UNIVERSITY OF WOLLONGONG IN DUBAI

Assignment Cover Sheet

Subject Code: CSCI323
Subject Name: Modern AI
Submission Type: Report
Assignment Title: CSCI323_Final_Submission

Student Name: Sarah Amina Selama, Leen Taha, Benyamin Khademi, Sana Ullah Bilal Syed
Student Number: 7104017, 7840287, 7817939, 7598063

Student Email: sas966, lakyt124, byk882, subs227

Lecturer Name: Dr Patrick Mukala
Due Date:
Date Submitted:

PLAGIARISM:	DECLARATION:
The penalty for deliberate plagiarism is FAILURE in the subject. Plagiarism is cheating by using the written ideas or submitted work of someone else. UOWD has a strong policy against plagiarism. The University of Wollongong in Dubai also endorses a policy of non-discriminatory language practice and presentation. **PLEASE NOTE:** STUDENTS MUST RETAIN A COPY OF ANY WORK SUBMITTED	I/We certify that this is entirely my/our own work, except where I/we have given fully-documented references to the work of others, and that the material contained in this document has not previously been submitted for assessment in any formal course of study. I/we understand the definition and consequences of plagiarism. Signature of Student:

Optional Marks:

Comments:

Lecturer Assignment Receipt **(To be filled in by student and retained by Lecturer upon return of assignment)**

Subject: **Assignment Title:**

Student Name: **Student Number:**

Due Date: **Date Submitted:**

Signature of Student:

Student Assignment Receipt **(To be filled in and retained by Student upon submission of assignment)**

Subject: **Assignment Title:**

Student Name: **Student Number:**

Due Date: **Date Submitted:**

Signature of Lecturer

SmartStock AI

Executive Summary

SmartStock AI is a data-focused tool created to assist restaurants in minimizing food waste and enhancing inventory management via ingredient-specific predictions. The project sought to enhance sustainable and economical decision-making by forecasting ingredient demand through historical data and machine learning methods.

The system was created to convert daily sales and recipe information into valuable insights regarding ingredient usage, assisting restaurant managers in making more precise purchasing choices. To more accurately represent real-world scenarios, the model additionally included a static wastage buffer. A primary challenge was the organization and cleaning of inconsistent datasets to necessitate careful preprocessing and aggregation.

To identify long-term trends, ingredient usage was examined on a quarterly basis, and a targeted training dataset was created using 492 entries spanning 123 ingredients. A hybrid model using Exponential Smoothing and XGBoost was created to generate forecasts. Although real-time dashboards and external data (such as weather or promotions) were not included in this project's scope, they continue to be promising opportunities for future growth.

In summary, SmartStock AI illustrates how data analysis can enhance kitchen operations. Providing insights at the ingredient level establishes a foundation for improved procurement, minimized waste, and scalable inventory approaches in restaurant settings

Project Description

SmartStock AI is an AI-powered inventory management solution designed for restaurants, focused on improving ingredient predictions according to menu requirements. Utilizing past order data, recipe details, and usage logs for ingredients, the tool forecasts the optimal amounts of ingredients required. This allows restaurants to keep ideal stock levels, preventing both excess buying and shortages. The solution ultimately aids in decreasing operational costs and minimizing food waste. SmartStock AI is a component of a broader initiative focused on data-driven sustainability and cost-effectiveness in the food service industry of the UAE (ne'ma, 2024)

Motivation

Food waste constitutes a significant global challenge, as nearly one-third of all food generated, around 1.3 billion tonnes, is thrown away each year (GSTC, 2024). This immense waste accounts for roughly 8% of global carbon emissions and is a major factor in biodiversity loss, as agricultural practices endanger 86% of threatened species (GSTC, 2024). Forecasts suggest that food waste might rise by 60% by 2030, resulting in an estimated economic loss of $1.5 billion (GSTC, 2024).

The hospitality industry, encompassing restaurants and catering, is essential, accounting for a minimum of 15% of worldwide food waste (GSTC, 2024). This underscores the critical demand for intelligent systems that minimize overproduction and enhance distribution effectiveness. In the UAE, the issue is notably critical, as each individual produces approximately 224 kg of food waste per year, one of the highest levels worldwide (Sadeh and Hijleh, 2024). Dubai's hospitality sector, with over 20,000 dining establishments, plays a major role in this problem because of its abundant buffet selections, wide-ranging menus, and inadequate demand predictions (Sadeh and Hijleh, 2024). The waste produced costs the city approximately 282 million AED annually (Sadeh and Hijleh, 2024). Even though tools like Winnow have effectively decreased waste in certain hotels by as much as 70%, broader data-driven approaches are still insufficient. SmartStock AI fills this void by helping restaurants enhance their menu and purchasing choices, while also aligning with national sustainability objectives like SDG 12.3 and the Zero Hunger by 2051 initiative (Sadeh and Hijleh, 2024)

Technical Requirements and Main Aspects

SmartStock AI is a Python-driven inventory optimization solution aimed at assisting restaurants in reducing food waste and enhancing procurement strategies via data-informed ingredient demand predictions. The system is developed using Python libraries such as Pandas, NumPy, Seaborn, Matplotlib, Scikit-learn, and XGBoost, and is structured into modular parts to guarantee flexibility and scalability.

The process starts with Data Acquisition and Preprocessing, involving the loading, cleaning, and merging of various datasets, encompassing daily sales information, menu items, recipe details, and ingredient metadata. Cleaning processes involve renaming inconsistent columns, managing missing values, changing data types, and standardizing ingredient names. Financial metrics like gross margin and gross margin percentage are calculated to enhance the dataset.

