CSC311 ML Data Challenge Report

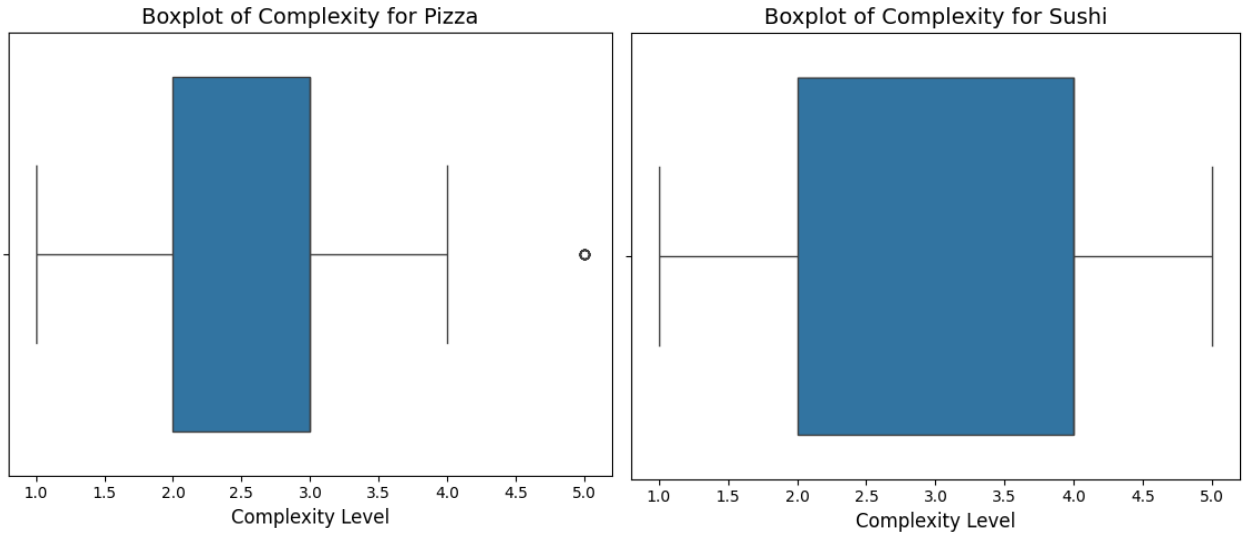
Zoya Siddiqui, Ariunzaya Bartsadgui, Alisha Hasan

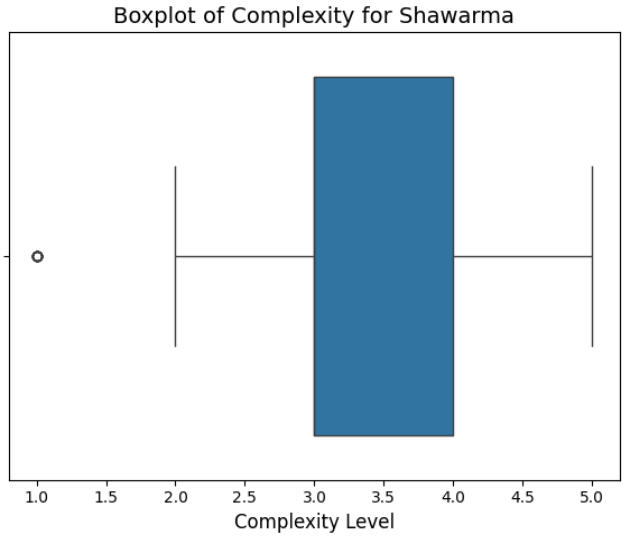
April 3, 2025

1 The Data

1.1 Data exploration

In this section, we explore the dataset to identify key highlights and outliers, which will aid in feature selection and the data splitting process. Upon analyzing the complexity of food preparation, we observed that the dataset contains minimal outliers, making it well-suited for feature selection. To better understand the characteristics of the data, we calculated summary statistics for the upcoming data split.





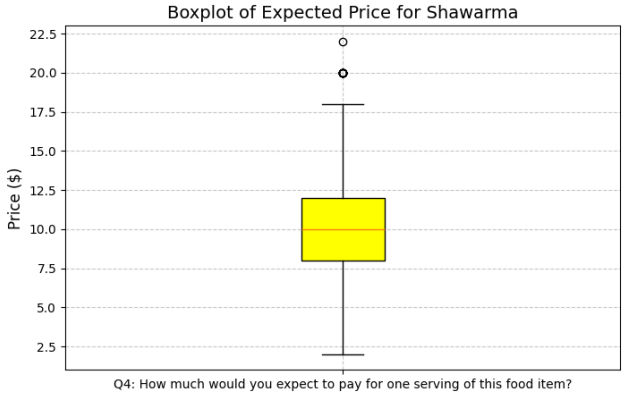
Regarding the number of ingredients required for each meal, many responses were open-ended, necessitating cleaning to extract and handle non-numeric entries. Respondents generally reported that dishes require

around 4-6 ingredients. As expected, these responses exhibited more outliers than those from the previous question, evidenced by the higher standard deviation relative to the mean. The results are shown in the table below:

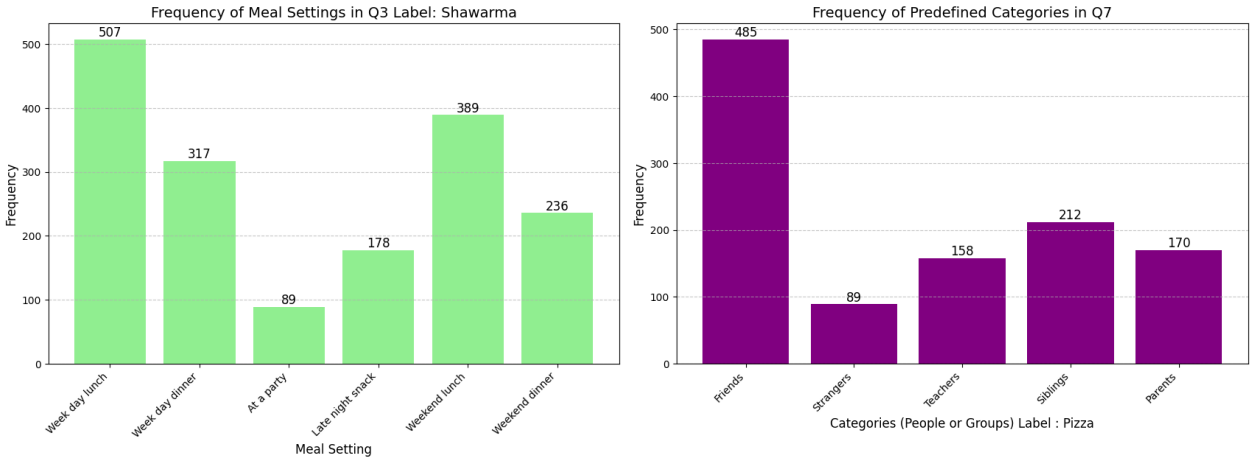
Food	Count	Mean	Std Dev	Min	Max
Pizza	519	5.48	2.38	1	15
Sushi	520	4.50	2.25	1	20
Shawarma	520	6.66	3.11	1	25

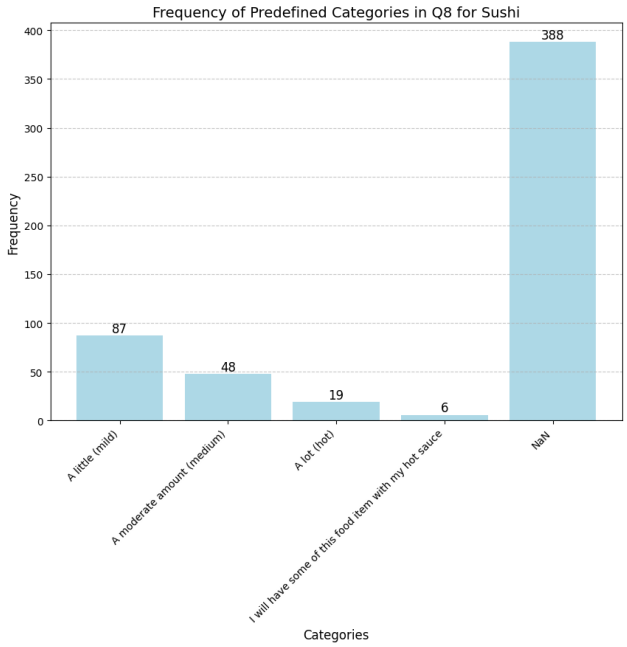
Table 1: Summary Statistics for Number of Ingredients

As for the food prices, the data reveals three distinct mean values along with several outliers resulting from open-ended responses, similar to the question regarding the expected number of ingredients. According to the survey, the average price was reported to be $5 for pizza, $16 for sushi, and $10 for shawarma.



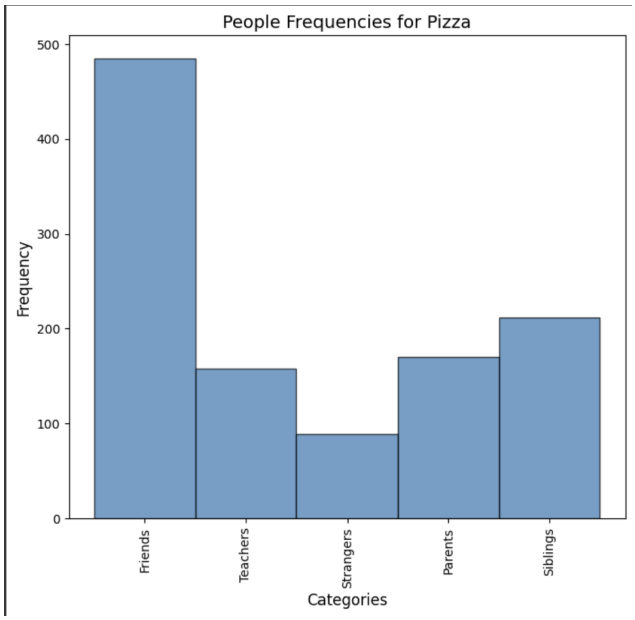
For categorical responses, we focused on the frequency of each response to help us identify missing data and determine the most common answers. The following graphs showcase examples from this analysis:

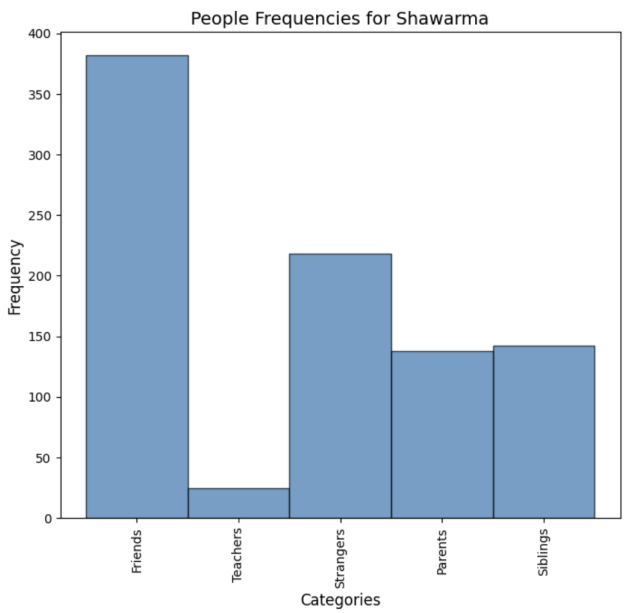


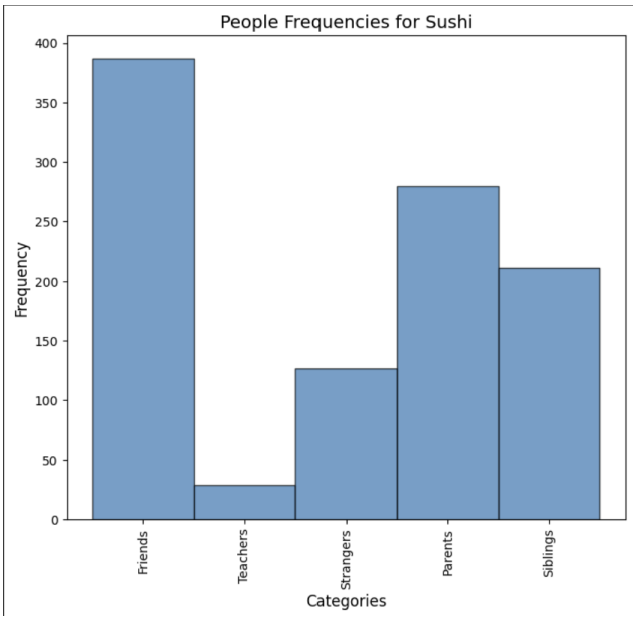


1.2 Description & Split

Upon initial review, it seemed reasonable to create features based on the questions asked. However, as we delved deeper into the data, we realized that the answers to some questions did not provide enough variability to differentiate between options effectively. For instance, when examining the frequencies of responses to the question "When you think about this food item, who does it remind you of?", there was no noticeable pattern within the responses that allowed us to differentiate between classes, as shown in the following graphs:







Comparing these three graphs with other potential features, it became clear that this feature would not provide sufficient distinction between the options.
For the features we ended up picking, we considered either:

1. The mean of the responses OR
2. The frequency of some response in the options

The following are the features we picked based off of the data:

1.2.1 Feature selection: Complexity of the Food

We calculated the average response for each food and found the following information.
For **Pizza**, the mean value was **2.876**. For **Shawarma**, it was **3.227**. For **Sushi**, it was **3.422**.
These values offer sufficient variability, making them ideal for feature selection.

1.2.2 Feature selection: Number of Ingredients

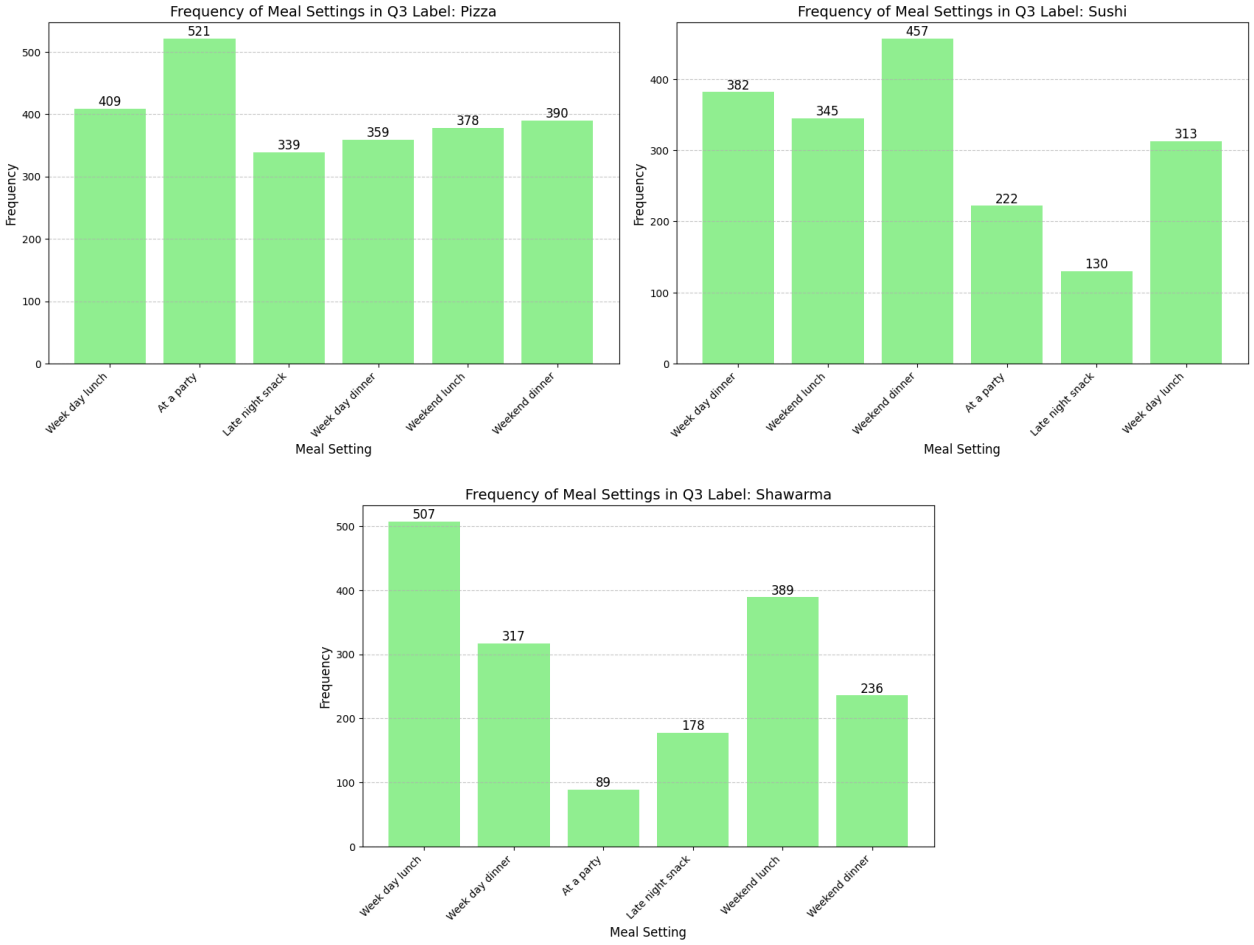
We calculated the average number of ingredients, based off of the responses, for each food. As shown in Table 1, the average number of ingredients was reported as 5.48 for pizza, 4.5 for sushi, and 6.66 for shawarma. Once again, these values are significantly different, making this a candidate feature.

