main Food Recommendation

September 19, 2025

[1]:
```
# Food Recommendation Project v4
# Context-aware recommendations for RandomForest & XGBoost, plus Item-CF

import pandas as pd
import numpy as np
from faker import Faker
import random
from collections import defaultdict

fake = Faker()
num_orders = 100000
```

[2]:
```
categories = {
    'Pizza': ['Margherita', 'Pepperoni', 'Hawaiian', 'BBQ Chicken', 'Veggie'],
    'Burger': ['Cheeseburger', 'Chicken Burger', 'Veggie Burger', 'Bacon␣
 ↪Burger'],
    'Pasta': ['Spaghetti Bolognese', 'Penne Arrabiata', 'Fettuccine Alfredo',␣
 ↪'Lasagna'],
    'Fish and chips': ['Classic Fish & Chips', 'Spicy Fish & Chips', 'Vegan␣
 ↪Fish & Chips'],
    'Kebabs': ['Chicken Kebab', 'Lamb Kebab', 'Beef Kebab', 'Veg Kebab'],
    'Wrap': ['Chicken Wrap', 'Veggie Wrap', 'Falafel Wrap'],
    'Calzone': ['Ham & Cheese Calzone', 'Veggie Calzone', 'Pepperoni Calzone']
}

drinks = ['Coke', 'Pepsi', 'Fanta', 'Water', 'Orange Juice', 'Lemonade']
sides = ['Fries', 'Onion Rings', 'Salad', 'Garlic Bread', 'Mashed Potatoes']
starters = ['Soup', 'Spring Rolls', 'Chicken Wings', 'Bruschetta', 'Garlic␣
 ↪Mushrooms']
desserts = ['Ice Cream', 'Brownie', 'Cake', 'Fruit Salad', 'Pudding']

prices = {
    'Pizza': (8, 20),
    'Burger': (5, 12),
    'Pasta': (7, 15),
    'Fish and chips': (6, 14),
    'Kebabs': (6, 14),
```

```
    'Wrap': (5, 12),
    'Calzone': (8, 16),
    'Drink': (1, 4),
    'Side': (2, 6),
    'Starter': (3, 8),
    'Dessert': (3, 7)
}

# Generate synthetic orders
order_data = []
for i in range(1, num_orders+1):
    customer_id = fake.uuid4()
    order_date = fake.date_time_this_year()
    main_category = random.choice(list(categories.keys()))
    main_item = random.choice(categories[main_category])
    quantity = random.randint(1, 5)

    total_price = random.uniform(*prices[main_category])
    chosen_drink = chosen_side = chosen_starter = chosen_dessert = None

    if random.random() < 0.6: chosen_drink = random.choice(drinks); total_price␣
 ↪+= random.uniform(*prices['Drink'])
    if random.random() < 0.5: chosen_side = random.choice(sides); total_price␣
 ↪+= random.uniform(*prices['Side'])
    if random.random() < 0.4: chosen_starter = random.choice(starters);␣
 ↪total_price += random.uniform(*prices['Starter'])
    if random.random() < 0.3: chosen_dessert = random.choice(desserts);␣
 ↪total_price += random.uniform(*prices['Dessert'])

    total_price *= quantity
    total_price = round(total_price, 2)
    order_data.append([i, customer_id, order_date, main_category, main_item,␣
 ↪chosen_drink, chosen_side, chosen_starter, chosen_dessert, quantity,␣
 ↪total_price])

# Create DataFrame
df = pd.DataFrame(order_data,␣
 ↪columns=['OrderID','CustomerID','OrderDate','MainCategory','MainItem','Drink','Side','Start
df['ComplementaryItems'] = df[['Drink','Side','Starter','Dessert']].
 ↪apply(lambda row: [x for x in row if pd.notna(x)], axis=1)

# Add time and customer features
s = pd.to_datetime(df['OrderDate'])
df['OrderTimestamp'] = s
df['Hour'] = s.dt.hour
df['DayOfWeek'] = s.dt.dayofweek
```

```
df['IsWeekend'] = (df['DayOfWeek']>=5).astype(int)

cust_stats = df.groupby('CustomerID').agg(PastOrders=('OrderID','count'),␣
 ↪AvgSpend=('TotalPrice','mean')).reset_index()
df = df.merge(cust_stats, on='CustomerID', how='left')
```