After conducting a cost analysis, a training dataset was formed on a quarterly basis, assessing 123 ingredients across 4 quarters, resulting in 492 data points. The choice to train at the ingredient level rather than focusing on the final dish was essential, as various ingredients feature in numerous recipes. A proportional share approach was implemented to allocate sales and cost data according to each ingredient's role in the final product, guaranteeing precise reflection in the model inputs.

The Ingredient Demand Calculation Engine determines the daily needs for each ingredient by multiplying sales figures with recipe amounts, factoring in a constant 5% waste buffer to mimic real-life spoilage and inefficiencies. A Z-score analysis-based Outlier Detection Module identifies anomalies in usage patterns, though stronger detection techniques might be explored in future versions.

To make predictions, the system assesses and contrasts various models, such as Exponential Smoothing, Random Forest, Ridge Regression, MLPRegressor, and the chosen model, XGBoost. Designed elements such as rolling averages, lags, quarterly intervals, and day-of-week indicators are utilized to improve predictive performance. Forecast results are compiled into a Parquet file for easy access and additional analysis.

To aid in decision-making, the system features Visual Analytics Dashboards, including time series graphs, ingredient-level heatmaps, and trend analyses. Correlation analyses were performed to examine how order types and sales channels affect ingredient usage, yet these factors were omitted from the present model and will be considered for future improvement.

SmartStock AI is built to be modular and scalable, able to adjust to various kitchens, recipes, and ingredient arrangements, as long as sufficient historical data exists. It signifies a tangible advancement in AI-supported procurement planning within food service settings.

Implementation

This section presents the complete implementation of our ingredient demand forecasting system, built in Python using libraries such as Pandas, NumPy, Matplotlib, Seaborn, and Scikit-learn. The workflow includes data cleaning, analysis, dataset merging, ingredient usage calculation, wastage adjustment, and the application of time-series forecasting models. Below is a breakdown of the code, along with an explanation of each functional block.

1. Importing Required Libraries

We began by importing essential libraries for data manipulation (pandas,numpy), visualization (matplotlib.pyplot,seaborn), and date/time handling (datetime). To maintain a cleaner output, warning messages were suppressed.

Sample Code:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

# For datetime
from datetime import datetime

# Suppress warnings
import warnings
warnings.filterwarnings("ignore")

```

2. Loading and Cleaning Datasets

The sales and production recipe datasets were loaded using the read_csv function, with the encoding set to 'latin1' to ensure compatibility with special characters. Column names were standardized to snake_case for consistency, unnecessary columns were removed, and typographical errors in column names were corrected.

Sample Code:

```
sales = pd.read_csv("Sales_Dataset.csv", encoding='latin1')
production_recipe = pd.read_csv("production_recipe.csv", encoding='latin1')
```

```python
# Remove whitespace and normalize column names to snake_case
#SALES
sales.columns = (
    sales.columns
    .str.strip()
    .str.replace(' ', '_')
    .str.lower()
)

# PRODUCTION
production_recipe.columns = (
    production_recipe.columns
    .str.strip()
    .str.replace(' ', '_')
    .str.lower()
)

# Sales
sales = sales[
    [
    'system_date',
    'food_id',
    'dish_name',
    'production_food_name',
    'quantity',
    'total_price',
    'food_cost'
    ]
]
# Convert date columns - fixing the date format issue by specifying dayfirst=True
sales['system_date'] = pd.to_datetime(sales['system_date'], dayfirst=True)

# Production
# Fix typo
production_recipe.rename(columns={'ingredians': 'ingredients'}, inplace=True)
```

3. Data Type Conversion and Metric Calculation

The system_date column was converted to datetime format to enable proper date-based operations. Dollar signs were removed from the financial columns total_price and food_cost, after which the values were converted to numeric type. Gross margin and gross margin percentage were then calculated and added as new columns.

Sample Code:

```
# Margin metrics (only if these columns exist)
if {'total_price', 'food_cost'}.issubset(sales.columns):
  #clean columns
    sales['total_price'] = sales['total_price'].astype(str).str.replace('$', '', regex=False).str.strip()
    sales['food_cost'] = sales['food_cost'].astype(str).str.replace('$', '', regex=False).str.strip()
  #convert columns to numeric values
    sales['total_price'] = pd.to_numeric(sales['total_price'], errors='coerce')
    sales['food_cost'] = pd.to_numeric(sales['food_cost'], errors='coerce')
  #add calculation
    sales['gross_margin'] = sales['total_price'] - sales['food_cost']
    sales['gross_margin_percent'] = (sales['gross_margin'] / sales['total_price']) * 100

print(sales.head())
print(production_recipe.head())
```

4. Exploratory Analysis of Sales Data

Descriptive statistics were generated, and the top 15 sold dishes were identified. This provided an initial understanding of sales trends.

Sample Code and Output:

Sales data summary

```
print("Descriptive Statistics for Sales:")
print("\nTop 15 Dishes Sold:")
top_dishes = sales.groupby(['food_id', 'dish_name'])['quantity'].sum().sort_values(ascending=False).head(15)
print(top_dishes)

print("\nSales Columns")
print(sales.columns)
print("\nProduction Columns")
print(production_recipe.columns)
```

```
Descriptive Statistics for Sales:

Top 15 Dishes Sold:
food_id  dish_name
```

Descriptive Statistics for Sales:

Top 15 Dishes Sold:

food_id	dish_name	
893	Strawberry Sunrise	5965
1002	Creole Shrimp and Grits	5046
894	Crispy Karma	4919
1319	Southern Fried Catfish Po' Boy	4698
939	The Crispy Crunch Burger	4319
1008	Sweet Tea-Brined Pork Chops	4243
963	The Philly Cheesesteak Inspired Burger	3706
1000	Peach Perfection	3431
931	Delish Dhaba	3200
1381	Cajun Jambalaya	2554
910	Curry Chronicles	2372
940	Saffron Symphony	2323
946	The Spicy Sriracha Kick Burger	2280
930	Epic Flavor Quest	2179
1311	Banana Pudding	2163

Name: quantity, dtype: int64

5. Merging Datasets for Ingredient Demand Calculation

Sales dataset was merged with the production recipe dataset using the join function on the production_food_name label as the common key. Total ingredient demand for each sale was then calculated by multiplying the quantity sold by the quantity of each ingredient required per dish.

Sample Code and Output:

```
# Step 1: Merge sales with production_recipe to get ingredient-level demand per sale
merged = pd.merge(
    sales,
    production_recipe,
    on="production_food_name",
    how="inner",
    suffixes=("_sale", "_production")
)
# Step 2: Calculate total ingredient demand per sale row
# Each sale row now has a 'quantity' (dishes sold) and a 'quantity_y' (ingredient amount per dish)
merged['ingredient_total_qty'] = merged['quantity_sale'] * merged['quantity_production']
# Step 3: Aggregate total demand for each ingredient
ingredient_demand = (
    merged
    .groupby('ingredients')['ingredient_total_qty']
    .sum()
    .reset_index(name='total_demand')
    .sort_values('total_demand',ascending=False)
)

# Step 4: Display the result
print(ingredient_demand.head(10))


#check in order of RM Name
ingredient_demand['RM Code'] = (
    ingredient_demand['ingredients']
    .str.extract(r'(\d+)$')
    .astype(int)
)
ingredient_demand_sorted = ingredient_demand.sort_values('RM Code')
print(ingredient_demand_sorted.head(10))
```

First output shows us ingredients based off demand in descending order and second output shows us ingredient demand based off the numeric order of ingredient

```
     ingredients  total_demand
74    RM Name 55   65695942.92
112    RM Name 9   32469605.79
59    RM Name 41   31843050.00
122   RM Name 99   28458000.00
23    RM Name 12   25027710.00
63    RM Name 45   24513821.69
1     RM Name 10   21283690.00
79     RM Name 6   16480574.00
90     RM Name 7   11046729.18
34    RM Name 19    7388593.18
     ingredients  total_demand  RM Code
0     RM Name 1     779040.00        1
35    RM Name 2    1228765.10        2
46    RM Name 3      19316.00        3
57    RM Name 4      19316.00        4
68    RM Name 5    2398083.47        5
79    RM Name 6   16480574.00        6
90    RM Name 7   11046729.18        7
101   RM Name 8    6918264.53        8
112   RM Name 9   32469605.79        9
```

6. Daily Ingredient Demand and Wastage Adjustment

Daily ingredient demand was calculated by grouping the data using a date-based grouper and extracting ingredient quantity from the previous cell.

Sample Code:

```
restaurant_daily_ingredient_demand = (
    merged
    .groupby([
        pd.Grouper(key='system_date', freq='D'),
        'ingredients',
        ])['ingredient_total_qty']
    .sum()
    .reset_index()
)

restaurant_daily_ingredient_demand.head()
```

	system_date dat...	ingredients object	ingredient_total_...
0	2024-01-01 00:0...	RM Name 1	480
1	2024-01-01 00:0...	RM Name 10	48540
2	2024-01-01 00:0...	RM Name 100	500
3	2024-01-01 00:0...	RM Name 102	5000
4	2024-01-01 00:0...	RM Name 103	300

Applying 5% wastage rate on daily ingredient demand + checking if rate was applied correctly

```python
wastage_rate = 0.05
restaurant_daily_ingredient_demand['ingredient_total_qty_adj'] = (
    restaurant_daily_ingredient_demand['ingredient_total_qty']
    * (1 + wastage_rate)
)

# Summarize the impact of wastage adjustment by ingredient
adj_summary = (
    restaurant_daily_ingredient_demand
    .groupby('ingredients')['ingredient_total_qty_adj']
    .sum()
    .reset_index(name='total_demand_with_wastage')
)
print("Total Adjusted Ingredient Demand by Type (with 5% wastage):")
print(adj_summary)

#Check the scale factor of RM Name 1
rm1 = restaurant_daily_ingredient_demand[
    restaurant_daily_ingredient_demand['ingredients'] == 'RM Name 1'
]
orig_sum = rm1['ingredient_total_qty'].sum()
adj_sum  = rm1['ingredient_total_qty_adj'].sum()

