ML CHALLENGE CSC 311

I. Data Exploration and Feature Selection

The collected dataset consists of responses to a survey about different food items (Pizza, Shawarma, and Sushi). Each row represents a respondent's answer to various questions regarding food complexity, ingredient count, setting, pricing, associations with movies, drinks, people, and hot sauce preference. The dataset includes **1,644 samples**, evenly distributed across the three food types.

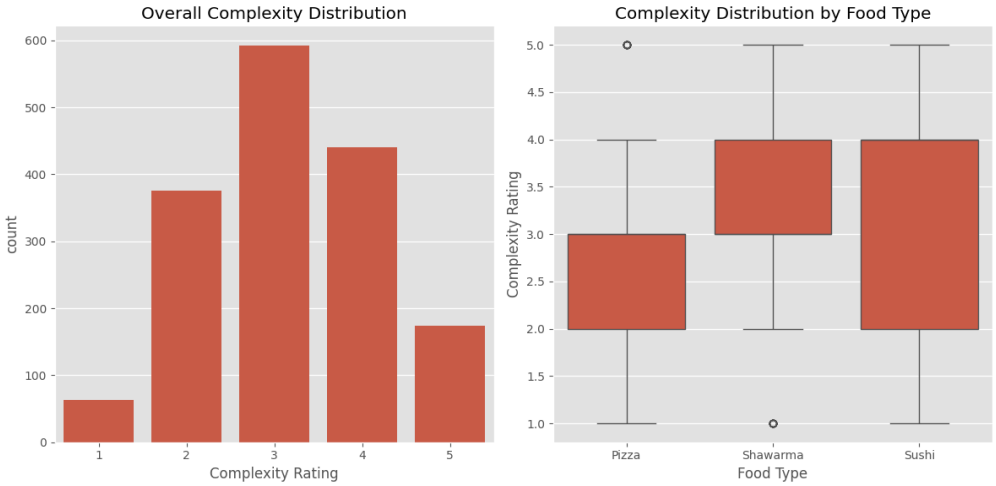
The dataset contains the following columns:

- `Q1`: Complexity rating (1-5).
- `Q2`: Expected ingredient count (numeric).
- `Q3`: Expected setting (categorical, multi-label).
- `Q4`: Expected price (numeric).
- `Q5`: Associated movie (categorical, open-ended).
- `Q6`: Suggested drink pairing (categorical).
- `Q7`: Associated person (categorical).
- `Q8`: Hot sauce preference (ordinal categorical).
- `Label`: The target variable (Pizza, Shawarma, or Sushi).

Initially, we noticed that "None" responses were being counted as missing (`NaN`). To ensure accurate analysis, we reloaded the data without converting "None" to `NaN`. After this adjustment, **we confirmed there are no missing values in any column**.

Complexity

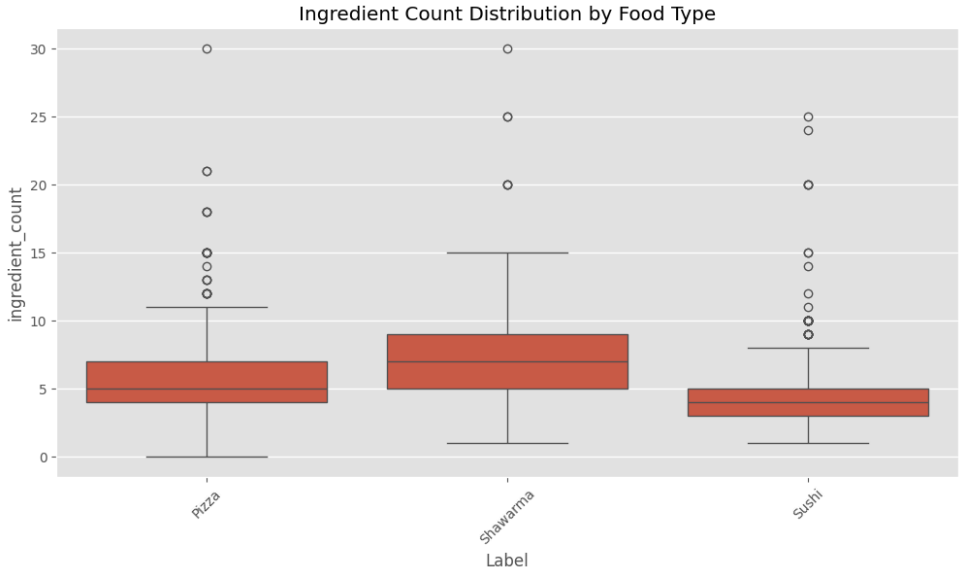
The complexity ratings vary significantly across food types. **Pizza** is consistently rated as the simplest, with the lowest mean complexity (2.88) and the tightest spread (std = 0.79), indicating strong agreement among respondents. **Shawarma** falls in the middle, with a slightly higher mean complexity (3.23) and greater variability (std = 0.94). **Sushi** is perceived as the most complex, with the highest mean (3.42) and median (4.0), as well as the widest disagreement among respondents (std = 1.21), suggesting differing opinions on its difficulty.



