record that ~1.05 kg raw potatoes (AP) are required [2]. Each recipe entry might have fields: `MenuItemID, IngredientID, Quantity (EP), Unit, Yield%`.

- **POS Sales Data:** Historical item-level sales from the Point-of-Sale system, ideally timestamped and tagged by date and daypart (e.g. lunch, dinner). Fields: `Date, MenuItemID, QuantitySold, Daypart`. This provides the volume of each item sold, which is crucial for mapping to ingredient usage [3]. Ensure sales data is granular enough (daily or shift-level) to capture patterns.

- **Ingredient Master Data:** A catalog of all ingredients with purchasing and conversion details. Fields might include: `IngredientID, IngredientName, Vendor, PurchaseUnitSize, PurchaseUnitName, ConversionToBaseUnit, LastPrice`. For example, “Chicken Breast” might have a purchase unit of *case (40 lbs)* and conversion *1 case = 40 lb*. This allows converting required ingredient amounts into vendor order units [4].

- **Par Levels & Reorder Points:** Pre-determined stock targets for each ingredient. Par level is the ideal on-hand quantity at the start of a vendor order cycle (often calculated as average usage over the cycle plus safety stock) [5] [6]. Reorder point (threshold) is the minimum allowable on-hand level before triggering a reorder. These can be defined per ingredient per vendor cycle (e.g. weekly par in lbs, with a reorder threshold in lbs). Fields: `IngredientID, ParLevel, ReorderPoint, VendorOrderCycle`. For instance, if a vendor delivers weekly, an ingredient’s par might be 70 kg (to cover 7 days at 10 kg/day + safety stock) [7], and the reorder threshold might be 20 kg (when stock falls to this level, order up to par).

- **Vendor Order Schedule:** Information on ordering cycles and lead times. This can be a simple configuration per vendor (e.g. deliveries every Monday, 2-day lead time). It’s used to align the forecast horizon with order cycles and to compute how long stock must last. For example, LeadTime=3 days for a vendor implies par is based on 3 days of usage [5].

- **Historical Deliveries (Optional):** Records of past orders received (by date and quantity) for each ingredient. This helps initialize and update the theoretical on-hand inventory. Fields: `DeliveryDate, IngredientID, QuantityReceived`. If physical inventory counts are not taken regularly, this acts as the "input" for stock levels, which the model will decrement by usage.

- **Pricing Data:** Latest cost per ingredient, either from the last invoice or a rolling average price. This is needed for the cost calculations (e.g. last known price per case or per pound). It may be included in the Ingredient Master or a separate price history table (`IngredientID, Price, PriceDate`).

By ensuring these inputs are available and updated (especially recipes and prices), the model can accurately compute theoretical usage and cost [3].

Sales Forecasting Module

Accurate forecasting of menu item sales is the foundation of the model. The goal is to project future sales over the next vendor order cycle (e.g. the upcoming week) using historical trends. Key steps in this module:

1. **Historical Averages by Day/Daypart:** Compute average sales for each menu item by day of week and daypart over the last *N* weeks. For example, calculate that "Item X" sells on average 50 units on Fridays (dinner) vs 20 units on Mondays. Smoothing over multiple weeks (N can be tuned, e.g. 4 or 8 weeks) helps iron out anomalies [8].

2. **Trend and Seasonality Adjustment:** Adjust the baseline with known trends. If sales are growing or seasonal patterns exist, apply a factor. For instance, if the last 4 Fridays show a 5% week-over-week growth for an item, forecast the coming Friday slightly higher. Incorporate seasonal indices (e.g. higher sales in summer) or upcoming local events and promotions that could affect demand [9] [10].

3. **Forecast Computation:** For the target period (e.g. next 7 days), generate a forecast for each menu item per daypart. One simple approach is: **Projected Sales = Average Daily Sales × (# of days or specific pattern for period) × Adjustment Factors** [11]. For example, if item X averages 100 sales per week and an upcoming holiday is expected to boost sales by 10%, then forecast ~110 sales for that week. A more granular approach is to forecast each day separately (Mon, Tue, …) using the corresponding day's historical average and adjustments.

4. **Daypart Consideration:** Ensure the forecast respects daypart patterns. If the model needs to simulate usage intra-day (e.g. lunch vs dinner prep), keep separate forecasts for each shift. This level of detail can improve ordering for items used only in certain dayparts (e.g. breakfast ingredients).