Food	Count	Mean	Std Dev	Min	Max
Pizza	548	2.88	0.79	1.00	5.00
Sushi	548	3.42	1.21	1.00	5.00
Shawarma	548	3.23	0.94	1.00	5.00

Table 2: Summary Statistics for Food Complexity Ratings

1.2.3 Feature selection: Expected Setting for Serving

To decide whether the associated setting for each class was a useful feature, we started by checking the frequency of each setting for each food.



Since each food has a different setting that is chosen most often, this is a viable choice for a feature.

1.2.4 Feature selection: Expected Price

We calculated the average price (mean) value over all responses, for one serving of each food.

Food	Median	Mean	IQR	Min	Max
Pizza	5.00	8.49	8.00	1.00	30.00
Sushi	12.00	13.99	8.00	0.00	100.00
Shawarma	10.00	10.44	4.00	2.00	22.00

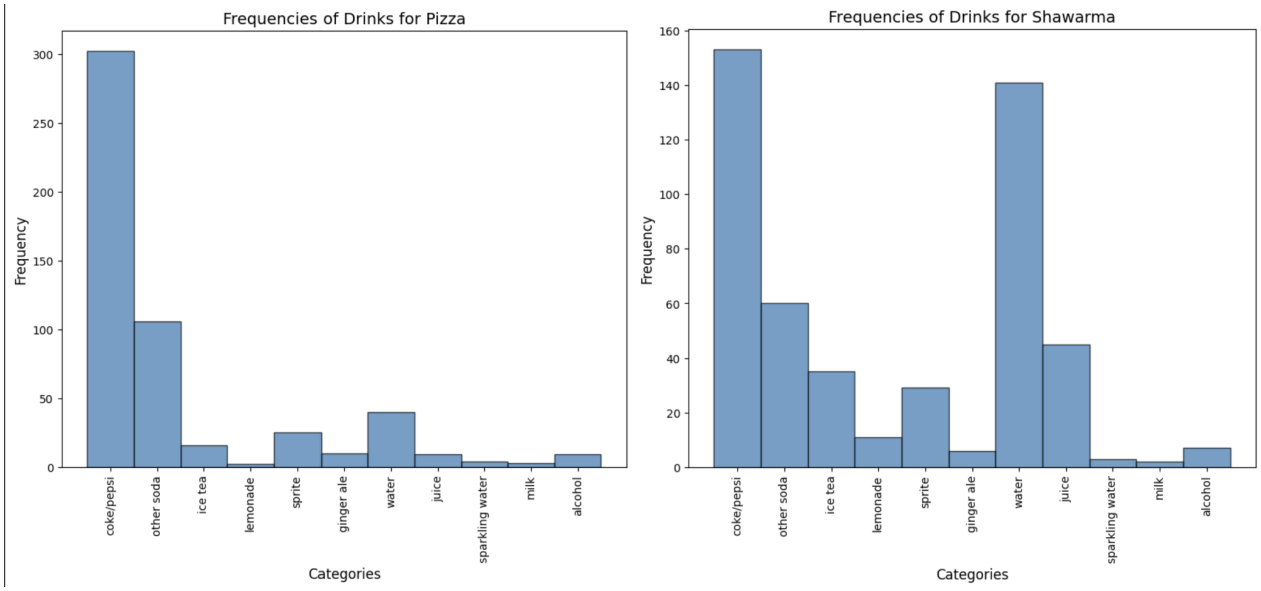
Table 3: Summary Statistics food price

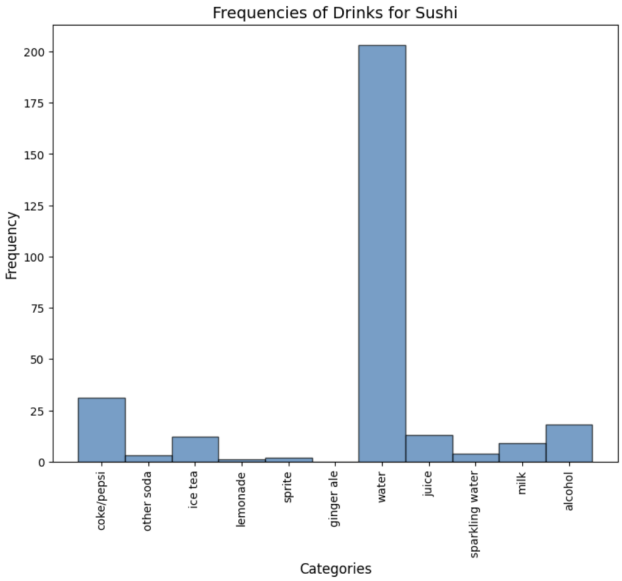
1.2.5 Feature selection: Preferred Drink

Since this question was open ended, we created our own categories and made it so that each response fell under one of these categories. Our categories were:

- Coke/Pepsi
- Other Soda
- Ice Tea
- Lemonade
- Sprite
- Ginger Ale
- Water
- Juice
- Sparkling Water
- Milk
- Alcohol

We found the following frequencies.