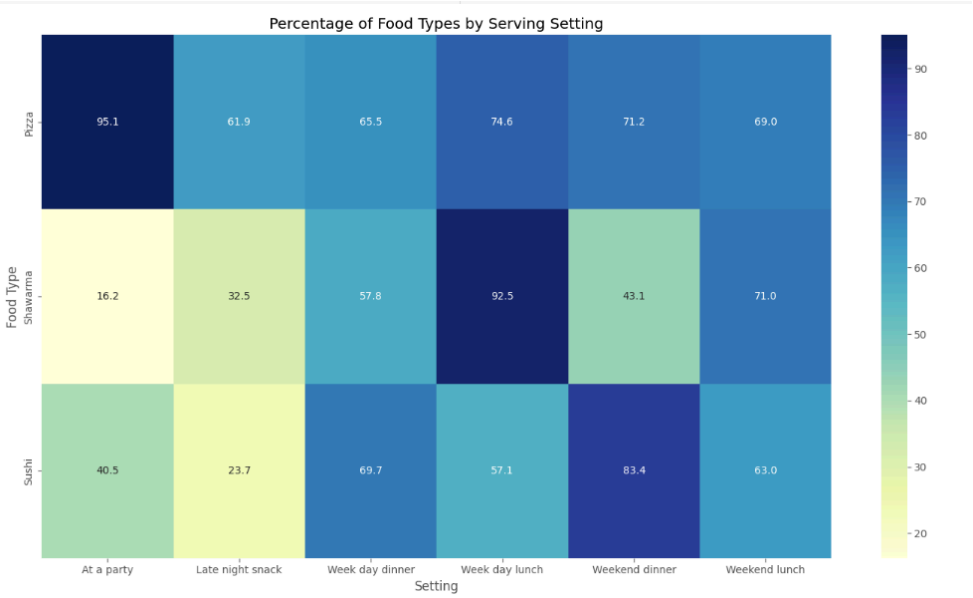
Ingredients

Ingredient counts were highly variable, with responses in different formats, including single numbers, ranges (e.g., "5-10"), and even text-based numbers ("four"). To standardize, we used **regex parsing** to extract numeric values, converting number words, averaging ranges (e.g., "5-7" → **6**), and handling anomalies like excessively high values (e.g., **810 ingredients**). We capped ingredient counts at **30** and rounded fractional values.

After cleaning, we observed a **clear separation** between food types, making ingredient count a strong predictive feature. **Shawarma** had the highest median ingredient count (**7**), followed by **Pizza** (**5**) and **Sushi** (**4**). The interquartile range (IQR) further highlights these differences: **Pizza** typically has **4-7** ingredients, **Shawarma 5-9**, and **Sushi 3-5**, reinforcing the distinction between categories.



Setting



Participants were asked in which settings they would expect each food to be served. Responses were **multi-select**, meaning each food could be associated with multiple settings. To make this data usable for modeling, we applied **binary encoding**, where each setting is represented as a separate column with `1` if the food was selected for that setting and `0` otherwise. This preserves all information while enabling the model to recognize associations effectively.

The distributions reveal interesting trends: **Pizza** is the most versatile, being highly associated with all settings, especially **parties (95.1%)** and **late-night snacks (61.9%)**. **Shawarma** is most linked to **weekday lunches (92.5%)**, while **Sushi** is strongly associated with **weekend dinners (83.4%)**. These patterns suggest that food type influences its expected dining context, making this a valuable feature for prediction.

Price

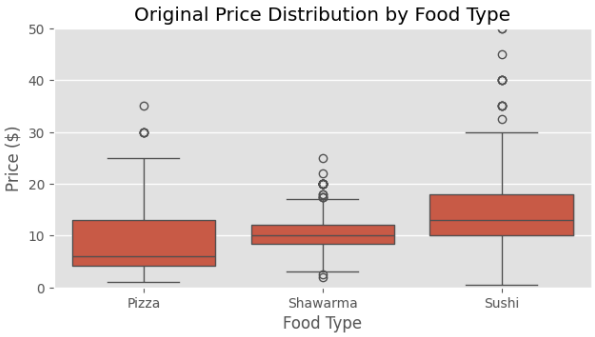
Cleaning the price data was the most challenging step due to the significant variation in how participants reported their expected costs. Many responses contained non-standard formats, including currency symbols, ranges (`$10-$15`), and references to whole items rather than single servings (`$12 per serving`, `$20 for a small platter of sushi`).

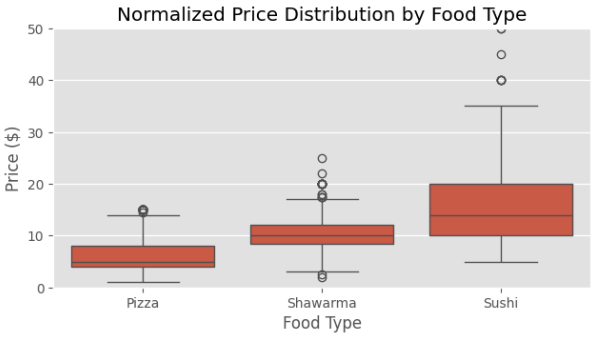
To standardize prices, we first removed any currency symbols (`$`, `CAD`, `dollars`) and extracted numeric values. For responses that contained a range (e.g., "$10-$15"), we took the average (`$12.5`). If a participant provided a descriptive response (e.g., “Around 15 dollars without tax”), we extracted the relevant number and ignored the rest.

Once we had the numerical values, we needed to determine whether the reported price referred to a **single serving** or a **whole item**. This was done using keyword detection:

- **Whole items**: Detected through words like “whole,” “entire,” “full,” “large,” and specific phrases like “whole pizza” or “entire roll.”
- **Single servings**: Indicated by terms such as “slice,” “piece,” “per serving,” or “single.”
- **Food-specific distinctions**:
    - **Pizza**: “Slice” meant a single serving, while “whole pizza” indicated a full item.
    - **Sushi**: “Piece” or “nigiri” suggested a per-piece price, whereas “roll” implied a full roll.
    - **Shawarma**: The data was more consistent, with most responses already referring to a single serving.

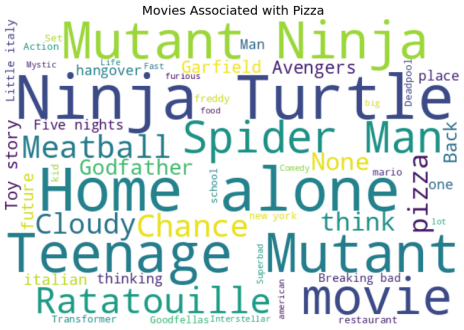
Before normalization, the average reported price for pizza was **$8.79**, but this included whole pizza prices. After adjusting for portion size, the normalized mean price dropped to **$6.53 per slice**, reducing variance. Sushi saw a slight increase in price after normalizing per serving, rising from **$14.34** to **$15.16**, as low per-piece prices were adjusted to reflect a standard serving size. Shawarma remained stable at **$10.66 per serving**.