5. **Output – Forecasted Item Sales:** The output is a table of expected sales counts for each menu item for each day (or aggregated over the order cycle). For instance: *"Grilled Salmon" – next week forecast: Mon 30, Tue 25, Wed 28, … (total 200)*.

Tuning: The forecasting window *N* weeks is configurable – a shorter window makes the forecast react faster to recent changes, while a longer window smooths out volatility. You can also choose forecasting methods ranging from simple moving averages to more advanced time-series models (exponential smoothing, ARIMA, or machine learning) if data is abundant. The baseline method should be easily adjustable based on periodic forecast accuracy reviews.

Theoretical Usage Calculation (Recipe Explosion)

Using the forecasted sales, the model computes theoretical ingredient usage via a “recipe explosion” (similar to a manufacturing BOM explosion):

1. **Map Menu Items to Ingredient Needs:** For each menu item in the forecast, retrieve its recipe ingredients and quantities (from the Recipes data). Multiply the forecasted quantity of that menu item by each ingredient’s recipe quantity to get the total **edible portion** usage of each ingredient. For example, if 50 burgers are forecast and each burger needs 0.5 lbs of beef (EP), that’s 25 lbs EP of beef required.

2. **Apply Yield Factors:** Convert edible portion to actual required amount using yield percentages. If an ingredient has a yield less than 100% (trim/cook loss), the model inflates the requirement so that the usable portion is met [1]. Using the above example, if ground beef yields 80% after cooking, to get 25 lbs EP, the kitchen needs ~31.25 lbs raw (25 ÷ 0.8). The model performs this for each ingredient: `Required_AP = Required_EP / Yield%`. This ensures we account for typical losses and get a *theoretical usage* that aligns with purchasing units.

3. **Aggregate by Ingredient:** Sum the usage of each ingredient across all menu items. This yields the **theoretical ingredient usage for the entire forecast period**. For instance, if beef is used in two different dishes, their requirements are added to get total beef needed. At this stage, the model has a list like: *Ingredient A – 10.5 kg, Ingredient B – 3 cases, Ingredient C – 2.3 lb*, etc., representing how much of each ingredient should be consumed over the upcoming cycle, given the sales projection.

4. **Time Phasing (Optional):** If needed, distribute the usage by day (or by delivery date) to know how inventory levels deplete day-by-day. This is useful for checking if an ingredient might run out before the cycle ends (especially if usage is front-loaded). In most cases, the total over the cycle is used for ordering, but critical or fast-depleting items might warrant intra-cycle checks.

The result is a **theoretical usage report** – essentially an ideal usage forecast – that assumes perfect portioning and no waste [12]. This can be compared later to actual usage or used directly for ordering decisions.

Inventory Simulation and Par Level Logic

With forecasted usage in hand, the model simulates inventory levels and decides on reorders without requiring a physical count. The core idea is to use **theoretical on-hand** stock (based on previous deliveries minus usage) to determine if and what to order. The process:

1. **Initialize Theoretical On-Hand:** Start with the last known on-hand quantity for each ingredient. Since no new physical counts are taken, this typically comes from the prior cycle’s calculations. For example, after the last order and consumption, the system might estimate you have 5 cases of

tomatoes left. If starting fresh, one could input a beginning inventory count or assume par level as a starting point.

2. **Subtract Forecasted Usage:** Compute the projected on-hand at the end of the cycle as `Ending_OnHand = Current_OnHand + Scheduled_Deliveries - Forecasted_Usage`. In many cases, you have no scheduled deliveries until the next order, so effectively `Ending_OnHand ≈ Current_OnHand - Forecasted_Usage`. This tells you how low each ingredient's stock will get if you **do not order** more.

3. **Check Reorder Thresholds:** For each ingredient, compare the projected ending on-hand to the predefined **reorder point**. If the projected stock will drop below the reorder threshold (or become negative, indicating a stockout), then that ingredient requires an order this cycle. The threshold serves as a buffer so that you're not ordering at the exact moment of running out [5] [7]. For example, if onions have a reorder point of 10 lbs and the model predicts ending on-hand would be 5 lbs, it's a trigger to reorder. If an ingredient is forecasted to remain above its threshold (e.g., a slow-moving spice might still have plenty in stock), the model can skip ordering it this round.

