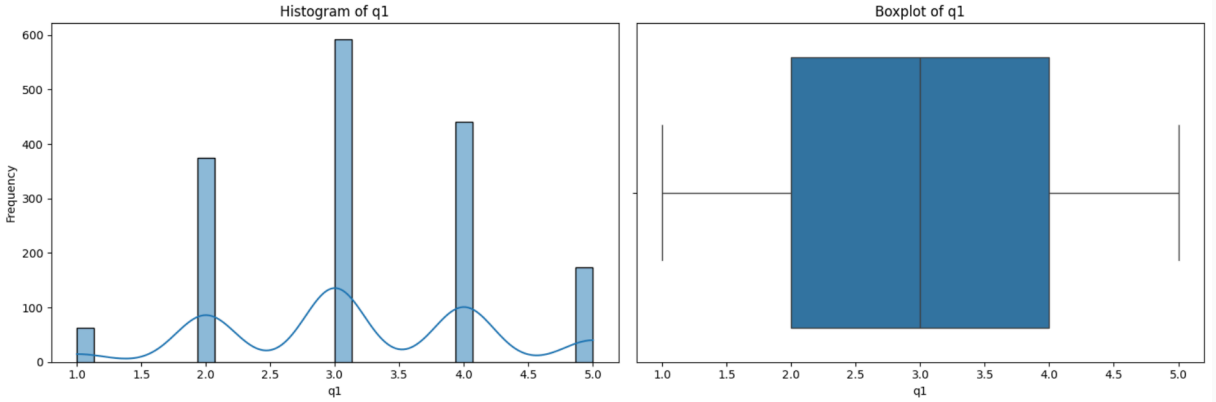
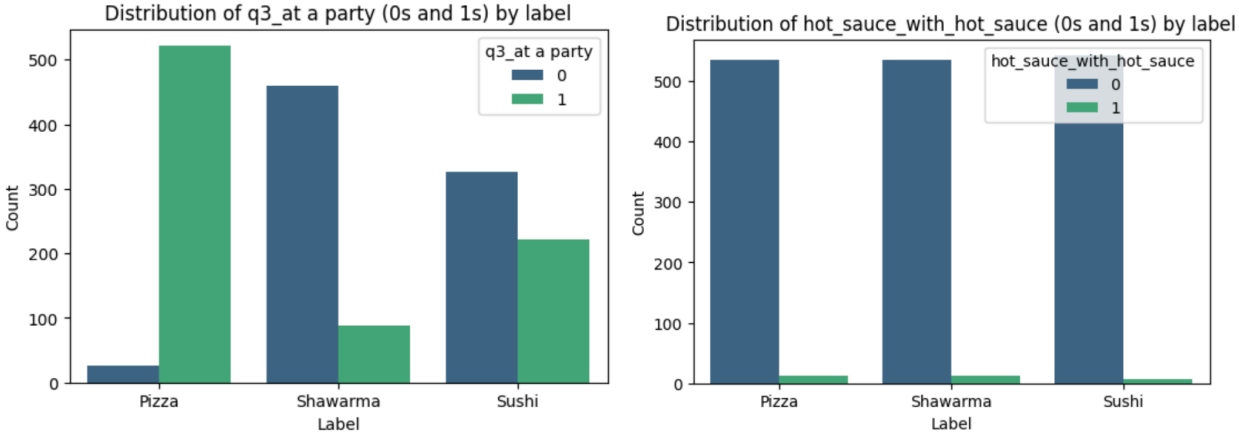
# ML Challenge Report