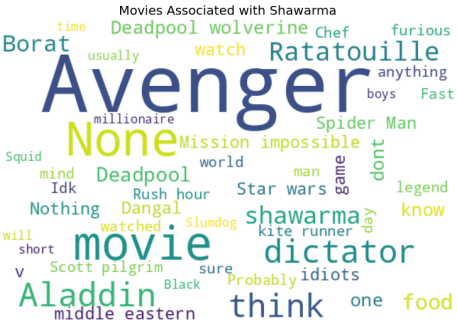


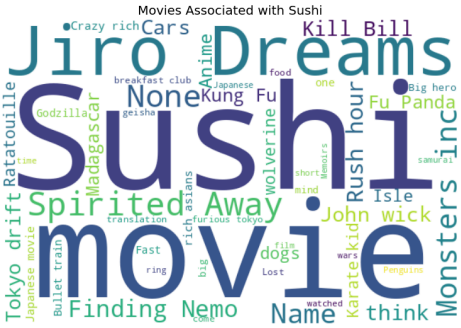


Movie and Drink Pairings

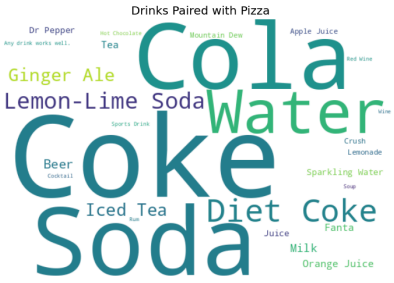
We standardized the movie data to address variations in formatting and capitalization, resulting in 585 unique movies. The top movies associated with pizza were "The Avengers" (244 mentions), "None" (85 mentions), "Teenage Mutant Ninja Turtles" (57 mentions), and "Home Alone" (55 mentions).

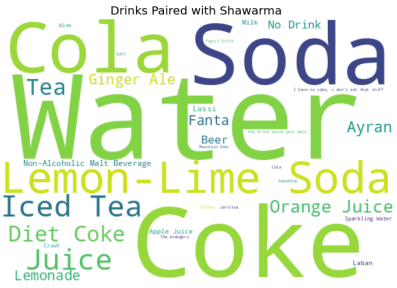


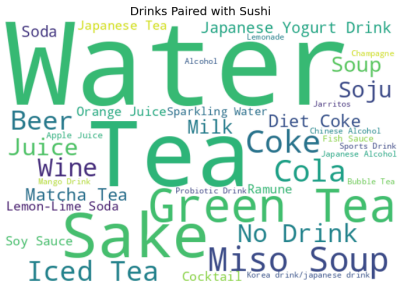




A similar approach was applied to standardize the drinks data. We note that we had to map drinks to address inconsistencies due to different ways of naming drinks (such as variations of "Coke"), so we implemented a mapping strategy. This involved standardizing similar drink names into unified categories. For instance, we mapped all variations of "Coca Cola" (e.g., 'coca cola', 'cocacola', 'coca') to "Cola," and different references to "Soda" (e.g., 'soda', 'pop', 'soft drink') to "Soda." This process reduced the number of unique values, bringing the total to 60 distinct drink categories.







Due to the high cardinality in movies and drinks, which have many unique values, directly applying one-hot encoding would drastically increase the dimensionality, leading to the curse of

dimensionality. This issue arises when features contain many unique categories, potentially leading to sparse representations and overfitting. We first capped the movies at a threshold of 5 mentions, so any movie with less than 5 mentions will be under the ‘Other’ category. A similar thing was done for drinks. This resulted in 25 unique drinks and 52 unique movies.

Then we used **target encoding**, which replaces each category with the average target value for that category. This helps capture relationships without exploding the feature space, which would occur with one-hot encoding. To prevent overfitting, particularly with rare categories, we applied **smoothing**, combining the category’s mean target value with the global mean target value, weighted by category frequency.

We applied **k-fold cross-validation** to ensure robustness and avoid data leakage. For each fold, we split the data into training and test sets, using the training set to calculate the smoothed probabilities for each category. In each fold, the target encoding for a category was computed based on the training data only, so test data from the current fold was never used to influence the encoding. This way, the model is trained on data that closely resembles the unseen data it will encounter in production. Additionally, categories not seen in the training set were assigned the global mean target values to handle new categories effectively. This method ensures that the model generalizes well while reducing the risk of overfitting.