4. **Calculate Order Quantity to Par:** For each triggered ingredient, determine how much to order. The target is to raise the theoretical on-hand up to the **par level**. The formula is typically `OrderQty = ParLevel - Projected_OnHand_at_OrderTime`. Here, *Projected_OnHand_at_OrderTime* means the amount you expect to have when the order would arrive (consider current on-hand minus usage until delivery day). In a weekly cycle where orders arrive at the start of the cycle, this is basically current stock. In other cases, if some days of usage will occur before delivery, subtract that out. For simplicity, many restaurants assume ordering happens right after the cycle, so effectively `OrderQty ≈ ParLevel - Current_OnHand`. Continuing the onion example: if par is 50 lbs and current theoretical on-hand is 5 lbs (with delivery imminent), the model would recommend 45 lbs to reach par.

5. **Prevent Negative or Excess Orders:** The model should floor orders at 0 (never suggest negative). If an ingredient's current on-hand already exceeds par (perhaps due to overstock in a prior cycle), no order is needed. If an ingredient is right at threshold but small usage will drop it, the model can either top it up or wait if it's still above par – this logic can be tuned based on how strictly the operator wants to adhere to par levels.

6. **Incorporate Vendor Order Cycle:** Group the order recommendations by vendor or delivery schedule. Each vendor order will include only the items that vendor supplies, and respect any minimum order quantities or cycles. The cycle length (weekly, bi-weekly) is essentially the forecast horizon used above. If different ingredients have different vendors with different schedules, run the forecast and ordering process per vendor schedule.

The outcome is a list of **suggested order quantities per ingredient** that will bring stock to par and avoid dropping below safe levels before the next delivery. All this is achieved without a physical inventory count – relying on the running theoretical inventory. It is crucial to periodically validate the theoretical model against reality, since errors can accumulate without physical counts. (We discuss tuning adjustments for this below.)

Vendor-Specific Purchase Unit Conversion

Vendors often sell products in specific pack sizes or units (cases, crates, gallons, etc.). The model must convert the required quantities (usually in kitchen units like pounds or pieces) into vendor order units:

- **Determine Purchase Unit Needs:** Using each ingredient's purchase unit information, convert the `OrderQty` (from the previous step, typically in base units like lbs or each) into the number of vendor units to order. This often means rounding **up** to the nearest whole unit, since you usually can't buy fractions of a case. For example, if the model determines you need 8 lbs of a product and it's sold by the case of 6 lbs, then `8 ÷ 6 = 1.33` cases, which rounds up to **2 cases** to ensure enough supply. The model would output "2 cases" as the order quantity for that item.

- **Adjust for Inner Packs or Multiples:** Some vendors have inner pack levels (e.g., a case contains 4 bags of 5 lb each, and you can perhaps purchase half cases). The schema should capture if partial cases are allowed or if ordering must be in full-case increments. The conversion logic will then round to the allowed increment. For instance, if half-cases are permitted, needing 8 lbs in the above example might translate to **1 case + 1 extra bag** (if one bag is 5 lbs, total 10 lbs ordered).

- **Output Formatting:** Present the order recommendations in vendor terms, including any product codes or names. For example: *Vendor A Order:* 2 cases of Chicken Breast, 1 case of Tomato (25 lb case), 3 bottles of Olive Oil (1 gal bottle). Each line can include the conversion detail (e.g., "2 cases (≈12 lb required)"). This ties directly into generating a purchase order or order guide.

This step ensures the theoretical requirements are actionable in the real world of supplier ordering [4]. It bridges the gap between kitchen units and vendor units, so the team knows exactly how much to order from each supplier.

Cost Prediction Module

To estimate theoretical food cost for the period, the model multiplies the forecasted ingredient usage by their costs:

- **Ingredient Cost Calculation:** For each ingredient, take the **theoretical usage** (forecast amount to be used, in purchase units) and multiply by the unit cost (from the pricing data). Use the last known price or an average price if prices fluctuate. For example, if the model predicts 30 lbs of cheese will be used and cheese costs \$5.00 per lb, the theoretical cost for cheese is \$150. Sum up all ingredients to get the **total theoretical cost of goods** for the period [12].