## Data

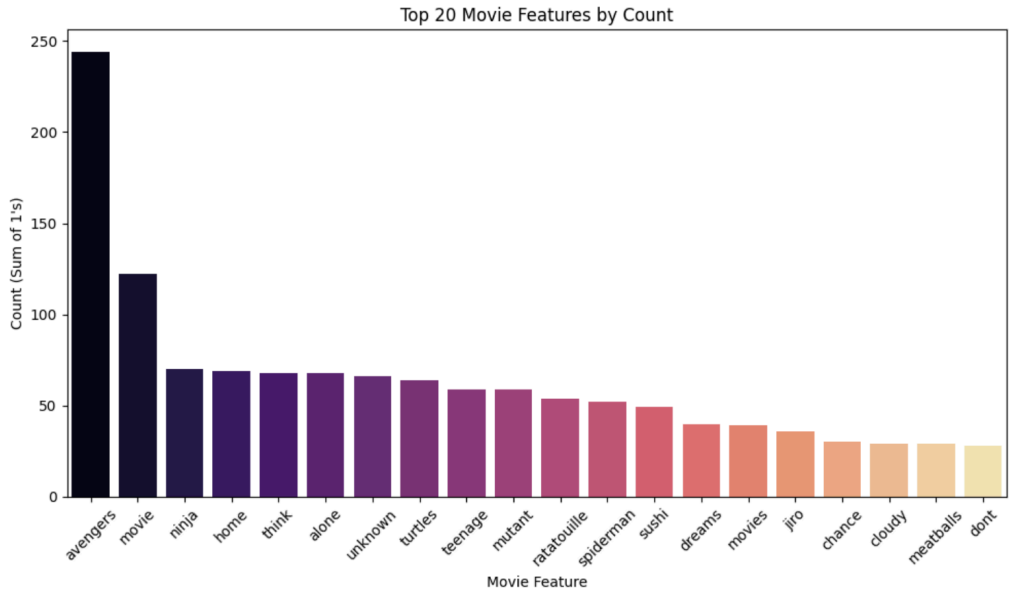
### Data Exploration:

We noticed that there were both numerical and categorical features in our data. For numerical features such as Q1 and Q2, we have decided to use histograms to visualize their spread and boxplots to show outliers. For categorical features such as Q3 and Q6, we used bar plots to show the distribution grouped by labels.