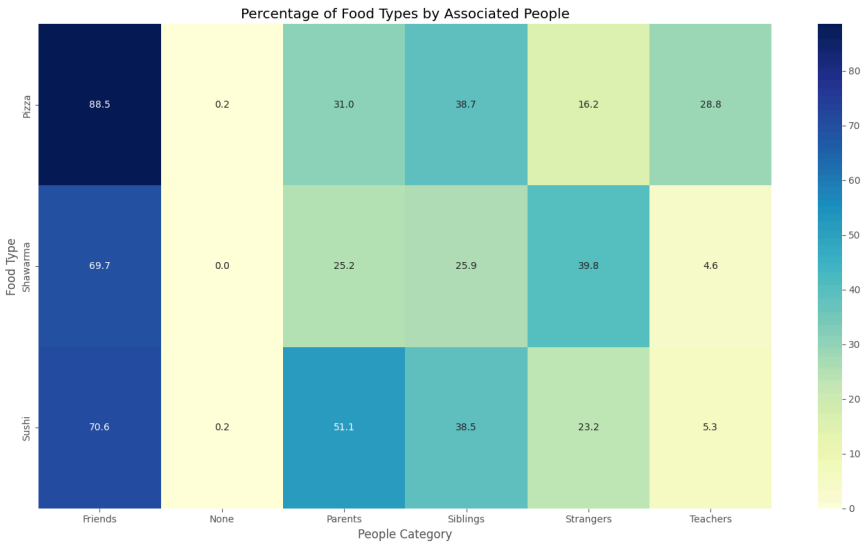
For example, in our movies target encoding "Teenage Mutant Ninja Turtles" shows a strong correlation with pizza (88.5%), while "Spider-Man" is associated with pizza (69.8%) but also shows significant associations with shawarma (16.7%) and sushi (13.5%). On the other hand, "The Avengers" is strongly linked to shawarma (92.5%), likely due to the famous post-credits scene, while Japanese films like "Jiro Dreams of Sushi" show a high association with sushi (82%). These patterns confirm that pizza is prevalent in American family films, sushi aligns with Japanese-themed content, and shawarma is strongly associated with Middle Eastern films like "Aladdin" (72.2%) and "The Dictator" (72.2%). This targeted encoding strategy ensured that we handle the complexities of high-cardinality features efficiently and preserve meaningful associations.

```
Teenage Mutant Ninja Turtles {'Pizza': 0.8850574712643678, 'Shawarma': 0.057471264367816084, 'Sushi': 0.057471264367816084}
Spider-Man {'Pizza': 0.6979166666666666, 'Shawarma': 0.16666666666666666, 'Sushi': 0.13541666666666666}
Other {'Pizza': 0.3509732360097324, 'Shawarma': 0.29440389294403896, 'Sushi': 0.35462287104622875}
None {'Pizza': 0.24796747967479674, 'Shawarma': 0.491869918699187, 'Sushi': 0.26016260162601623}
The Avengers {'Pizza': 0.053968253968253964, 'Shawarma': 0.9253968253968254, 'Sushi': 0.020634920634920634}
Isle of dogs {'Pizza': 0.2222222222222222, 'Shawarma': 0.2222222222222222, 'Sushi': 0.5555555555555555}
Kung Fu Panda {'Pizza': 0.17543859649122806, 'Shawarma': 0.17543859649122806, 'Sushi': 0.6491228070175438}
Aladdin {'Pizza': 0.13888888888888887, 'Shawarma': 0.7222222222222222, 'Sushi': 0.13888888888888887}
```

People association

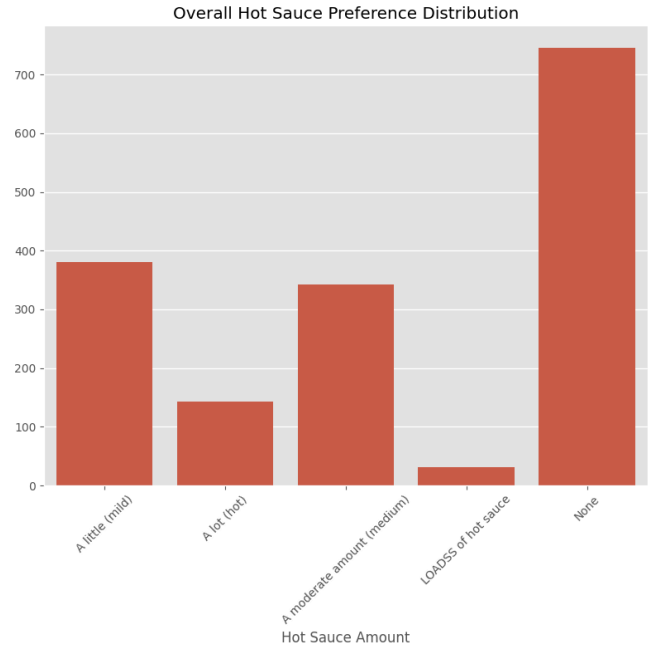
For the people association data, we used a binary one-hot encoding approach to capture the relationship between different food items and the categories of people with whom they are most commonly associated. The categories include 'Friends,' 'None,' 'Parents,' 'Siblings,' 'Strangers,' and 'Teachers.' These insights were gathered to evaluate whether people associations could be a good predictive feature for food items.

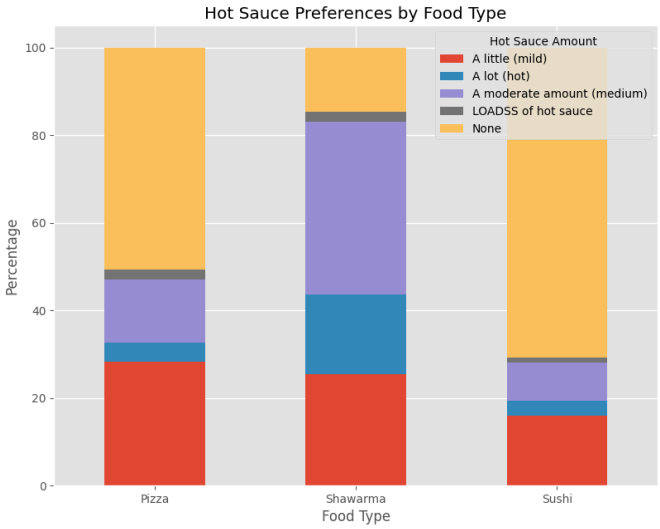
Pizza is strongly associated with **friends** (88.5%), followed by **siblings** (38.7%) and **parents** (31.0%), making it a highly social and family-friendly food. **Shawarma** shows a similar trend with **friends** (69.7%) but also has a significant association with **strangers** (39.8%), suggesting it may be viewed as a convenient street food. **Sushi**, on the other hand, has the highest association with **parents** (51.1%), indicating it might be perceived as a more refined or family-oriented dining choice. These patterns suggest that people associations could be a valuable feature to predict food preferences, with social and cultural contexts in determining how food items are perceived and enjoyed across different relationships and settings



Hot sauce

Sushi stands out with the highest proportion of "none" responses, indicating that most people avoid adding hot sauce to it. In contrast, shawarma shows the highest usage of hot sauce, with preferences ranging from medium to hot and even "loads of hot sauce," reflecting its spicy street food appeal. Pizza falls in the middle, with a mix of mild, medium, and some hot sauce preferences, though a notable portion still opts for no sauce at all. These trends align with the typical flavor profiles and cultural associations of each dish.