- **Menu Item or Category Cost (Optional):** The model can also break down cost by menu item by summing the cost of each item's ingredients times quantity sold. This yields insight into which menu items incur the most cost. However, since the focus is on ingredient-level ordering, the primary output is usually ingredient cost.

- **Theoretical Food Cost Percentage:** If an estimate of sales revenue for the period is available (from the sales forecast multiplied by menu prices), the model can compute the theoretical food cost percentage. **Theoretical Cost % = (Theoretical COGS / Forecasted Sales) × 100**. This indicates what the food cost *should* be as a percentage of sales if everything goes according to plan (no waste, correct portions). For instance, if forecast sales are \$50,000 and theoretical cost is \$15,000, the theoretical food cost % is 30%. This is a benchmark to compare against actual results later [1].

- **Cost Report Output:** A **theoretical cost summary** might list each ingredient's expected usage and cost, plus total cost. Example:

- Chicken Breast: 40 lbs × \$2.50/lb = \$100.00
- Tomatoes: 1 case (25 lb) × \$30.00/case = \$30.00
- … (list all ingredients)
- **Total Theoretical Food Cost = \$X** for the week, which at \$Y projected sales yields **Z%** food cost.

This cost model not only projects spending but also can drive budgeting and pricing decisions. It's important that the recipe costs and prices are kept up-to-date; the accuracy of theoretical cost depends on reflecting current market prices for ingredients [13].

Step-by-Step Usage & Order Simulation Logic

Bringing it all together, below is a stepwise summary of how the model operates each cycle (e.g. weekly):

1. **Pull Data for Period:** Load the past N weeks of POS sales data and latest recipe definitions, ingredient stock levels, and prices.

2. **Forecast Sales:** For each menu item, calculate projected sales for the next cycle (e.g. next 7 days) using historical averages and adjustments (account for daypart patterns, seasonality, events).

3. **Translate Sales to Ingredient Usage:** For each menu item's forecast, multiply by its recipe ingredient quantities. Adjust for yield % to get the **as-purchased** quantity needed per ingredient. Aggregate across all items to get total forecasted usage per ingredient for the cycle [4].

4. **Calculate Theoretical On-Hand:** For each ingredient, take the current theoretical on-hand (carry-over from last cycle's ending balance) and subtract the forecasted usage from step 3. This yields the projected ending inventory if no new supply is received.

5. **Compare to Par/Threshold:** Check if projected ending inventory is below the safety threshold or if current inventory is below par. If yes for a given ingredient, plan to place an order. Determine the needed order quantity = (Par level – current on-hand + expected usage before delivery) *if delivery is not immediate*. In many cases this simplifies to Par – current on-hand [5].

6. **Convert to Order Units:** Convert each needed ingredient order quantity into the vendor's purchase units (round up to case/pack sizes). Prepare a list of items to order with quantities in supplier terms.

7. **Output Recommendations:** Generate the **Suggested Order Guide** for the upcoming delivery. This includes each product, the recommended order quantity, and possibly a note like "(to reach par of X units)". Separately, generate a **Theoretical Usage & Cost Report** summarizing how much of each ingredient is expected to be used and the associated cost, as well as the total cost projection.

8. **Update Theoretical Inventory:** When the order is "placed" (or when the new stock arrives), update the theoretical on-hand by adding the ordered quantities. This resets the starting stock for the next cycle's calculations. No physical count is done; instead, the model assumes the delivered amounts and calculated usage are accurate. For example, if 45 lbs of onions were ordered to raise stock to par 50 lbs (assuming 5 lbs left), the new theoretical on-hand for onions becomes 50 lbs at the start of the cycle.

9. **Monitor and Refine:** As the cycle progresses, actual POS sales are tracked. The model can optionally adjust mid-cycle if sales deviate significantly (e.g., a surge in sales might trigger an earlier reorder). At cycle end, compare predicted ending stock vs. what the model shows after subtracting actual sales (the model's "book inventory"). Significant variances might indicate issues like waste or recipe inaccuracy, prompting refinement.

Each cycle, the above loop repeats, allowing continuous planning without on-hand counts. This periodic simulation aligns orders with actual consumption patterns, ensuring the kitchen has what it needs without overstocking.