[3]:
```
df
```

[3]:
```
       OrderID                            CustomerID           OrderDate  \
0            1  137bea87-6f44-4a4a-b028-5e0497bedfbb 2025-02-07 14:49:00
1            2  9bca46d2-4cc5-461e-9595-067c0e8d97cb 2025-08-09 10:01:19
2            3  d64f6507-0f67-4e02-95ce-2c4f1f74d389 2025-09-04 06:26:43
3            4  593cdfea-f654-4cc6-a6a9-e36b42aecd97 2025-01-15 22:01:34
4            5  0ded5916-6458-4d8e-9348-a5c42a6191de 2025-04-30 02:54:03
…          …                                     …                  …
99995    99996  9dbe6f01-7c95-4ce2-a085-329f8d5793b4 2025-04-21 01:28:39
99996    99997  09631cf5-5c4f-4aa5-9b65-5ef435a02c0d 2025-08-28 02:50:59
99997    99998  7b956a47-406f-4937-85c3-2010f7d738a8 2025-09-05 12:29:00
99998    99999  6754a2dc-8c57-4eea-a03a-88881937bb4f 2025-04-25 15:18:33
99999   100000  d4f7b800-6f1d-4ddd-b1bc-500044f0f913 2025-05-27 19:00:10

         MainCategory              MainItem     Drink             Side  \
0              Kebabs         Chicken Kebab     Water     Garlic Bread
1              Kebabs             Veg Kebab     Fanta             None
2      Fish and chips  Classic Fish & Chips      None  Mashed Potatoes
3              Kebabs         Chicken Kebab     Water             None
4               Pizza           BBQ Chicken      Coke             None
…                   …                     …       …                …
99995           Pizza            Pepperoni     Fanta             None
99996           Pizza            Pepperoni     Pepsi            Salad
99997           Pasta       Penne Arrabiata  Lemonade             None
99998         Calzone     Pepperoni Calzone     Water            Fries
99999           Pasta       Penne Arrabiata     Fanta            Salad

                Starter  Dessert  Quantity  TotalPrice  \
0                  None     None         1       22.91
1                  None     None         2       15.72
2             Bruschetta     None         5      100.95
3                  None  Pudding         3       64.91
4                  None     None         4       59.27
…                     …       …         …           …
99995              None     None         4       85.93
99996      Spring Rolls     None         4       86.85
99997  Garlic Mushrooms     None         3       65.82
99998      Spring Rolls     None         5      113.50
99999              None     None         2       43.85
```

```
                  ComplementaryItems      OrderTimestamp  Hour  DayOfWeek  \
0              [Water, Garlic Bread] 2025-02-07 14:49:00    14          4
1                           [Fanta] 2025-08-09 10:01:19    10          5
2      [Mashed Potatoes, Bruschetta] 2025-09-04 06:26:43     6          3
3                   [Water, Pudding] 2025-01-15 22:01:34    22          2
4                             [Coke] 2025-04-30 02:54:03     2          2
…                                  …                  …   …        …
99995                       [Fanta] 2025-04-21 01:28:39     1          0
99996   [Pepsi, Salad, Spring Rolls] 2025-08-28 02:50:59     2          3
99997   [Lemonade, Garlic Mushrooms] 2025-09-05 12:29:00    12          4
99998   [Water, Fries, Spring Rolls] 2025-04-25 15:18:33    15          4
99999                 [Fanta, Salad] 2025-05-27 19:00:10    19          1

       IsWeekend  PastOrders  AvgSpend
0              0           1     22.91
1              1           1     15.72
2              0           1    100.95
3              0           1     64.91
4              0           1     59.27
…          …         …       …
99995          0           1     85.93
99996          0           1     86.85
99997          0           1     65.82
99998          0           1    113.50
99999          0           1     43.85

[100000 rows x 18 columns]
```