# Print both and the ratio
print(f"RM Name 1 — Original: {orig_sum:.2f}, Adjusted: {adj_sum:.2f}, Ratio: {adj_sum/orig_sum:.4f}")
```

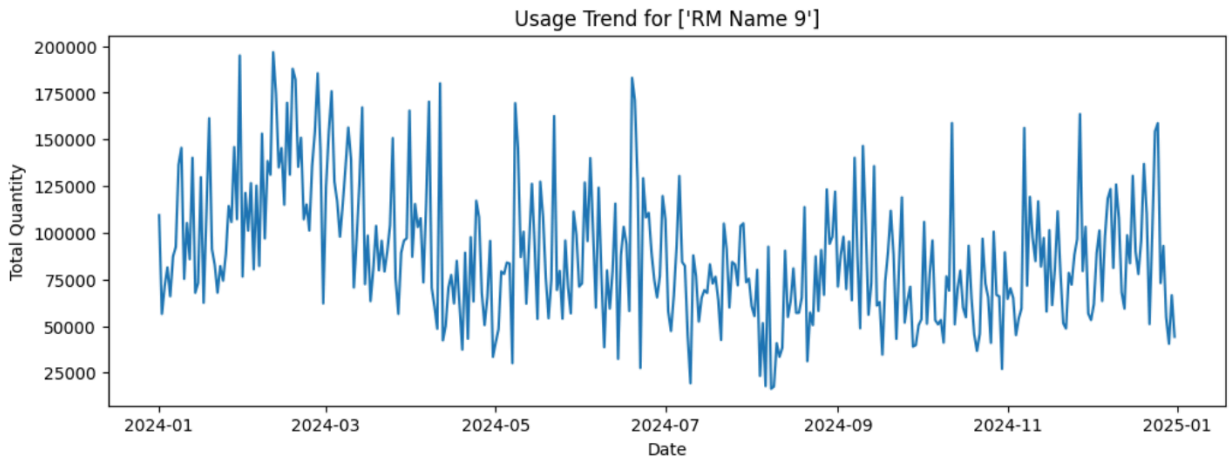
```
Total Adjusted Ingredient Demand by Type (with 5% wastage):
     ingredients  total_demand_with_wastage
0      RM Name 1               8.179920e+05
1     RM Name 10               2.234787e+07
2    RM Name 100               2.490075e+05
3    RM Name 101               3.528000e+03
4    RM Name 102               1.737750e+06
..           ...                        ...
118   RM Name 95               1.018500e+05
119   RM Name 96               1.782375e+04
120   RM Name 97               2.053800e+06
121   RM Name 98               1.611203e+05
122   RM Name 99               2.988090e+07

[123 rows x 2 columns]
RM Name 1 — Original: 779040.00, Adjusted: 817992.00, Ratio: 1.0500
```

Exploratory Analysis

7. Visualization and Trend Analysis

Trends for a specific ingredient (e.g., RM Name 9) were visualized through a time series plot. Additionally, dishes containing that ingredient were identified using logical filtering techniques.

Sample Output:



8. Dish-to-ingredient Mapping

Here we find dishes that use a specific ingredient by filtering the production recipe for foods containing it, then checking sales data for dishes made from those foods. It selects key columns, removes duplicates, and shows the first 15 results.

Sample Code and Output:

```
# Map which dishes use a specific ingredient
ingredient_name = 'RM Name 9'

# Find all dishes that use that ingredient in production
food_withRM = production_recipe.loc[
    production_recipe['ingredients']==ingredient_name,
    'production_food_name'
].unique()

#find all the dishes that use that food_name in sales
dish_withRM = (
    sales[sales['production_food_name'].isin(food_withRM)]
    [['food_id','production_food_name','dish_name']]
    .drop_duplicates(subset=['production_food_name'], keep='first')
)

#display results
print(f"Dishes using {ingredient_name}:")
display(dish_withRM.head(15))
```

Dishes using RM Name 9:

	food_id int64 917 - 1570	production_food_name object Production Food Name 2 6.7% Production Food Name 5 6.7% 13 others 86.7%	dish_name object Pimento Cheese-St... 6.7% Southern Fried Gree... 6.7% 13 others 86.7%
3	1010	Production Food Name 2	Pimento Cheese-Stuffed Jala...
6	1006	Production Food Name 5	Southern Fried Green Tomatoes
105	1371	Production Food Name 13	Slow-Cooked Pulled Pork
760	920	Production Food Name 17	Chaat Carnival
761	928	Production Food Name 18	Artisanal Appetites
883	921	Production Food Name 20	Caramel Macchiato Delight
1044	917	Production Food Name 21	Blue Lagoon Mocktail
1345	922	Production Food Name 35	Savory Spell
1662	946	Production Food Name 42	The Spicy Sriracha Kick Burger
2118	965	Production Food Name 49	The Tangy Teriyaki Pineapple ...

15 rows, 3 cols 10 / page « ‹ Page 1 of 2 › »

9. Outlier Detection

Outliers (based on Z-score beyond ±3)

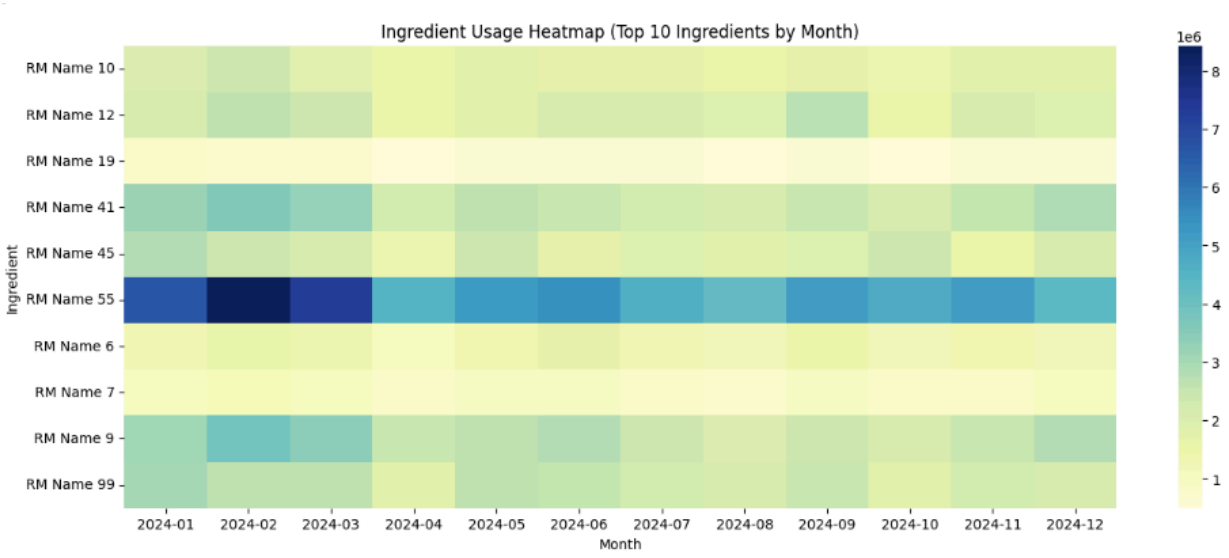
Sample Code and Output:

```
# Identify days with unusually high ingredient usage (e.g., z-score > 3)
from scipy.stats import zscore
restaurant_daily_ingredient_demand['zscore'] = restaurant_daily_ingredient_demand.groupby('ingredients')['ingredient_t
outliers = restaurant_daily_ingredient_demand[restaurant_daily_ingredient_demand['zscore'].abs() > 3]
print(outliers[['system_date', 'ingredients', 'ingredient_total_qty', 'zscore']])
```

10. Heatmap Visualization

A heatmap was created to show usage of top 10 ingredients across restaurants. The pivot table provided a summary that was visualized using Seaborn's heatmap().

Sample Output:



11. Cost Analysis

We merged food cost data from the sales dataset into the production recipe, ensuring only unique combinations were retained to avoid duplicates. Next, we calculated the total quantity of ingredients used in each dish and determined each ingredient's proportional share within its recipe. Using these proportions, we allocated the total food cost accordingly. Due to the initial results of the cost per ingredient varying a lot due to the different dates and presumably external factors (i.e. inflation), we decided a good approach would be to find the average cost of the ingredient. Finally, we sorted the results by ingredient number to improve readability.

We then organized ingredient data by quarter to evaluate supply, demand, cost, revenue, and profitability. First, we added a quarter column to the sales data and calculated the total quantity of each ingredient produced. Next, we estimated quarterly demand per ingredient based on sales data, merged in cost information, and calculated revenue using ingredient shares and sale prices. Finally, we computed supply cost, demand cost, profit, and surplus or shortage per quarter, and exported the results for further analysis.

Sample Code:

```python
# 1. Merge food_cost from sales into production_recipe
# Only keep unique production_food_name & food_cost combinations
sales_cost = (
    sales[['production_food_name', 'food_cost']]
    .drop_duplicates()
    )

# Merge
recipe_with_cost = (
    production_recipe.merge(
    sales_cost,
    on='production_food_name',
    how='left')
)

# 2. Calculate the total quantity of all ingredients for each production_food_name
recipe_with_cost['total_recipe_quantity'] = (
    recipe_with_cost.groupby('production_food_name')['quantity'].transform('sum')
)

# 3. Calculate the share of each ingredient in its recipe
recipe_with_cost['ingredient_share'] = (
    recipe_with_cost['quantity'] / recipe_with_cost['total_recipe_quantity']
)

# 4. Allocate the food cost to each ingredient (share * total cost)
recipe_with_cost['food_cost'] = recipe_with_cost['food_cost'].astype(float)
recipe_with_cost['ingredient_cost'] = (
    recipe_with_cost['ingredient_share'] * recipe_with_cost['food_cost']
)

# 5. Create the final DataFrame for model training by getting the sum of total cost of every ingredient
#and computing the average as the final cost value for cleaner data
ingredient_total_cost_df = (
    recipe_with_cost
    .groupby('ingredients')['ingredient_cost']
    .mean()
    .reset_index()
    .rename(columns={'ingredients': 'ingredient_RM', 'ingredient_cost': 'total_cost'})
    .sort_values('total_cost', ascending=False)
)

# Dispaly
display(ingredient_total_cost_df)


# display in order of ingredient
ingredient_total_cost_df['rm_number'] = (
    ingredient_total_cost_df['ingredient_RM']
    .str.extract(r'(\d+)')
    .astype(int)
    )

ingredient_total_cost_df_sorted = ingredient_total_cost_df.sort_values('rm_number').reset_index(drop=True)
display(ingredient_total_cost_df_sorted)
```

	ingredient_RM	total_cost
117	RM Name 94	0.099896
27	RM Name 123	0.084361
89	RM Name 69	0.076809
81	RM Name 61	0.071956
92	RM Name 71	0.060115
...	...	...
2	RM Name 100	0.000122
57	RM Name 4	0.000082
46	RM Name 3	0.000082
37	RM Name 21	0.000078
61	RM Name 43	0.000028

123 rows × 2 columns

Creating training dataset

- **Made an educated guess of going by quarter as the production dataset did not contain any dates and so we went off the assumption that the same production quantity (supply) was used every quarter**

```python
# ARRANGE INGREDIENTS BY QUARTER
# Create column in sales for every quarter
sales['quarter'] = pd.to_datetime(sales['system_date']).dt.to_period('Q')