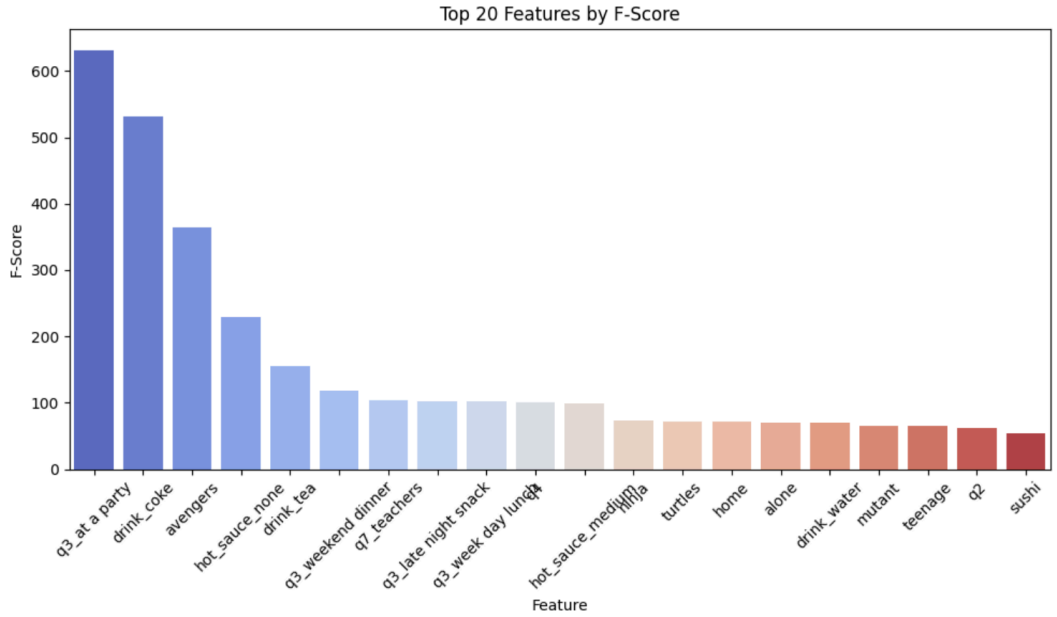


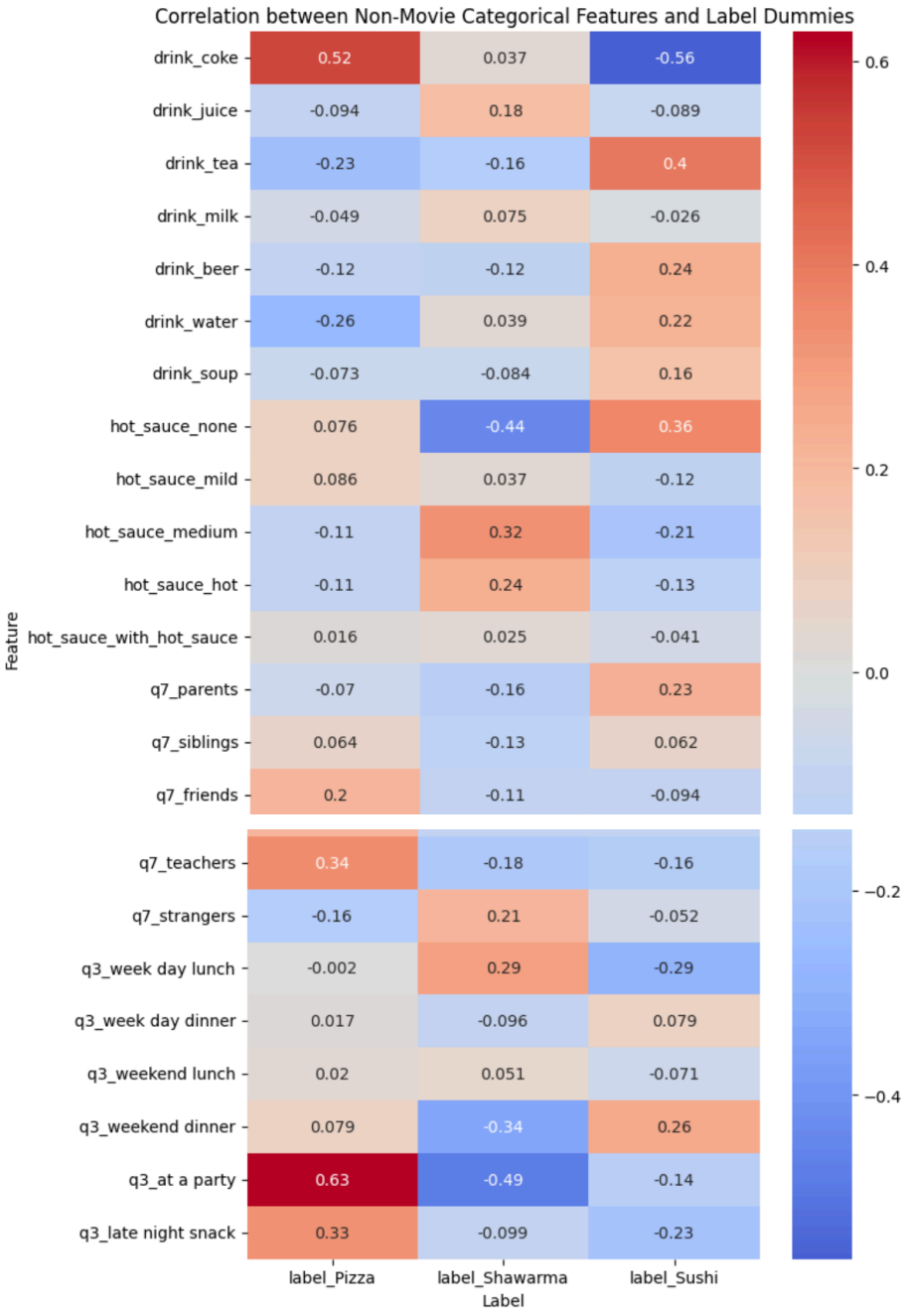

The bar plots helped us get an idea of the correlation and usefulness of features. The bar plot of distribution of eating at a party indicates possible correlation between pizza and eating at a party since the portion of responses selecting eating at a party for pizza is significantly higher than those selecting shawarma or sushi. On the other hand, the bar plot of distribution of having “this” food item with hot sauce indicates that this feature possibility offers very little to our prediction since most responses did not select this option and the portion that did is split mostly evenly between the three types of food.