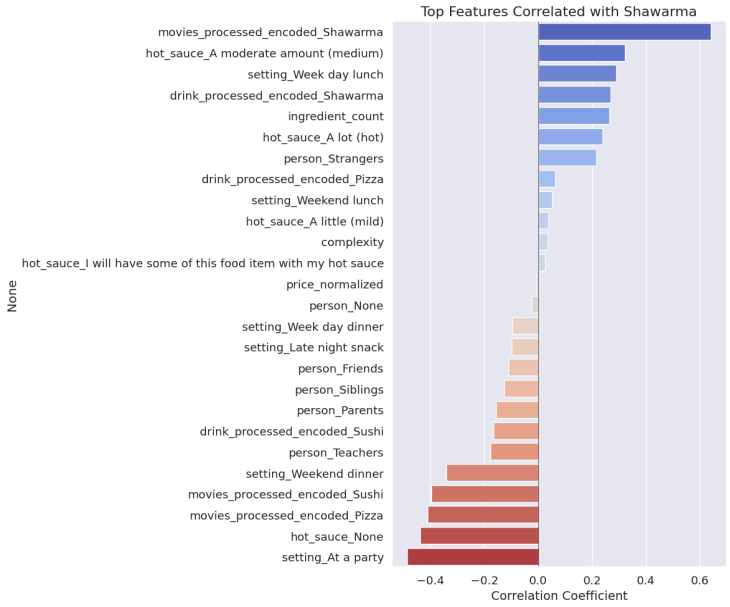


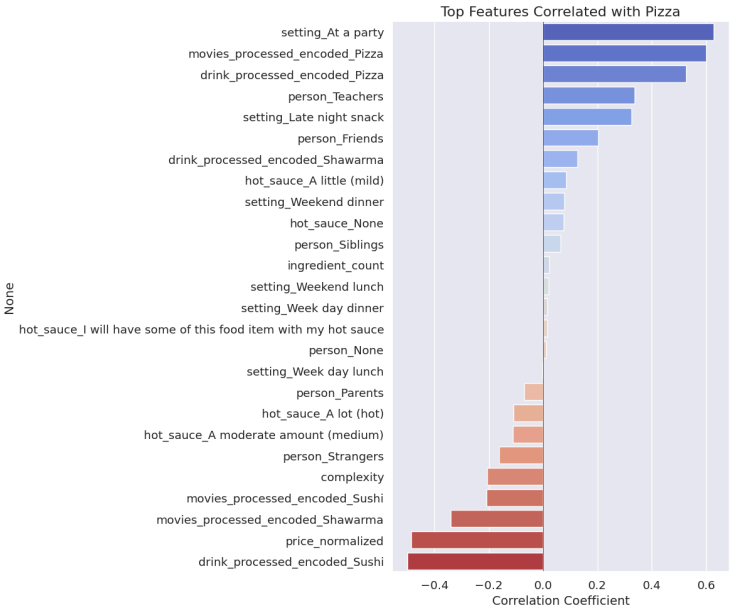


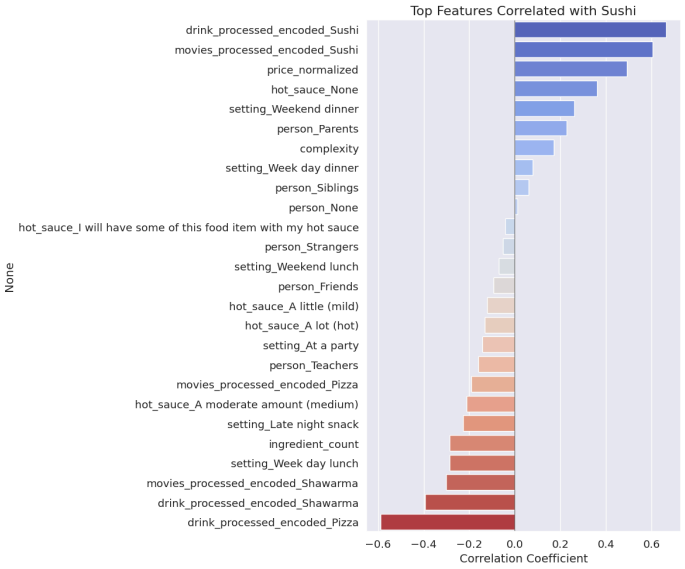
Correlation Analysis

To better understand which features are most relevant for predicting food preferences, we conducted a correlation analysis between numerical features and food labels. By converting categorical labels into a one-hot encoded format, we could assess how strongly each feature was associated with different food items. This helped us identify the most informative variables while filtering out less relevant ones.

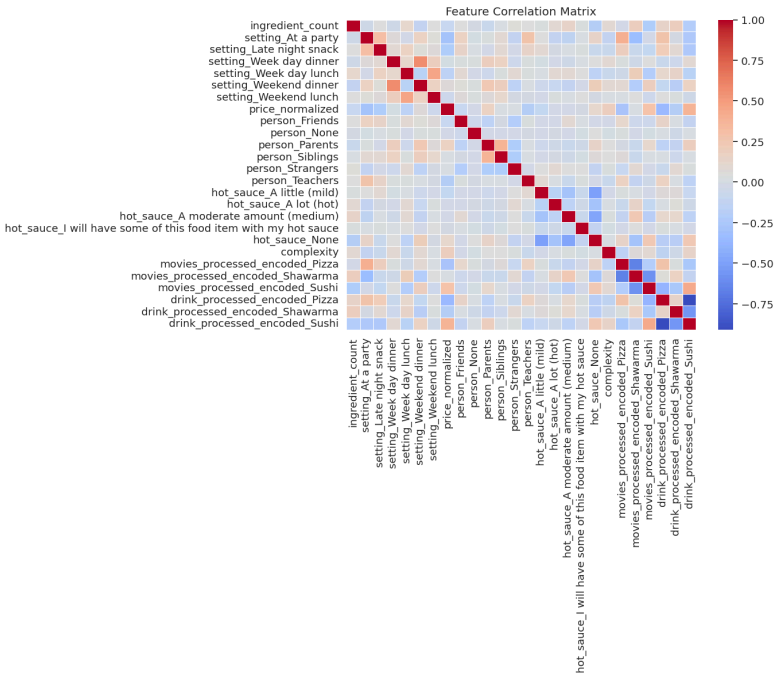
We then ranked features based on their correlation strength, highlighting the top positive and negative associations for each food category. This allowed us to see which factors—such as social context, hot sauce preferences, or other attributes—had the greatest influence on food choices.


