main Food Recommendation

September 20, 2025

[1]:
```
import pandas as pd
import numpy as np
from faker import Faker
import random
from collections import defaultdict

fake = Faker()
num_orders = 100000
```

[2]:
```
# I considered these random categories as main item and recommendation items.
categories = {
    'Pizza': ['Margherita', 'Pepperoni', 'Hawaiian', 'BBQ Chicken', 'Veggie',␣
 ↪'Create Your own'],
    'Burger': ['Cheeseburger', 'Chicken Burger', 'Veggie Burger', 'Bacon␣
 ↪Burger'],
    'Pasta': ['Spaghetti Bolognese', 'Penne Arrabiata', 'Fettuccine Alfredo',␣
 ↪'Lasagna'],
    'Fish and chips': ['Small fish', 'Scampi', 'Battered Chicken', 'White␣
 ↪Pudding', 'Haggis'],
    'Kebabs': ['Chicken Kebab', 'Lamb Kebab', 'Beef Kebab', 'Veg Kebab',␣
 ↪'Donner Kebab'],
    'Wrap': ['Chicken Wrap', 'Veggie Wrap', 'Falafel Wrap', 'Donner Wrap'],
    'Calzone': ['Ham & Cheese Calzone', 'Veggie Calzone', 'Pepperoni Calzone',␣
 ↪'Donner Calzone']
}

drinks = ['Coke', 'Pepsi', 'Fanta', 'Water', 'Orange Juice', 'Lemonade', 'Irn␣
 ↪Bru']
sides = ['Fries', 'Onion Rings', 'Salad', 'Garlic Bread', 'Pickle Egg', 'Chips␣
 ↪& Cheese']
starters = ['Chicken Pakora', 'Spring Rolls', 'Chicken Wings', 'Bruschetta',␣
 ↪'Garlic Mushrooms']
desserts = ['Ice Cream', 'Brownie', 'Chocolate Cake', 'Waffle', 'Cookie dough']

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)
}
```

[3]:
```
# Now we generate dataset randomly.
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])
```

[4]:
```
# 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')
```

[5]:
```
df
```

[5]:
```
       OrderID                            CustomerID           OrderDate  \
0            1  8673bb76-8db5-408a-82cc-c82c872a78cd 2025-06-11 22:28:35
1            2  980d52a8-5702-48da-bcf9-250ffc336887 2025-04-10 23:28:09
2            3  5e99affb-2f4e-46a3-8895-3632dbd7df93 2025-07-31 06:55:11
3            4  a1cee434-40c9-4f29-b259-2e9851b3fe33 2025-08-26 00:21:03
4            5  a7053d29-8a64-414d-8613-f8ef3d4f9aa9 2025-04-25 00:38:13
…          …                                     …                   …
99995    99996  ea7d379a-ec16-4b3a-93dd-971447971414 2025-08-22 18:14:10
99996    99997  4117df1b-c1b0-4d8a-9b28-c9068a1085be 2025-06-18 08:16:00
99997    99998  0b5cf11e-29b1-4e72-b122-ce4a943effbb 2025-03-14 15:46:44
99998    99999  c7d172d5-3f73-4e47-93b4-56fbb9b82cf2 2025-06-08 19:07:54
99999   100000  9849a773-010f-4dc2-8336-2e2c15baeb10 2025-08-21 12:15:33

         MainCategory              MainItem          Drink             Side  \
0      Fish and chips            Small fish       Lemonade             None
1               Pasta    Fettuccine Alfredo   Orange Juice             None
2              Burger          Cheeseburger        Irn Bru             None
3                Wrap          Chicken Wrap       Lemonade             None
4      Fish and chips                Haggis           None       Pickle Egg
…                 …                     …             …                …
99995          Burger         Veggie Burger       Lemonade   Chips & Cheese
99996         Calzone   Ham & Cheese Calzone          None      Onion Rings
99997            Wrap           Donner Wrap           Coke             None
99998           Pizza       Create Your own           None            Salad
99999          Burger          Bacon Burger          Water             None

              Starter        Dessert  Quantity  TotalPrice  \
0                None   Cookie dough         3       61.60
1      Chicken Pakora           None         4       68.22
2                None           None         2       22.22
3      Chicken Pakora           None         4       69.51
4          Bruschetta   Cookie dough         1       27.04
…                 …              …        …          …
99995    Spring Rolls           None         2       47.85
99996    Spring Rolls           None         3       53.89
99997            None           None         2       30.61
99998            None         Waffle         2       48.69
```

```
99999            None          None         4       49.11

                          ComplementaryItems      OrderTimestamp  Hour  \
0                    [Lemonade, Cookie dough] 2025-06-11 22:28:35    22
1               [Orange Juice, Chicken Pakora] 2025-04-10 23:28:09    23
2                                   [Irn Bru] 2025-07-31 06:55:11     6
3                   [Lemonade, Chicken Pakora] 2025-08-26 00:21:03     0
4        [Pickle Egg, Bruschetta, Cookie dough] 2025-04-25 00:38:13     0
…                                          …                 …   …
99995  [Lemonade, Chips & Cheese, Spring Rolls] 2025-08-22 18:14:10    18
99996                [Onion Rings, Spring Rolls] 2025-06-18 08:16:00     8
99997                                     [Coke] 2025-03-14 15:46:44    15
99998                             [Salad, Waffle] 2025-06-08 19:07:54    19
99999                                    [Water] 2025-08-21 12:15:33    12

       DayOfWeek  IsWeekend  PastOrders  AvgSpend
0              2          0           1     61.60
1              3          0           1     68.22
2              3          0           1     22.22
3              1          0           1     69.51
4              4          0           1     27.04
…            …          …          …       …
99995          4          0           1     47.85
99996          2          0           1     53.89
99997          4          0           1     30.61
99998          6          1           1     48.69
99999          3          0           1     49.11

[100000 rows x 18 columns]
```