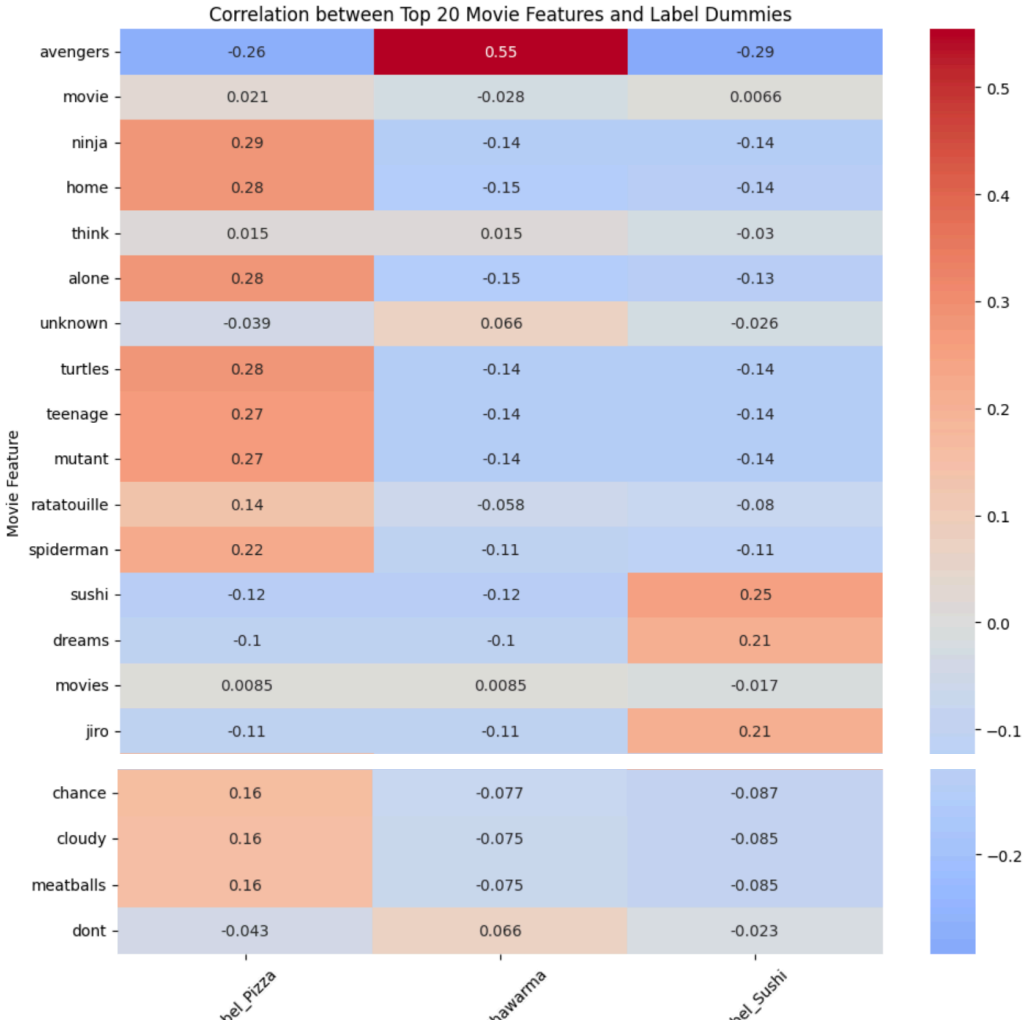

We also created a bar plot that shows the top 20 movies by count that we derived using bag of words. Some noticeable movies are avengers, teenage mutant ninja turtles, home alone, ratatouille, etc. We would be paying attention to these titles that would potentially show high correlation with certain food items.




We also computed the F-score and plotted the highest 20 features in a bar plot. This also gives us a glance at potentially important features.




We also created a heatmap that provides visual representation of how each feature correlates with each label. This also verified our previous observations such as high correlation between eating at a party and pizza.




We created the same heatmap for top 20 movies. It also verified our previous observations on certain movies such as avengers or home alone having strong correlation with shawarma and pizza respectively.

### Feature Selection:

Initially, we experimented with removing certain features based on statistical criteria—such as eliminating outliers using average thresholds, or dropping features with low F-scores or small coefficients from linear models. While this approach seemed promising in theory, it didn't translate well in practice, particularly when using Random Forests. Because decision tree–based models are inherently robust to irrelevant or less informative features and are not significantly affected by high-dimensional input, we found that reducing the number of features actually hurt performance. This was reflected in our results, where test accuracy consistently dropped by 1–3% whenever features were removed. As a result, our final solution kept all original features, which led to better overall performance.