[4]:
```python
# Item-based Collaborative Filtering
from sklearn.preprocessing import MultiLabelBinarizer
from sklearn.metrics.pairwise import cosine_similarity
transactions = df.apply(lambda r: [r['MainItem']] + r['ComplementaryItems'],␣
 ↪axis=1).tolist()
mlb_items = MultiLabelBinarizer()
trans_mat = mlb_items.fit_transform(transactions)
items = list(mlb_items.classes_)
sim_matrix = cosine_similarity(trans_mat.T)
index_of = {item: idx for idx, item in enumerate(items)}
categories_split = {'Drink': drinks, 'Side': sides, 'Starter': starters,␣
 ↪'Dessert': desserts}

def recommend_itemcf(main_item, top_k=6, by_category=True):
    if main_item not in index_of: return []
    idx = index_of[main_item]
    scores = sim_matrix[idx].copy()
    scores[idx] = -1
    recs = []
```

```
    if by_category:
        for cat_name, cat_items in categories_split.items():
            candidates = [(item, scores[index_of[item]]) for item in cat_items␣
 ↪if item in index_of]
            candidates.sort(key=lambda x: x[1], reverse=True)
            if candidates and candidates[0][1] > 0: recs.
 ↪append(candidates[0][0])
    if len(recs) < top_k:
        ranked = sorted([(items[i], scores[i]) for i in range(len(items))],␣
 ↪key=lambda x: x[1], reverse=True)
        for it, sc in ranked:
            if it not in recs and sc>0: recs.append(it)
            if len(recs)>=top_k: break
    if not recs:
        from collections import Counter
        counter = Counter()
        for t in transactions:
            if main_item in t:
                for it in t:
                    if it != main_item: counter[it]+=1
        recs = [it for it,_ in counter.most_common(top_k)]
    return recs[:top_k]

print('Item-CF recommendations for Margherita:', recommend_itemcf('Margherita'))
```

```
Item-CF recommendations for Margherita: ['Water', 'Fries', 'Soup', 'Pudding',
'Fanta', 'Orange Juice']
```

[5]:
```
print('Item-CF recommendations for Veg Kebab:', recommend_itemcf('Veg Kebab'))
```

```
Item-CF recommendations for Veg Kebab: ['Lemonade', 'Mashed Potatoes', 'Garlic
Mushrooms', 'Cake', 'Garlic Bread', 'Orange Juice']
```

[6]:
```
print('Item-CF recommendations for Fettuccine Alfredo:',␣
 ↪recommend_itemcf('Fettuccine Alfredo'))
```

```
Item-CF recommendations for Fettuccine Alfredo: ['Orange Juice', 'Salad',
'Garlic Mushrooms', 'Brownie', 'Lemonade', 'Mashed Potatoes']
```

[7]:
```
# RandomForest with per-item/context-aware recommendations
from sklearn.ensemble import RandomForestClassifier
from sklearn.multioutput import MultiOutputClassifier
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MultiLabelBinarizer

X = pd.
 ↪get_dummies(df[['MainItem','Hour','DayOfWeek','IsWeekend','PastOrders','AvgSpend']])
mlb = MultiLabelBinarizer()
```

```
Y = mlb.fit_transform(df['ComplementaryItems'])
X_train,X_test,Y_train,Y_test = train_test_split(X,Y,test_size=0.
 ↪2,random_state=42)
rf =␣
 ↪RandomForestClassifier(n_estimators=200,max_depth=10,random_state=42,n_jobs=-1)
multi_rf = MultiOutputClassifier(rf)
multi_rf.fit(X_train,Y_train)
print('RandomForest training done')

def recommend_rf_per_item(main_item, hour=12, dayofweek=2, is_weekend=0,␣
 ↪past_orders=5, avg_spend=15, top_k=6):
    vec = pd.DataFrame(np.zeros((1, X.shape[1])), columns=X.columns)

    # Set the correct MainItem column
    main_col = f'MainItem_{main_item}'
    if main_col in X.columns:
        vec[main_col] = 1

    # Time features
    hour_col = f'Hour_{hour}'
    dow_col = f'DayOfWeek_{dayofweek}'
    weekend_col = f'IsWeekend_{is_weekend}'

    if hour_col in X.columns: vec[hour_col] = 1
    if dow_col in X.columns: vec[dow_col] = 1
    if weekend_col in X.columns: vec[weekend_col] = 1

    # Customer numerical features
    if 'PastOrders' in X.columns: vec['PastOrders'] = past_orders
    if 'AvgSpend' in X.columns: vec['AvgSpend'] = avg_spend

    # Predict probabilities
    proba_list = []
    for est in multi_rf.estimators_:
        try:
            p = est.predict_proba(vec)
            proba_list.append(p[:,1])
        except:
            p = est.predict(vec)
            proba_list.append(p)
    proba = np.array(proba_list).flatten()
    items = mlb.classes_
    ranked = sorted(zip(items, proba), key=lambda x:x[1], reverse=True)
    return [it for it,_ in ranked[:top_k]]
```

```
print('RandomForest recommendations for Margherita:',␣
↪recommend_rf_per_item('Margherita', hour=18, dayofweek=5, is_weekend=1,␣
↪past_orders=10, avg_spend=20))
```

```
RandomForest training done
RandomForest recommendations for Margherita: ['Mashed Potatoes', 'Pepsi',
'Orange Juice', 'Fanta', 'Coke', 'Garlic Bread']
```