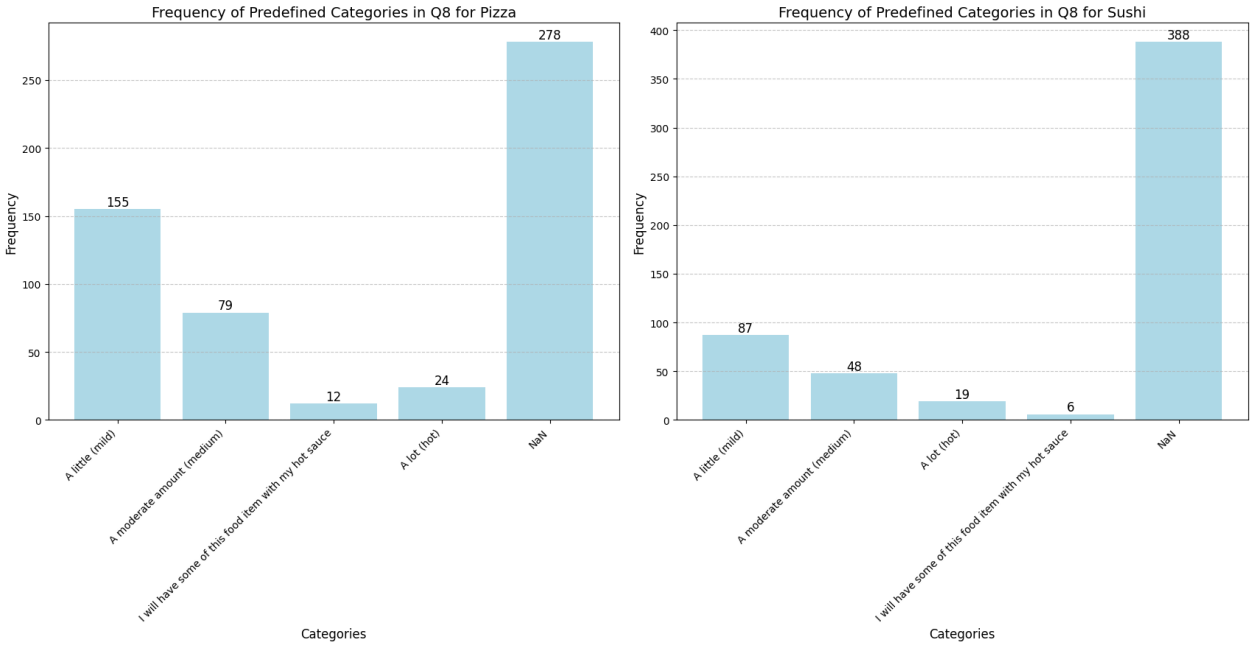


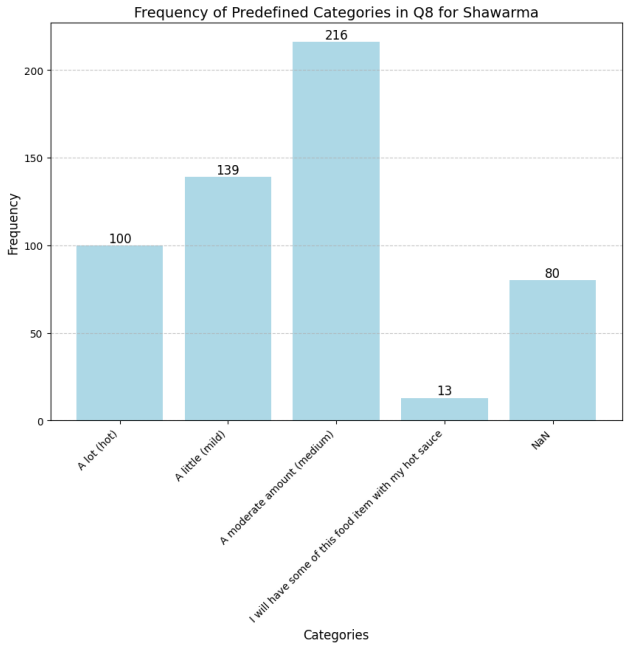


Although Pizza and Shawarma have the same highest-frequency, their second-most are very different. Altogether, the frequencies for each food item are generally very different, making this a good candidate for a feature.

1.2.6 Feature selection: Preferred Amount of Hot Sauce

Observing highly distinguished distributions of hot sauce amount in response from three classes, we concluded that Q8 will be a good feature for the models to evaluate.





1.3 Data Split

To evaluate the model's performance effectively, we split the dataset into training, validation, and testing subsets, using a 60-20-20 split . This strategy ensures a balanced approach, with the majority of the data dedicated to training the model while still preserving enough data for validation and testing.
The data was divided as follows:

- Training Set (60%): This portion of the data is used to train the model. It allows the model to learn the patterns, relationships, and trends within the dataset. The training set is the largest subset, ensuring that the model has enough information to generalize well to unseen data.
- Validation Set (20%): This subset is used during model development to fine-tune the model's parameters and select the best configuration. The validation set is not used for training but serves as a holdout set to help assess the model's performance during training and guide hyperparameter tuning.
- Testing Set (20%): The testing set is kept separate from the training and validation sets. It is used solely to evaluate the model's final performance after training and tuning. This ensures an unbiased

evaluation since the model has never seen this data during the training or validation process.