Example Output Formats

To illustrate, here are example outputs that a restaurant inventory app implementing this model might generate:

1. Ingredient Usage Forecast (Weekly): A report listing each ingredient and how much will be used theoretically in the next cycle. For example:

```
Ingredient           Forecast Usage (next 7 days)   Unit
-----------------------------------------------------------
Chicken Breast       40 lb                          (raw, boneless)
Roma Tomatoes        18 lb                          (usable weight)
Cheddar Cheese       5 kg                           (shredded)
Burger Buns          200 each
...
```

This can be derived from the recipe explosion. It helps kitchen staff anticipate prep needs and also serves as the basis for ordering.

2. Suggested Order Guide: Grouped by vendor, the quantities to order to meet par levels:

```
Vendor: FreshFoods Co. (Weekly delivery every Monday)
Product                 Current On-Hand   Par Level   Order Qty   UOM   Notes
-------------------------------------------------------------------------------
Chicken Breast, boneless       10 lb         50 lb      1 case     40 lb case
(covers 40 lb need)
Roma Tomatoes                  5 lb          30 lb      1 case     20 lb case
(+5 lb safety stock)
```

```
Onions (Yellow, jumbo)          8 lb            40 lb       1 bag       50 lb bag
(will have extra stock)
Cheddar Cheese, shredded        2 kg            5 kg        1 bag       5 kg bag
(par to cover 1 week)
...
```

In this example, Chicken Breast needed 40 lb (forecast usage) and par is 50 lb; with 10 lb on hand, the model orders 40 lb (1 case) to reach par. Tomatoes are sold by 20 lb case; 1 case brings stock from 5 lb to 25 lb (slightly below the 30 lb par, but maybe sufficient given a safety margin). Onions show ordering a full 50 lb bag even though that exceeds par – this could happen due to vendor pack size (the model decided to slightly overstock rather than risk a shortage). Each line shows the reasoning (cover forecast + safety).

3. Theoretical Food Cost Summary:

```
Period: Next Week (Nov 6–Nov 12)
Projected Sales (7 days): $50,000
Total Theoretical Usage Cost: $14,500
Theoretical Food Cost %: 29.0%

Breakdown by Category:
- Proteins: $8,900
- Produce: $3,100
- Dry Goods: $1,200
- Dairy: $1,300
(See detailed ingredient cost report for line-by-line costs)
```

This summary gives management a sense of expected cost and margin. It could be accompanied by a detailed ingredient cost list (not shown here) enumerating each ingredient's usage and cost (e.g., Chicken Breast – 40 lb at $2.50 = $100; Tomatoes – 1 case at $18 = $18; etc.).

4. Menu Item Cost Projection (optional):

```
Menu Item              Forecast Sold   Item Cost   Total Cost
--------------------------------------------------------------
Grilled Chicken Salad    100           $3.20       $320
Cheeseburger             200           $2.10       $420
Fish Tacos               150           $1.50       $225
...
```

This ties the theoretical usage back to menu items, showing how each contributes to cost. While not required for ordering, it's useful for menu engineering and verifying that the recipe costs align with the forecast.

All outputs are in a clear, tabular format for easy reading. The actual app could export these as spreadsheets or display them on a dashboard. Importantly, any **embedded citations or references** in the model's explanation (like those seen here) are for documentation and can be omitted or translated into internal comments in an implementation.

Tuning and Configuration Suggestions

No predictive model works optimally without tuning. Here are areas to refine for better accuracy and efficiency:

- **Forecast Window & Method:** Adjust the number of weeks used for averaging in the sales forecast. If the restaurant business is stable, a longer window (8-12 weeks) smooths out noise; if it's rapidly changing or seasonal, a shorter window (4 weeks) or inclusion of same-period-last-year data might improve accuracy. Consider using a rolling forecast updated weekly [11], and monitor forecast error by comparing predicted vs actual sales to fine-tune the approach.

- **Daypart and Trend Adjustments:** If the model consistently under- or over-forecasts certain days or dayparts, introduce correction factors. For example, if Friday dinners are repeatedly busier than forecast, increase the Friday dinner multiplier. Incorporating external data (weather, local events, reservations on the books) as needed can refine the forecast.