[6]:
```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}
```

[7]:
```
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]
```

[8]:
```
print('Item-CF recommendations for Margherita:', recommend_itemcf('Margherita'))
```

```
Item-CF recommendations for Margherita: ['Pepsi', 'Fries', 'Chicken Wings', 'Ice
Cream', 'Lemonade', 'Orange Juice']
```

[9]:
```
print('Item-CF recommendations for Veg Kebab:', recommend_itemcf('Veg Kebab'))
```

```
Item-CF recommendations for Veg Kebab: ['Lemonade', 'Onion Rings', 'Bruschetta',
'Cookie dough', 'Pepsi', 'Water']
```

[10]:
```
print('Item-CF recommendations for Fettuccine Alfredo:',␣
↪recommend_itemcf('Fettuccine Alfredo'))
```

```
Item-CF recommendations for Fettuccine Alfredo: ['Fanta', 'Salad', 'Bruschetta',
'Waffle', 'Lemonade', 'Chips & Cheese']
```

[11]:
```
# Now Train dataset by RandomForest Classification
from sklearn.ensemble import RandomForestClassifier
from sklearn.multioutput import MultiOutputClassifier
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MultiLabelBinarizer
```

[12]:
```
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)
```

[12]:
```
MultiOutputClassifier(estimator=RandomForestClassifier(max_depth=10,
                                                       n_estimators=200,
                                                       n_jobs=-1,
                                                       random_state=42))
```

[13]:
```
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]]
```

Now predict the items for some example:

[14]:
```
print('RandomForest recommendations for Margherita:',␣
  ↪recommend_rf_per_item('Margherita', hour=18, dayofweek=5, is_weekend=1,␣
  ↪past_orders=10, avg_spend=20))
```

```
RandomForest recommendations for Margherita: ['Ice Cream', 'Fries', 'Orange
Juice', 'Chicken Pakora', 'Water', 'Chicken Wings']
```

[15]:
```
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: ['Lemonade', 'Pickle Egg',
'Chicken Pakora', 'Coke', 'Orange Juice', 'Fries']
```

[16]:
```
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: ['Orange Juice', 'Garlic
Mushrooms', 'Fanta', 'Coke', 'Chicken Pakora', 'Water']
```

[17]:
```
# Now Train dataset by XGBoost
import xgboost as xgb
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)
```

[17]:
```
MultiOutputClassifier(estimator=XGBClassifier(base_score=None, booster=None,
                                              callbacks=None,
                                              colsample_bylevel=None,
                                              colsample_bynode=None,
                                              colsample_bytree=None,
                                              device=None,
                                              early_stopping_rounds=None,
                                              enable_categorical=False,
                                              eval_metric='logloss',
                                              feature_types=None,
                                              feature_weights=None, gamma=None,
                                              grow_policy=None,
                                              importance_type=None,
                                              interaction_constraints=None,
                                              learning_rate=None, max_bin=None,
                                              max_cat_threshold=None,
                                              max_cat_to_onehot=None,
                                              max_delta_step=None,
                                              max_depth=None, max_leaves=None,
                                              min_child_weight=None,
                                              missing=nan,
                                              monotone_constraints=None,
                                              multi_strategy=None,
                                              n_estimators=None, n_jobs=-1,
                                              num_parallel_tree=None, …))
```

[18]:
```
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]]
```

Now predict some items

[19]:
```
print('XGBoost recommendations for Margherita:',␣
 ↪recommend_xgb_per_item('Margherita', hour=18, dayofweek=5, is_weekend=1,␣
 ↪past_orders=10, avg_spend=20))
```

```
XGBoost recommendations for Margherita: ['Fries', 'Lemonade', 'Ice Cream',
'Chicken Wings', 'Garlic Mushrooms', 'Orange Juice']
```

[20]:
```
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: ['Fanta', 'Lemonade', 'Fries',
'Ice Cream', 'Salad', 'Pickle Egg']
```

[21]:
```
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: ['Spring Rolls', 'Chicken Pakora',
'Lemonade', 'Fanta', 'Salad', 'Cookie dough']
```