To maintain consistency and avoid bias, the split was done randomly, but in a stratified manner for categorical variables to preserve the distribution of each class across all subsets. For time-dependent features, such as transaction dates, a temporal split was applied to avoid any data leakage, where future data could influence the model's training.

The structured 60-20-20 split ensures that the model is trained on a sufficiently large dataset, validated effectively during training, and evaluated on a distinct set of data to assess its true performance.

2 Model

Throught ML Challenge, we have explored four different model to select the model that performs best. While our data split methods prevents data leakage by evenly splitting each class, choosing the model is mostly based on it's accuracy on the test set. In this section, we will mention all the models we have tried and their reason to be not selected.

2.1 Model 1: K nearest neighbor and Decision Tree (not selected)

This model was our initial idea, as when we explored the features it seemed like each feature could be easily split using one of these two models.

2.1.1 Model Description

This model is a combination of two models. The k-NN model considered

- Complexity
- Number of Ingredients
- Expected Price

The decision tree model considered

- Expected Setting
- Preferred Drink
- Preferred Amount of Hot Sauce

Our combined model took the prediction of the k-NN and the decision tree. We tried two different approaches to combining the models, and ran into separate issues with each of them. For both models, if the k-NN prediction and the decision tree prediction agreed, we simply returned the prediction. However, there were two different approaches if they disagreed.

1. *Making weighted choice based off of past predictions*
   This approach creates a dictionary and assigns weights for k-NN and the decision tree. These weights are calculated as follows:
   $$w_k = \frac{\text{kTestAccuracy}}{\text{dTestAccuracy} + \text{kTestAccuracy}}$$
   $$w_d = \frac{\text{dTestAccuracy}}{\text{dTestAccuracy} + \text{kTestAccuracy}}$$
   Then, it selects the label with the highest weight. This is essentially weighted voting.

   Problem
   Since the decision tree's test accuracy was higher than the k-NN model's test accuracy, this always chooses the decision tree's output, so this combined model is essentially only the decision tree model. It totally disregards the k-NN model.

2. *Making random choice*
   This approach uses a more probabilistic approach. It first calculates the probability of using the kNN model:
   $$prob_{kNN} = \frac{w_k}{w_k + w_d}$$
   Then, it generates a random value. If the random value is less than the probability of selecting k-NN, we use the k-NN prediction. Otherwise, we use the decision tree prediction.

   Problem
   Since the k-NN accuracy is lower than the decision tree accuracy, this model's combined accuracy is lower the decision tree accuracy. It performs poorly, especially considering that using the decision tree individually would result in better test accuracy.

2.1.2 Evaluation Results

For our first combined model approach (see **Model Description**), we got the following results:

k-NN Test Accuracy: 59%

Decision Tree Test Accuracy: 79%

Overall Test Accuracy: 79%

For our second combined model approach, we got the following results:

k-NN Test Accuracy: 59%

Decision Tree Test Accuracy: 79%

Overall Test Accuracy: 60%

2.1.3 Conclusion

This model performed alright, with the caveat that it seemed to take the decision tree's outcome over the k-NN model's outcome at a disproportionate rate. You can see this in the fact that the test accuracy is always very similar to the decision tree's test accuracy, only off by a few very small decimal points. No amount of playing around with the parameters seemed to solve this, as the decision tree simply performed better than the k-NN model, most likely due to the feature split. Ultimately, this model was outperformed significantly by logistic regression, making that the more viable choice.

2.2 Model 2: Neural network (not selected)

2.2.1 Model description

In applying the MLP model to solve the challenge, we used both categorical and numerical features together. The data obtained from previously implemented data split method. We selected the following choices were made for data preprocessing and model configuration:

Adjustments Made for the Task

- **Preprocessing:**
  - One hot encoding is applied to categorical features to convert them into a numerical format that can be processed by the neural network. This method is preventing from normalization issues the NN model might face as its sensitive to normalization. Missing values were handled in the data split process.
- **Feature Scaling:**
  - Numerical features are standardized to have zero mean and unit variance, ensuring that all features are on a comparable scale using min-max normalization. This ensure more compatible performance with ReLU activation.

- Dropout is applied during training with a rate of **0.3** to prevent overfitting and improve generalization.
- The model is evaluated using the **accuracy** metric, which provides an interpretable measure of the classification performance.
- Early stopping with a patience of **5 epochs** is employed to prevent overfitting, halting training if validation performance does not improve for a set number of epochs.

Mathematically, here is how our model will produce a prediction:

$$\mathbf{m} = XW_1^T + b_1 \tag{1}$$

$$\mathbf{h} = \text{ReLU}(\mathbf{m}) \tag{2}$$

$$\mathbf{z} = \mathbf{h}W_2^T + b_2 \tag{3}$$

$$\mathbf{y_i} = \text{Softmax}(\mathbf{z_i}) \tag{4}$$

$$\mathbf{y_i} \in [0,1], \quad \sum_{j=1}^{C} y_{i,j} = 1 \tag{5}$$

2.2.2 Evaluation Results

The model is evaluated using the accuracy metric, which is the percentage of correct predictions. The accuracy is calculated by comparing the predicted class labels with the true labels. The predicted labels are obtained by applying the softmax function to the output layer of the model. The prediction function uses the trained model to compute the output probabilities for new input data. The class with the highest probability is selected as the predicted class.
The model was trained on a dataset and evaluated using test data. The test accuracy was acceptable, calculated and reported as:

Test Accuracy: 76.88%

Validation Accuracy: 52.29%

2.2.3 Conclusion

The MLP model was partially successful, showing low validation accuracy and high test accuracy. When the test accuracy is not too high, large difference between validation and test set accuracy shows data leakage even after perfectly split data.