Top Features Correlated with Shawarma
movies_processed_encoded_Shawarma
hot_sauce_A moderate amount (medium)
setting_Week day lunch
drink_processed_encoded_Shawarma
ingredient_count
hot_sauce_A lot (hot)
person_Strangers
drink_processed_encoded_Pizza
setting_Weekend lunch
hot_sauce_A little (mild)
complexity
hot_sauce_I will have some of this food item with my hot sauce
price_normalized
person_None
setting_Week day dinner
setting_Late night snack
person_Friends
person_Siblings
person_Parents
drink_processed_encoded_Sushi
person_Teachers
setting_Weekend dinner
movies_processed_encoded_Sushi
movies_processed_encoded_Pizza
hot_sauce_None
setting_At a party
None
−0.4
−0.2
0.0
0.2
0.4
0.6
Correlation Coefficient

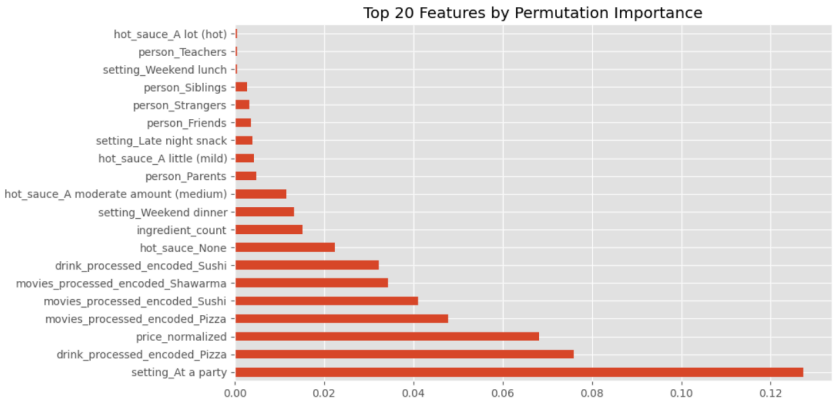

Top Features Correlated with Pizza
setting_At a party
movies_processed_encoded_Pizza
drink_processed_encoded_Pizza
person_Teachers
setting_Late night snack
person_Friends
drink_processed_encoded_Shawarma
hot_sauce_A little (mild)
setting_Weekend dinner
hot_sauce_None
person_Siblings
ingredient_count
setting_Weekend lunch
setting_Week day dinner
hot_sauce_I will have some of this food item with my hot sauce
person_None
setting_Week day lunch
person_Parents
hot_sauce_A lot (hot)
hot_sauce_A moderate amount (medium)
person_Strangers
complexity
movies_processed_encoded_Sushi
movies_processed_encoded_Shawarma
price_normalized
drink_processed_encoded_Sushi
None
−0.4
−0.2
0.0
0.2
0.4
0.6
Correlation Coefficient



To ensure that our features were independent and not redundant, we analyzed the correlation between them. First, we separated the target variable from the features and applied one-hot encoding to convert categorical data into a numerical format. This allowed us to calculate a correlation matrix, visualized through a heatmap, to observe relationships between different features.We then identified pairs of highly correlated features (correlation above 0.8). If two features were strongly correlated, it could indicate redundancy, meaning one of them might be removed to simplify the model without losing valuable information.



We found a strong negative correlation (-0.910) between **drink_processed_encoded_Sushi** and **drink_processed_encoded_Pizza**, indicating that people who associated certain drinks with sushi almost never did so with pizza, and vice versa. This likely reflects distinct cultural and habitual pairings—sushi is commonly enjoyed with beverages like tea or sake, while pizza is often paired with soda or beer. Since these features provide nearly identical but inverse information, keeping both may introduce redundancy and multicollinearity, which could impact the model's performance. To simplify the model without losing predictive power, we dropped one of these features, **drink_processed_encoded_Sushi**, ensuring that the retained feature still captures the necessary variation in drink preferences.



Amongst all the features, **complexity** has the highest correlation(0.179), which is still weak, with **price_normalized** indicating that the change of value in one does not signifcantly affect the other. However, the permutation importance of **price_normalized** is much higher than that of **complexity,** indicating **price_normalized** contributes more significance in predicting the target value. Because of the weak correlation and low permutation importance, we dropped **complexity** from our final list of input features.

Data split

In this project, we divided the dataset into three distinct parts: training, validation, and testing, each serving a specific role in building a reliable prediction model. We allocated 70% of the data for testing, which allows us to evaluate the model's performance on unseen data. This is crucial for simulating real-world situations where the model will need to make predictions on new information it has never encountered before. A larger proportion for testing provides a more stable and trustworthy assessment of the model's ability to generalize. Additionally, 20% of the data was used for validation during the training process. The validation set helped us fine-tune the model's parameters and adjust hyperparameters to avoid overfitting. By doing so, we

ensured that the model does not simply memorize the training data but learns patterns that are useful when dealing with new inputs. Finally, 10% of the data was reserved as a generalization set. This portion remained untouched during both training and validation, allowing us to make a final unbiased evaluation of the model's ability to generalize beyond the data it has seen during development. By following this data splitting strategy, we enhanced the credibility and robustness of our prediction results

II. Model

The following three models were explored to predict our target values of food item

1. **Random Forest Classifier**
2. **kNN Classifier**
3. **MLP classifier**

All of these models were trained on the whole dataset, except for the columns: **complexity** and **drink_processed_encoded_Sushi**, which we decided to drop while exploring the data. The encoding of the categorical features were done as shown in data exploration.