[8]:
```
print('RandomForest recommendations for Pepperoni Calzone:',␣
↪recommend_rf_per_item('Pepperoni Calzone', hour=18, dayofweek=5,␣
↪is_weekend=1, past_orders=10, avg_spend=20))
```

```
RandomForest recommendations for Pepperoni Calzone: ['Orange Juice', 'Coke',
'Pepsi', 'Salad', 'Mashed Potatoes', 'Garlic Bread']
```

[9]:
```
print('RandomForest recommendations for Chicken Wrap:',␣
↪recommend_rf_per_item('Chicken Wrap', hour=18, dayofweek=5, is_weekend=1,␣
↪past_orders=10, avg_spend=20))
```

```
RandomForest recommendations for Chicken Wrap: ['Pepsi', 'Orange Juice',
'Water', 'Mashed Potatoes', 'Fanta', 'Coke']
```

[10]:
```
# XGBoost context-aware recommendations
try:
    import xgboost as xgb
    XGBOOST_AVAILABLE=True
except:
    XGBOOST_AVAILABLE=False

if XGBOOST_AVAILABLE:
    xgb_clf = xgb.XGBClassifier(eval_metric='logloss', n_jobs=-1,␣
↪random_state=42)
    multi_xgb = MultiOutputClassifier(xgb_clf)
    multi_xgb.fit(X_train,Y_train)
    print('XGBoost training done')
    def recommend_xgb_per_item(main_item, hour=12, dayofweek=2, is_weekend=0,␣
↪past_orders=5, avg_spend=15, top_k=6):
        vec = pd.DataFrame(np.zeros((1, X.shape[1])), columns=X.columns)
        main_col = f'MainItem_{main_item}'
        if main_col in X.columns: vec[main_col] = 1
        if f'Hour_{hour}' in X.columns: vec[f'Hour_{hour}'] = 1
        if f'DayOfWeek_{dayofweek}' in X.columns: vec[f'DayOfWeek_{dayofweek}']␣
↪= 1
        if f'IsWeekend_{is_weekend}' in X.columns:␣
↪vec[f'IsWeekend_{is_weekend}'] = 1
        if 'PastOrders' in X.columns: vec['PastOrders'] = past_orders
        if 'AvgSpend' in X.columns: vec['AvgSpend'] = avg_spend
        proba_list = []
```

```
        for est in multi_xgb.estimators_:
            try: p = est.predict_proba(vec); proba_list.append(p[:,1])
            except: p = est.predict(vec); proba_list.append(p)
        proba = np.array(proba_list).flatten()
        items = mlb.classes_
        ranked = sorted(zip(items, proba), key=lambda x:x[1], reverse=True)
        return [it for it,_ in ranked[:top_k]]

print('XGBoost recommendations for Margherita:',␣
 ↪recommend_xgb_per_item('Margherita', hour=18, dayofweek=5, is_weekend=1,␣
 ↪past_orders=10, avg_spend=20))
```

```
XGBoost training done
XGBoost recommendations for Margherita: ['Orange Juice', 'Chicken Wings',
'Pepsi', 'Garlic Bread', 'Onion Rings', 'Garlic Mushrooms']
```

[11]:
```
    print('XGBoost recommendations for Vegan Fish & Chips:',␣
 ↪recommend_xgb_per_item('Vegan Fish & Chips', hour=18, dayofweek=5,␣
 ↪is_weekend=1, past_orders=10, avg_spend=20))
```

```
XGBoost recommendations for Vegan Fish & Chips: ['Ice Cream', 'Coke', 'Water',
'Fries', 'Onion Rings', 'Pepsi']
```

[12]:
```
    print('XGBoost recommendations for Veggie Burger:',␣
 ↪recommend_xgb_per_item('Veggie Burger', hour=18, dayofweek=5, is_weekend=1,␣
 ↪past_orders=10, avg_spend=20))
```

```
XGBoost recommendations for Veggie Burger: ['Ice Cream', 'Garlic Bread',
'Fanta', 'Coke', 'Chicken Wings', 'Water']
```