During training, we experimented with Xavier initialization, tanh and sigmoid activations, as well as learning rate tuning until convergence. However, the results remained consistent despite these efforts to prevent overfitting. Thus, we concluded that the model was too complex for the challenge and the features we selected. Consequently, the logistic regression model introduced in the following section proved to be a more optimal choice.

2.3 Model 3: Logistic regression (selected)

2.3.1 Model Description

The quantitative features, which were complexity, number of ingredients, and expected price, provided objective dimensions for classification of each of the three food items.
The categorical variables, which were expected setting, preferred drink, and preferred amount of hot sauce, were transformed through one-hot encoding to make them suitable for the logistic regression algorithm.
In addition to training the model on these features, data preprocessing was a crucial step in the process of preparing the features. We standardized all numerical features using their respective means and standard deviations. For categorical variables, the one-hot encoding process created binary features for each possible category value. We implemented handling of missing values by assigning standard defaults based on the data distribution, and this preprocessing pipeline was utilized consistently throughout the training process. Regularization was applied to prevent overfitting by penalizing large coefficient values. The regularization strength (λ) was tuned using cross-validation to balance bias and variance.

The logistic regression model makes predictions using the following equations:

$$z = XW + b$$

$$P(y_i = 1|X) = \frac{1}{1 + e^{-z}}$$

$$\mathbf{y} = \text{argmax} P(y_i|X)$$

2.3.2 Model Evaluation

The model was evaluated using accuracy as the primary metric. The performance on the test set was as follows:

Test Accuracy: 81%

Validation Accuracy: 79%

Out of all the models, logistic regression performed the best.

3 Model Choice and Hyperparameters

3.1 Evaluation Metrics

Before beginning with designing any models, we did a thorough exploration of the data, which included taking a look at the distributions of responses and cleaning up the data to be split into training, validation, and test sets. In doing this, we setup much of the pre-processing to be individual of the models. That meant that when we were coding our models, we only had to slightly tweak the pre-processed features to work well with our individual models, so the training, validation, and test data we were working with was very similar. Below is the code we used to generate our sets:

```
# SPLITTING DATA
df = pd.DataFrame(cleaned_data)

# Set random seed for reproducibility
np.random.seed(42)

# Get indices for each class
pizza_indices = df[df['Label'] == 'Pizza'].index.tolist()
shawarma_indices = df[df['Label'] == 'Shawarma'].index.tolist()
sushi_indices = df[df['Label'] == 'Sushi'].index.tolist()

def split_indices(indices, train_ratio=0.6, val_ratio=0.2, test_ratio=0.2):
    """Split indices into train, validation, and test sets while preserving order."""
    np.random.shuffle(indices)
    n = len(indices)
    train_end = int(train_ratio * n)
    val_end = train_end + int(val_ratio * n)

    train_idx = indices[:train_end]
    val_idx = indices[train_end:val_end]
    test_idx = indices[val_end:]

    return train_idx, val_idx, test_idx

# Split each class indices
pizza_train, pizza_val, pizza_test = split_indices(pizza_indices)
shawarma_train, shawarma_val, shawarma_test = split_indices(shawarma_indices)
sushi_train, sushi_val, sushi_test = split_indices(sushi_indices)

# Combine indices
```

```
train_indices = pizza_train + shawarma_train + sushi_train
val_indices = pizza_val + shawarma_val + sushi_val
test_indices = pizza_test + shawarma_test + sushi_test

# Create DataFrames - THIS IS WHAT WE'LL USE
train_df = df.loc[train_indices].reset_index(drop=True)
val_df = df.loc[val_indices].reset_index(drop=True)
test_df = df.loc[test_indices].reset_index(drop=True)
```

Specifically, take a look at **line 4** of this code: we set a random seed that helps us to generate this data. Each of us played around with different values of this random seed and found a general trend between this number and our models.
Our **main evaluation metric** was **test accuracy**.

3.2 Hyperparameters

The only hyperparameter tuned in our model was a regularization parameter C. The regularization parameter C controls the trade-off between fitting the training data whilst also stopping the model from being overcomplex, where smaller values would impose stronger regularization. We selected $C = 1.0$ after analyzing the bias-variance tradeoff in the model. This choice of metric is justified by bias-variance analysis giving a good representation of how simple or complex a model is, which can further be projected to infer how well the model can generalize. The results were as follows, computed using cross-entropy loss:

C	bias_indicator	variance_indicator	total_error
0.1	0.580987	0.054469	0.635456
1.0	0.389216	0.147440	0.536656
10.0	0.275607	0.273296	0.548903

Table 4: Summary Statistics for Bias-Variance Analysis

As shown, the sum of the bias and variance indicators were minimized for a choice of $C = 1.0$, which rendered it the optimal choice.

3.3 The Final Model

The final model chosen was the multinomial logistic regression model with six features - three of which are numerical, and three of which are categorical. The numerical features were standardized using mean and standard deviation from the given training data, while the categorical features were one-hot encoded.