III. Model Choice and Hyperparameters

Prior to training, the dataset had already been divided into training and validation sets. As a result, all models were trained using the same training set and evaluated using the same validation set, allowing for consistent comparison of their evaluation metrics. The objective for our final model was to reduce both variance and error, while preventing overfitting. We utilized the estimate_variance() function from lab 7 and experimented with different hyperparameters for each model, plotting graphs for the training error, validation error, and variance.

For each model, we tuned a different set of hyperparameters. For the **Random Forest Classifier**, we tuned the number of trees (ex. n_estimators=5), the maximum depth of each tree (ex. max_depth=5), and the minimum samples per leaf (ex. min_samples_leaf=2) to balance simplicity and interpretability while avoiding overfitting. For the **kNN Classifier**, we tuned the number of neighbors (ex. n_neighbors=1), the distance metric (ex. metric='euclidean'), and the weighting strategy (ex. weights='uniform'). For the **MLP Classifier**, we tuned the architecture (ex. hidden layer sizes (128, 64, 32)), the activation function (ex. activation='sigmoid'), the batch size (ex. batch_size=64), the number of training iterations (ex. epochs=30), and the optimizer (ex. optimizer='sgd'). Unlike the other models, the MLP’s hyperparameters are critical to avoid overfitting due to its high flexibility, which is why choices like layer sizes and activation functions must be carefully optimized.

To determine the best combination of hyperparameter values for each model, we used the GridSearchCV function imported from sklearn. Instead of using multiple nested loops to test different values for each hyperparameter of a model, GridSearchCV automates this task for us.

For example, in KNN we used the GridSearchCV for hyperparameter tuning, then we used the model attributes best_params_ to determine the hyperparameter for the model. The hyperparameters we want to evaluate are specified in a dictionary format. Each key represents a hyperparameter name, and its associated value is a list containing the different parameter values to test. GridSearchCV employs cross-validation to assess model performance and identify the optimal parameters. Below, we can see an example of how GridSearchCV was used to find hyperparameters for our kNN model.

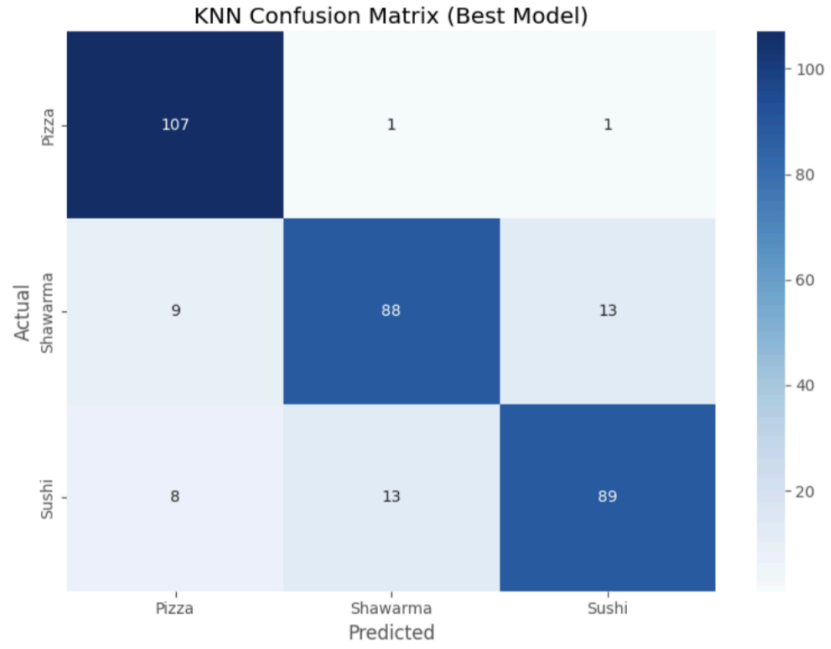
```
# Set up the parameter grid to search over
param_grid = {
    'n_neighbors': [1, 3, 5, 7, 9, 11, 13, 15],  # Values of k to test
    'metric': ['euclidean', 'manhattan', 'minkowski'],  # Different distance metrics
    'weights': ['uniform', 'distance']  # Weighting methods
}

knn = KNeighborsClassifier()

grid_search = GridSearchCV(knn, param_grid, cv=5, scoring='accuracy', n_jobs=-1, verbose=1)
grid_search.fit(X_train_scaled, y_train)

print("Best Hyperparameters: ", grid_search.best_params_)
```

The GridSearchCV will print all the Mean Test Score for each combination of hyperparameters to determine the optimal one

```
Mean Test Score: 0.8555, Params: {'metric': 'euclidean', 'n_neighbors': 13, 'weights': 'uniform'}
Mean Test Score: 0.8578, Params: {'metric': 'euclidean', 'n_neighbors': 13, 'weights': 'distance'}
Mean Test Score: 0.8548, Params: {'metric': 'euclidean', 'n_neighbors': 15, 'weights': 'uniform'}
Mean Test Score: 0.8563, Params: {'metric': 'euclidean', 'n_neighbors': 15, 'weights': 'distance'}
Mean Test Score: 0.8259, Params: {'metric': 'manhattan', 'n_neighbors': 1, 'weights': 'distance'}
Mean Test Score: 0.8555, Params: {'metric': 'manhattan', 'n_neighbors': 3, 'weights': 'distance'}
```

And finally, it will indicate the most optimal hyperparameters for each model. For example, in kNN, the best hyperparameters were found to be using the Manhattan distance metric (metric='manhattan'), 13 neighbors (n_neighbors=13), and distance-based weighting (weights='distance'), which achieved an accuracy of 86.32%