- **Yield and Waste Overrides: Theoretical usage assumes perfect yields** – which is rarely true. To prevent the model from drifting due to unaccounted waste, introduce a "yield override" or waste factor. For instance, if you observe that theoretical on-hand vs actual on-hand (when occasionally checked) diverges by ~5% for produce, you might globally reduce yield percentages or add a 5% extra usage to account for waste/spoilage [14]. Similarly, if a particular ingredient always runs out despite the model (indicating hidden usage or prep loss), adjust its yield or safety stock upward. This effectively starts **accounting for waste and shrinkage in the theoretical model** to improve its accuracy over time [14].

- **Safety Stock Levels:** Revisit safety stock (the buffer in par calculations) based on variability. Ingredients with highly variable sales or long lead times may need larger safety stock. The reorder threshold can be tuned accordingly. If you notice frequent last-minute 86'd items (stockouts), it's a sign to raise par or safety stock for those ingredients.

- **Cycle Length:** If the vendor order cycle can be adjusted (e.g., switching from weekly to twice a week orders), test shorter cycles for volatile items to reduce stockout risk. The model can accommodate this by simply changing the forecast horizon and par calculations (e.g., 3-4 day forecast instead of 7).

- **Continuous Learning:** Implement a feedback loop where after each cycle, you compare **predicted vs actual** ingredient usage (actual can be approximated by purchases plus starting stock minus ending stock, if ending stock is occasionally measured or inferred). Significant variances indicate issues: recipe inaccuracies, theft, portion control problems, or forecasting error. Use this to recalibrate recipe quantities or yields and improve the model. Over time, the goal is to shrink the gap between theoretical and actual usage, known as reducing the *Actual vs Theoretical variance* [15].

- **Exception Handling:** Flag anomalies such as zero sales days, inventory-negative projections, or huge order suggestions for human review. For example, if the model says to order 10x the usual amount of an ingredient (perhaps due to a data glitch or one-time event), managers should verify before the order is placed automatically.

- **User Overrides:** Allow managers to input manual adjustments (e.g., “we plan a special menu next week needing +20 lbs of salmon” or “vendor is running a promotion, buy extra now”). The system should accept these inputs and integrate them into the calculations.

- **Technology Integration:** To implement this in a restaurant inventory app, consider linking directly to the POS for real-time sales data, and to electronic vendor ordering systems for automatically placing suggested orders. Modern systems like Nory and others dynamically update par levels using POS data and send predictive order suggestions automatically [16] [17]. Ensuring data flows in real-time will make the model more reactive and reduce manual data entry errors.

In summary, this framework provides a **comprehensive, step-by-step approach** to forecasting sales, calculating theoretical usage, and managing inventory in a restaurant without frequent inventory counts. By tuning the forecast parameters, yield assumptions, and safety thresholds, the model can be optimized for different restaurant concepts and volumes. Implementing these calculations in software (with a clear UI for the outputs above) will enable restaurant operators to make data-driven ordering decisions, minimize waste, and keep food cost under tight control – all based on **theoretical usage and predictive analytics** rather than labor-intensive counts. [4] [12]

---

[1] A restaurant operator’s guide to actual vs. theoretical food costs and usage
https://www.marginedge.com/blog/a-restaurant-operators-guide-to-actual-vs-theoretical-food-costs-and-usage

[2] What is Yield and How Does it Affect Your Recipe Costs
https://www.recipe-costing.com/blog/what-is-yield-and-how-to-calculate-it/

[3] [12] [13] [14] [15] How to Consistently Conduct Actual vs. Theoretical Food Cost Analysis?
https://pos.toasttab.com/blog/on-the-line/actual-vs-theoretical-food-cost?srsltid=AfmBOoojP0bjL1tYZRzBqAe8792grAuGDISHNAwlKkOfLXOJ6kNCNUy_

[4] [11] Restaurant Forecasting Methods, Tools, and Best Practices
http://www.gsslp.com/restaurant-forecasting-methods

[5] [6] [7] [16] [17] How to Calculate Par Level in a Restaurant | Nory — Nory
https://www.nory.ai/blog/how-to-calculate-par-level-in-a-restaurant

[8] How to Create Restaurant Sales Forecast | Lineup.ai
https://lineup.ai/how-to-create-restaurant-sales-forecast/

[9] [10] How to Create Restaurant Sales Forecast | Lineup.ai
https://www.lineup.ai/how-to-create-restaurant-sales-forecast/