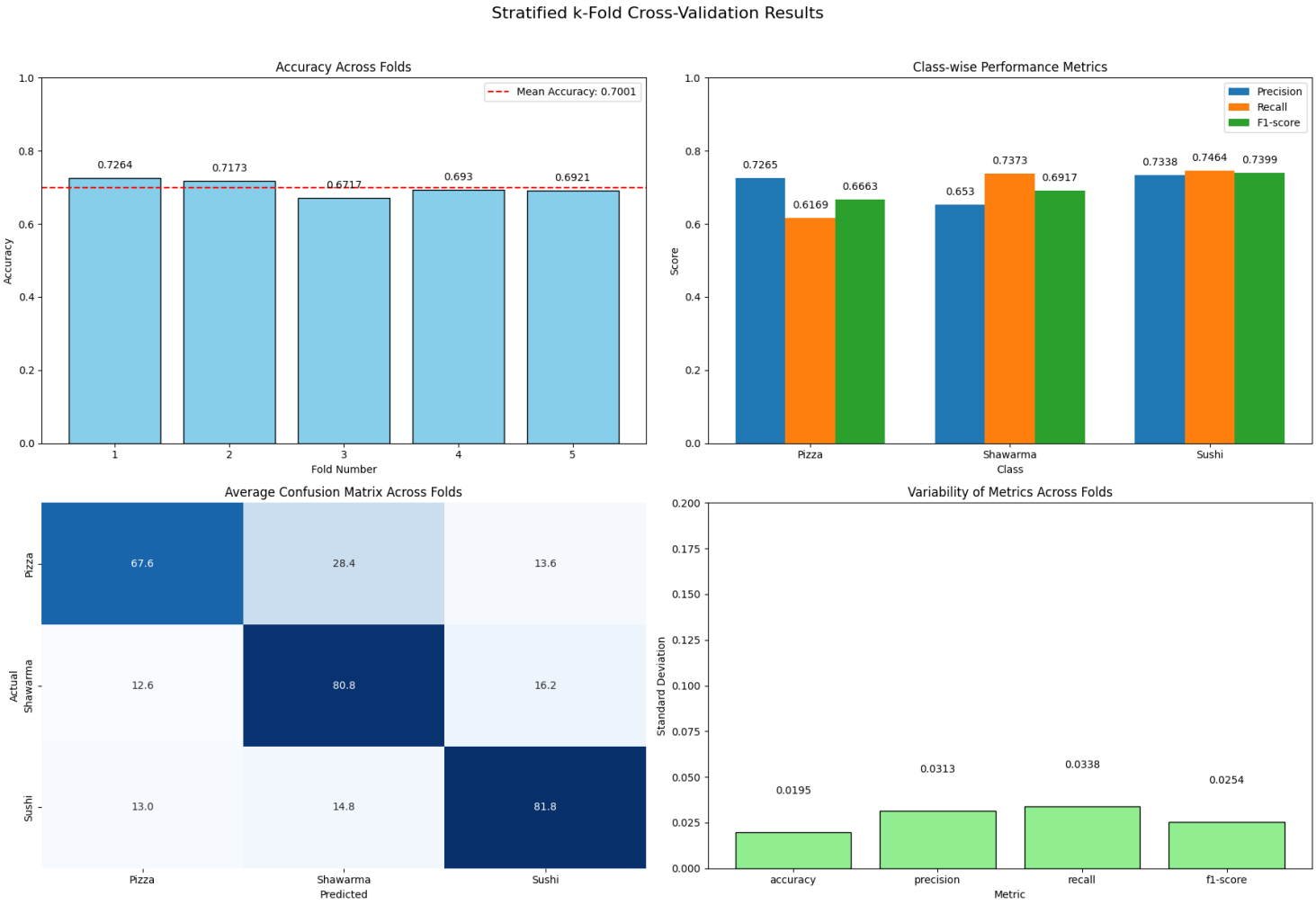
Our model begins by cleaning the data, during which missing data values are replaced by the mean of the set for numerical features or the mode of the set for categorical features, and also during which similar categorical features are mapped together as one (for example, different types of tea are combined, since they are associated with the same food anyhow). After the data is cleaned, the features are unified into one vector.

For each possible class of food, the model calculates linear logits by multiplying each feature by its learned coefficient and adding an intercept term, as per the logistic regression model used to generate the coefficients. These logits are then fed through softmax activation to get probabilities, and the class with the highest probability is selected as the prediction.

4 Prediction

For the prediction analysis part, we used a stratified k-fold cross-validation. Across 5 folds, the average test accuracy was 0.7001, so we are expecting $\tilde{7}0\%$ accuracy on the final unseen dataset. We further investigated false positives and true negatives as over-reliance on a single accuracy method could cause unforeseen alignment issues. The choice stratification ensures that each fold maintains the same class distribution as the original dataset. We have already seen roughly even distribution of classes in the data exploration stage.

Thus, there is no inherent risk of data leakage and data disparity from our testing analysis part. This, way, we are preventing the under representation of a single class. After performing 5 fold stratified cross validation on our logistic regression model, we obtained the following result:



Evaluating the results above, We see that the f1 score is around 0.7, indicating the model is acceptable but not too good model for the unforeseen dataset. The lower variability between classes (0.0338) validated the accuracy parity (equalized odds). Finally, from the confusion matrix, we see that the model might bias the Pizza dataset over the Sushi, but the variability is low enough to neglect on final result.

5 Workload Distribution

5.1 Zoya Siddiqui

Zoya wrote up the feature description, as well as the script that split and cleaned all the data to get it ready to be used in the models. She implemented model 1.1 and 1.2, using k-Nearest-Neighbours and Decision Trees. Finally, she worked on writing the report.

5.2 Ariunzaya Bartsadgui

Ariunzaya was responsible for the data exploration and prediction analysis parts of the challenge as well as implemented the Neural network approach which hasn't been selected for the final model. Ariunzaya also took part in the report-writing process, documenting her contribution.

5.3 Alisha Hasan

Alisha was responsible for feature selection and programming a Logistic Regression model. She also implemented `pred.py`, and wrote up the portion of the report regarding hyperparameter tuning and the final model.