### Data Cleaning/Representation:

First we standardized column names by converting them to lowercase, replacing spaces with underscores for readability, and replacing any missing values with “unknown”. We categorized Q1, Q2, and Q4 as numerical features. Q1 was already numerical, so we left it unchanged. For Q2, we extracted numeric values from text responses. If multiple numbers were present, we averaged those between 1 and 10. If no numbers but commas existed, we counted the commas assuming they represented a list of ingredients. If the response contained neither numbers nor commas, we estimated the count by dividing the character count by 5. We made sure to only have integers by applying the ceiling function. For Q4, we extracted the first numeric value from the response. If no numbers were found, we would set it to 0. For Q5, we tokenized responses and converted them into a bag-of-words hot encoding. For Q6, we hot encoded drink names and matched them against a predefined set of drink types. For Q8, we used a similar approach by extracting the level of hot sauce from the responses and one-hot encoding them into features. Q3 and Q7 allowed multiple categorical options to be selected, so we applied one-hot encoding again to represent each choice as a separate binary feature.

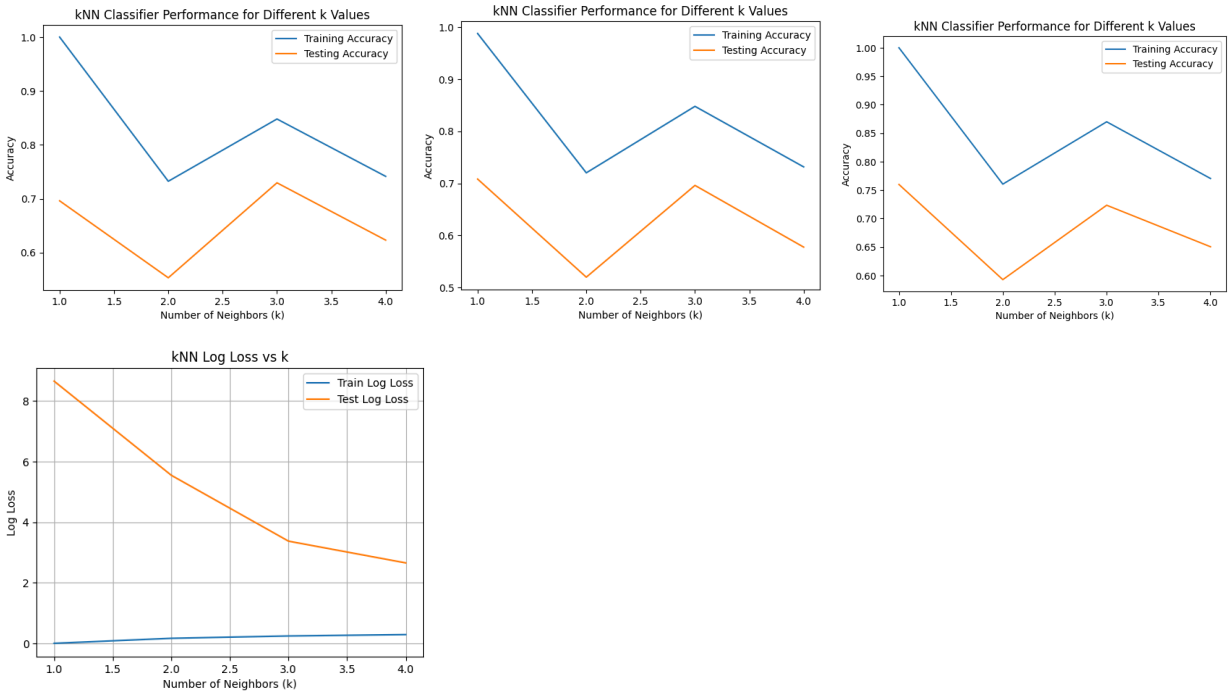
### Data Splitting:

We used a standard 70/15/15 data split, where 70% of the data was allocated for training, 15% for validation, and the remaining 15% for testing. The training set is used to teach the model by allowing it to learn patterns and adjust its internal parameters. The validation set helps us tune hyperparameters and monitor the model’s performance during development, ensuring it generalizes well and doesn’t overfit. Finally, the test set is reserved for evaluating the model’s performance on completely unseen data, providing an unbiased estimate of how well the model would perform in real-world scenarios. This approach offers a balanced way to train, fine-tune, and assess the model effectively.