# Get unique list of quarters in your sales data
quarters = sales['quarter'].unique()
ingredients = production_recipe['ingredients'].unique()

# Compute total supply of each ingredient for the 3-month period (from production_recipe)
# If you assume you produce the entire production_recipe each quarter:
supply_by_ingredient = (
    production_recipe
    .groupby('ingredients')['quantity']
    .sum()
    .reset_index()
    .rename(columns={'quantity': 'supply_quantity'})
    )

# Make a DataFrame with all ingredient/quarter combos
quarterly_supply = pd.MultiIndex.from_product([ingredients, quarters], names=['ingredients', 'quarter']).to_frame(index=False)
quarterly_supply = quarterly_supply.merge(supply_by_ingredient, on='ingredients', how='left')

# 1. Compute demand per ingredient per quarter from sales
# First, total units sold per product per quarter
units_sold_quarter = (
    sales
    .groupby(['production_food_name', 'quarter'])['quantity']
    .sum()
    .reset_index()
    .rename(columns={'quantity': 'units_sold'})
    )

# Merge with production_recipe to get total ingredient demand per product per quarter
ingredient_demand_quarter = (
    production_recipe
    .merge(units_sold_quarter, on='production_food_name', how='left')
)
ingredient_demand_quarter['total_demand'] = ingredient_demand_quarter['quantity'] * ingredient_demand_quarter['units_sold']

# Group to get total ingredient demand per ingredient per quarter
ingredient_total_demand_quarter = (
    ingredient_demand_quarter.groupby(['ingredients', 'quarter'])['total_demand']
    .sum().reset_index().rename(columns={'total_demand': 'ingredient_total_demand'})
```

```
    ingredient_demand_quarter.groupby(['ingredients', 'quarter'])['total_demand']
    .sum().reset_index().rename(columns={'total_demand': 'ingredient_total_demand'})
)

# 2. Merge cost per ingredient
quarterly_profit_df = (
    quarterly_supply.merge(
        ingredient_total_cost_df[['ingredient_RM', 'total_cost']],
        left_on='ingredients',
        right_on='ingredient_RM',
        how='left'
    )
)

# 3. Merge in ingredient demand per quarter
quarterly_profit_df = quarterly_profit_df.merge(
    ingredient_total_demand_quarter,
    on=['ingredients', 'quarter'],
    how='left'
).fillna(0)

# 4. Allocate revenue per ingredient per quarter
# Re-calculate ingredient share from the recipe
production_recipe['total_recipe_quantity'] = production_recipe.groupby('production_food_name')['quantity'].transform('sum')
production_recipe['ingredient_share'] = production_recipe['quantity'] / production_recipe['total_recipe_quantity']

# Merge share into sales and calculate ingredient revenue for each sale/quarter
sales_ingredients = sales.merge(
    production_recipe[['production_food_name', 'ingredients', 'ingredient_share']],
    on='production_food_name', how='left'
)
sales_ingredients['quarter'] = pd.to_datetime(sales_ingredients['system_date']).dt.to_period('Q')
sales_ingredients['ingredient_revenue'] = sales_ingredients['ingredient_share'] * sales_ingredients['total_price']

# Sum up revenue per ingredient per quarter
ingredient_revenue_quarter = sales_ingredients.groupby(['ingredients', 'quarter'])['ingredient_revenue'].sum().reset_index()

# Merge revenue into quarterly_profit_df
quarterly_profit_df = quarterly_profit_df.merge(
    ingredient_revenue_quarter, on=['ingredients', 'quarter'], how='left'
).fillna(0)
```

```

# 5. Calculate profit/loss and surplus/shortage for each quarter
quarterly_profit_df['total_supply_cost'] = quarterly_profit_df['supply_quantity'] * quarterly_profit_df['total_cost']
quarterly_profit_df['total_demand_cost'] = quarterly_profit_df['ingredient_total_demand'] * quarterly_profit_df['total_cost']
quarterly_profit_df['profit'] = quarterly_profit_df['ingredient_revenue'] - quarterly_profit_df['total_demand_cost']
quarterly_profit_df['surplus_shortage'] = quarterly_profit_df['supply_quantity'] - quarterly_profit_df['ingredient_total_demand']
quarterly_profit_df['profit_margin'] = (quarterly_profit_df['profit'] / quarterly_profit_df['ingredient_revenue'].replace(0, pd.NA)) * 100

# 6. Display results
cols_to_display = [
    'ingredients', 'quarter', 'supply_quantity', 'ingredient_total_demand', 'surplus_shortage',
    'total_cost', 'total_supply_cost', 'total_demand_cost',
    'ingredient_revenue', 'profit', 'profit_margin'
]
quarterly_profit_df_sorted = quarterly_profit_df[cols_to_display].sort_values(
    ['quarter', 'profit'], ascending=[True, False]
)
quarterly_profit_df_sorted.head(20)

# 7. Export results
quarterly_profit_df.to_csv('quarterly_ingredient_data.csv', index=False)
```

12. Using Newly Generated Dataset

Sample Code and Output:

Load dataset