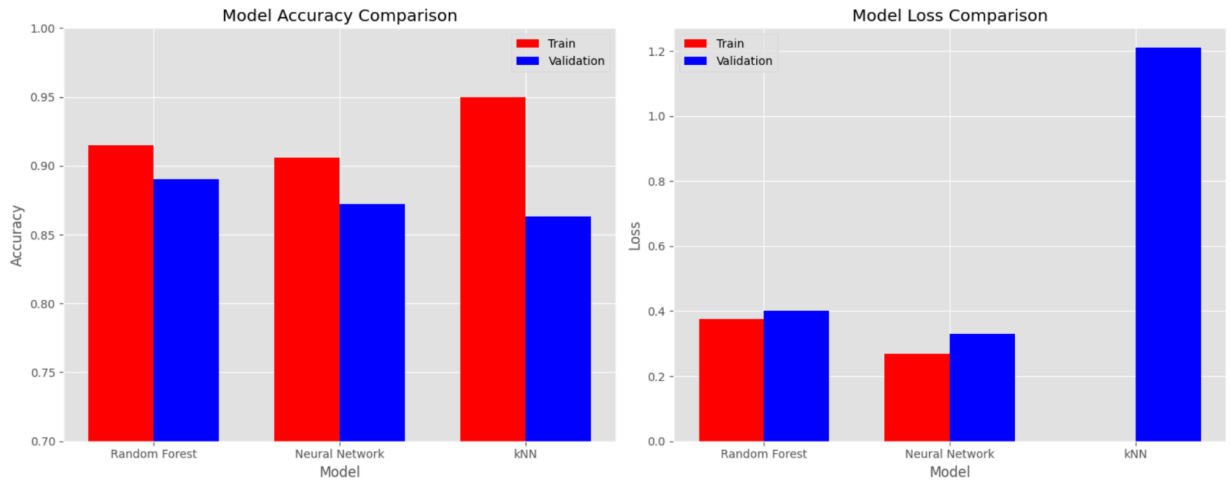
```
Best Hyperparameters:  {'metric': 'manhattan', 'n_neighbors': 13, 'weights': 'distance'}
Best KNN Accuracy: 0.8632

Best KNN Classification Report:
              precision    recall  f1-score   support

       Pizza       0.86      0.98      0.92       109
    Shawarma       0.86      0.80      0.83       110
       Sushi       0.86      0.81      0.84       110

    accuracy                           0.86       329
   macro avg       0.86      0.86      0.86       329
weighted avg       0.86      0.86      0.86       329
```



We evaluate models holistically, balancing validation performance with generalization. While validation error helps identify optimal hyperparameters and predict real-world accuracy, we equally prioritize low variance to ensure robustness across all test scenarios. This dual focus prevents overfitting—where high training accuracy becomes misleading—and instead delivers a model that performs consistently well on unseen data.

The following visualizations provide a comprehensive comparison of model performance across three machine learning approaches: Random Forest, Neural Network, and k-Nearest Neighbors. The analysis examines two critical evaluation metrics - accuracy and loss - to assess both predictive capability and generalization behavior. The accuracy comparison reveals how frequently each model makes correct predictions, contrasting performance on training data versus validation data to identify potential overfitting. Meanwhile, the loss comparison quantifies the magnitude of prediction errors, with lower values indicating better alignment with true labels and smaller discrepancies between training and validation loss suggesting more stable models. The side-by-side presentation of accuracy and loss metrics enables a balanced evaluation of

each model's effectiveness in both learning patterns from the training data and maintaining robust performance on unseen validation data.



Our evaluation of the three models reveals critical insights into their predictive performance and generalization capabilities. The Random Forest model demonstrates exceptional balance, achieving a training accuracy of 0.915 and validation accuracy of 0.890, complemented by consistently low loss values (0.375 training, 0.400 validation). This minimal performance gap indicates robust generalization without overfitting. The Neural Network shows slightly diminished but still competitive results, with 0.906 training accuracy and 0.872 validation accuracy, though its loss values (0.269 training, 0.329 validation) suggest mild overfitting. Notably, the kNN model performs better than initially indicated - while its near perfect training accuracy (0.95) and low training loss (0.000) initially suggested severe overfitting, its validation accuracy of 0.8632 and loss of 1.210 reveal it actually achieves reasonable (though relatively lower) predictive performance on unseen data compared to the other models.

Random Forest stands out as the best-performing model overall, delivering both high accuracy (0.890 validation) and exceptional stability (minimal loss gap). While the Neural Network remains viable, its slight overfitting suggests room for improvement through regularization techniques. The kNN model, though showing the weakest comparative performance with its 0.850 validation accuracy, still demonstrates functional predictive capability - it simply underperforms relative to the superior generalization of Random Forest. This comparison underscores that while all models achieve at least reasonable validation performance, Random Forest provides the optimal balance of accuracy and reliability for deployment. Therefore we decide to use the Random Forest model to make predictions.

In the pred.py script, we first load the CSV file containing the input data and extract the relevant features used during training, utilizing standard libraries such as NumPy and Pandas. After reading the data, each instance is passed directly to the exported Random Forest model, which has been previously trained and saved. The model automatically generates predictions for each input based on its learned structure. These predictions are then gathered into a list and returned as the final output of the script.

IV. Our Prediction

The performance of our model was evaluated using the reserved test set, resulting in an accuracy of 85.45%. This evaluation provides a direct measure of the model's predictive capability on unseen data, reflecting its ability to generalize the patterns learned during training without overfitting to the training or validation sets.

Workload Distribution

1. Omar Khamis - I led the exploration of features and reviewed the models to implement the final prediction model.
2. Krit Kasikpan - I tested different models and tuned the hyperparameters for each to achieve the best results.
3. Inayah Dhaliwal - I performed EDA and contributed to report writing and final testing
4. Laurence Liu - I assisted in both data and model exploration