```
df = pd.read_csv('quarterly_ingredient_data.csv')
df.head()
```

	ingredients object	quarter object	supply_quantity f...	ingredient_RM o...	total_cost float64	ingredient_total_...	ingredient_reven...	t
0	RM Name 1	2024Q1	120	RM Name 1	0.002679319966	189000	699.1664486	
1	RM Name 1	2024Q4	120	RM Name 1	0.002679319966	187320	642.8066179	
2	RM Name 1	2024Q2	120	RM Name 1	0.002679319966	195960	685.2013851	
3	RM Name 1	2024Q3	120	RM Name 1	0.002679319966	206760	724.3659869	
4	RM Name 2	2024Q1	2105.31	RM Name 2	0.00097630677...	358128.51	2086.439849	

13. Model Design

We implemented a forecasting model to predict ingredient supply using both traditional time series methods and machine learning techniques. First, we sorted the data and encoded ingredient names.

Next, to account for the major issues faced in real-world forecasting, which are being able to capture past trends and utilizing all of the additional data about products, expenses, and operations, we found that an effective method for predicting time series data, such as the quarterly supply of ingredients, would be using Exponential Smoothing.

Exponential Smoothing will help detect general patterns for each ingredient in terms of supply and demand and each trend per ingredient is used as an additional **feature** to give XGBoost model a form of direction and perspective**.** Then, using selected cost, demand, revenue variables, profit margins etc, we trained an XGBoost regressor with 5-fold grouped cross-validation based on ingredient to account for external factors missed by Exponential Smoothing.

Model performance was evaluated using mean absolute error, and predictions were stored alongside actual values and other relevant metrics for further analysis.

Sample Code and Output:

```
"""# Model Design

### Encoding, Feature Engineering and Training
"""

!pip install xgboost==3.0.2

import warnings
warnings.filterwarnings("ignore", category=UserWarning, module="statsmodels")

from statsmodels.tsa.holtwinters import ExponentialSmoothing
from sklearn.model_selection import GroupKFold
from sklearn.metrics import mean_absolute_error
from xgboost import XGBRegressor
from sklearn.preprocessing import LabelEncoder
import pandas as pd # Import pandas if not already imported

# 1. Sort and encode
df = df.sort_values(['ingredients', 'quarter'])
le = LabelEncoder()
df['ingredients_encoded'] = le.fit_transform(df['ingredients'])

# Convert 'quarter' from object to Period and then to a numerical format
# Assuming 'quarter' is in 'YYYYQn' format after reading the CSV
df['quarter'] = pd.PeriodIndex(df['quarter'], freq='Q')
df['quarter_encoded'] = df['quarter'].apply(lambda x: x.year * 4 + x.quarter)


# 2. Compute Exponential Smoothing forecast as a feature
df['es_forecast'] = np.nan
for ingr, group in df.groupby('ingredients'):
    group = group.sort_values('quarter')
    es_preds = []
    for i in range(len(group)):
        if i < 2:
            es_preds.append(np.nan)
        else:
            es = ExponentialSmoothing(
                group['supply_quantity'].iloc[:i],
                trend='add', seasonal=None, initialization_method="estimated"
            )
            es_fit = es.fit()
            es forecast = es fit.forecast(1).values[0]
```

```
            es_fit = es.fit()
            es_forecast = es_fit.forecast(1).values[0]
            es_preds.append(es_forecast)
    df.loc[group.index, 'es_forecast'] = es_preds

print("ES forecasts added as feature.")

# 3. Choose features for the regression
features = [
    'ingredient_total_demand', 'profit', 'profit_margin', 'total_cost', 'ingredient_revenue',
    'es_forecast',
    'ingredients_encoded', 'quarter_encoded' # Use the new numerical quarter feature
]
target = 'supply_quantity'

# 4. Drop rows missing any values
df_model = df.dropna(subset=[target, 'es_forecast'] + features).copy()
X = df_model[features]
y = df_model[target]
groups = df_model['ingredients']

# 5. Cross-validation (by ingredient)
kf = GroupKFold(n_splits=5)
mae_scores = []
mae_pct_scores = []
all_preds = []

for train_idx, test_idx in kf.split(X, y, groups):
    X_train, X_test = X.iloc[train_idx], X.iloc[test_idx]
    y_train, y_test = y.iloc[train_idx], y.iloc[test_idx]
    # Removed enable_categorical as we converted 'quarter' to numerical
    model = XGBRegressor(n_estimators=100, max_depth=5, learning_rate=0.1, random_state=42)
    model.fit(X_train, y_train)
    preds = model.predict(X_test)
    fold_mae = mean_absolute_error(y_test, preds)
    mean_actual = np.mean(y_test)
    fold_mae_pct = 100 * fold_mae / mean_actual if mean_actual != 0 else np.nan
    mae_scores.append(fold_mae)
    mae_pct_scores.append(fold_mae_pct)
    print(f"Fold MAE (XGB + ES feature): {fold_mae:.2f} units ({fold_mae_pct:.2f}% of mean actual supply)")

    # Save results
    fold_df = pd.DataFrame({
```

```
        })
        all_preds.append(fold_df)

    # Combine all and save
    preds_df = pd.concat(all_preds, ignore_index=True)
    print("Computation complete.")
```

```
ES forecasts added as feature.
Fold MAE (XGB + ES feature): 1940.03 units (47.87% of mean actual supply)
Fold MAE (XGB + ES feature): 182.67 units (7.36% of mean actual supply)
Fold MAE (XGB + ES feature): 620.94 units (22.66% of mean actual supply)
Fold MAE (XGB + ES feature): 292.90 units (5.25% of mean actual supply)
Fold MAE (XGB + ES feature): 544.00 units (12.86% of mean actual supply)
Computation complete.
```

14. Model Evaluation and Performance

Finally, the model's financial performance was evaluated by comparing predicted and actual profits and losses for each ingredient on a quarterly basis. Profit was calculated as revenue minus cost, with negative values recorded as losses. Total, net, and average figures were then computed to provide a comprehensive assessment. This approach enabled evaluation of the model's accuracy in terms of financial impact as well as unit predictions. The analysis concluded by highlighting the top five most and least profitable predictions, illustrating areas of strong and weak model performance.

Sample Output:

```
Total predicted profit: 185,961.60
Total actual profit:    185,208.42
Total predicted loss:   770.18
Total actual loss:      832.79
Net predicted profit: 184,421.24
Net actual profit:    183,542.84
```

```
Model Performance Across Folds
Mean MAE: 716.11 units
Mean MAE (as % of mean supply): 19.20%
Average predicted profit: 752.81
Average actual profit:    749.49
Average predicted loss:   3.13
Average actual loss:      3.39
```

```
Top 5 most profitable predictions:
     ingredient quarter   pred_profit
180   RM Name 6  2024Q3  11856.342075
181   RM Name 6  2024Q4  11148.929090
239  RM Name 81  2024Q4   8494.151107
238  RM Name 81  2024Q3   8160.059122
51   RM Name 10  2024Q4   7401.932874

Top 5 least profitable predictions:
      ingredient quarter  pred_profit
150  RM Name 102  2024Q3  -171.010169
151  RM Name 102  2024Q4  -117.895240
10   RM Name 123  2024Q3   -73.886265
184   RM Name 69  2024Q3   -61.244438
61   RM Name 122  2024Q4   -53.366422
```

Limitations and Difficulties:

Despite the ingredient demand forecasting system created in this project including several elements: from data cleaning and cost analysis to feature engineering and model-based forecasting, numerous limitations and challenges emerged during the process. A major challenge was the organization and cleaning of the initial datasets. Ingredient names frequently varied between datasets, necessitating manual normalization to facilitate accurate merging. This procedure took a lot of time and posed a risk of human mistakes; employing semantic normalization or Natural Language Processing methods could have enhanced data quality but was outside the project's limits.

A new challenge arose following the cost analysis stage. A fresh dataset was generated for training the model, relying on the quarterly performance of each ingredient. Nevertheless, this dataset was quite limited, comprising just 492 entries (123 ingredients spread over 4 quarters). The small dataset size restricted the model's capacity to generalize patterns and probably led to more unstable performance across validation folds. To address this, forecasting was performed at the ingredient level instead of the final product level. This was needed because numerous ingredients were found in several production food items, and their contribution (or ingredient share) differed among recipes. Distributing revenue, expenses, and demand according to ingredient proportion guaranteed that the model reflected real-world consumption and financial trends more accurately

The forecasting model included Exponential Smoothing as a time series element and employed XGBoost for regression, but it failed to account for external seasonal or contextual elements like holidays, marketing efforts, or weather conditions, which could greatly affect demand. Additionally, while simple hyperparameter tuning was conducted, more sophisticated forecasting methods like LSTM networks or Temporal Fusion Transformers were not utilized because of time and scope limitations.

Outlier identification depended on Z-score calculations, which were effective but had constraints when dealing with skewed or atypical data. Stronger techniques such as Isolation Forest or IQR methods were evaluated but ultimately dismissed. Moreover, ingredient waste was estimated at a uniform 5% overall, which, although suitable for illustration, does not accurately represent the varying spoilage rates and overproduction hazards of specific ingredients. A historical or data-driven method for wastage modeling might have resulted in more precise cost estimates.

Ultimately, data quality remained an ongoing concern. Certain columns contained formatting discrepancies like dollar signs in numerical fields and numerous records were missing precise timestamps, restricting the accuracy of time-related analysis and possibly affecting the usefulness of the quarterly segmentation

Conclusion

This initiative effectively created a fundamental framework for forecasting ingredient needs in restaurant operations. Through the integration of data preprocessing, enrichment, consolidation, visualization, anomaly detection, and machine learning predictions, we developed a solution capable of estimating ingredient needs while considering real-world challenges such as ingredient sharing and waste.

The model training utilized a dataset aggregated quarterly to reflect trends over time, and forecast accuracy improved through the use of engineered time-series features. Integrating various models leading to the selection of XGBoost which enabled us to evaluate different forecasting methods and identify the most efficient option. The system's modular structure guarantees adaptability and flexibility for different application scenarios.

Even with various constraints like a constant wastage rate, excluding external factors such as holidays or weather, and depending on simple anomaly detection, the system provides real benefits by encouraging more precise purchasing and minimizing overproduction. The visual insights foster increased confidence in decision-making, and the predictive element provides planning abilities for the future.

In summary, SmartStock AI demonstrates the effective combination of data science and operational forecasting, providing a smart and scalable solution for inventory management within the food service sector. With improvements such as external data integration, NLP-driven ingredient normalization, and deep learning, the system possesses substantial promise for additional development and influence.

Reference list:

GSTC (2024). *Purposeful Travel Food Waste Reduction GSTC2024 Sweden Global Sustainable Tourism Conference*. [online] Available at: https://www.gstc.org/wp-content/uploads/Koko-Tang_compressed.pdf [Accessed 11 May 2025].

ne'ma (2024). *About - ne'ma*. [online] Nema.ae. Available at: https://www.nema.ae/en/about.

Sadeh, R. and Hijleh, B.A., 2024. *Investigating the food waste status in the hospitality sector of the Emirate of Dubai-UAE*. In: K. Al Marri et al., eds. BUiD Doctoral Research Conference 2023. Cham: Springer, pp.182–189. doi:10.1007/978-3-031-56121-4_18.