## Models Explored: KNN, Neural Networks, Random Forest

### KNN:

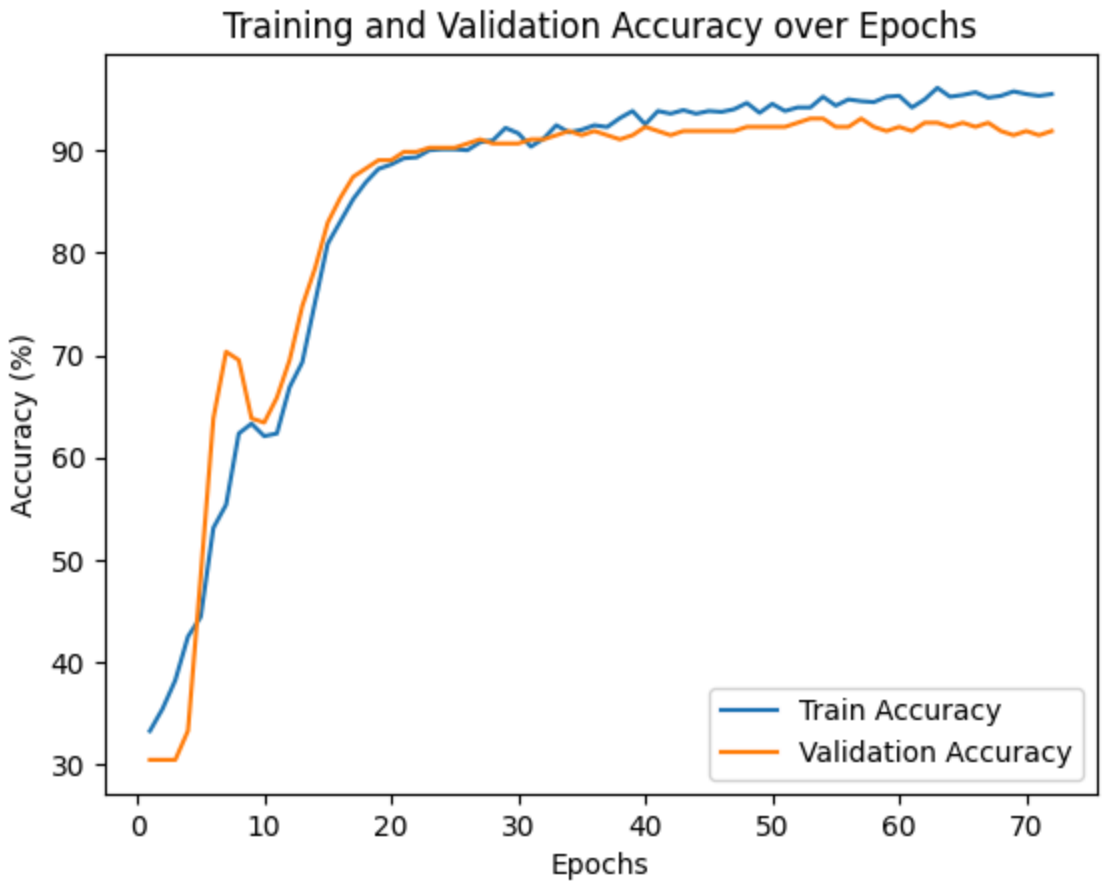
We chose to include K-Nearest Neighbors as a baseline model to benchmark the performance of our other models. KNN is well-suited for this purpose because of its simplicity and fast training time, there’s no need to retrain when making predictions, and results are directly computed based on the distance between data points. While KNN tends to struggle with large datasets due to slow inference and high memory usage, this wasn’t an issue for us given the small dataset size of 1,644 data points.




The three graphs below compare different feature combinations. The left graph shows performance when trained on just the basic data, excluding both the drinks and movie features. The middle graph uses the same base features but adds drink-related data excluding movies, and the rightmost graph includes both the drinks and a bag-of-words encoding of movie titles. Initially, the results seemed a bit odd, the middle and right graphs differ quite a bit even though the only change was adding movies. However, it became clear that the movie encodings had a significant impact, consistently boosting test accuracy. This suggests that the inclusion of movie-related features contributes meaningful information to the model's predictions.

## Neural Networks:

For the neural network, we chose this model because it can capture complex, non-linear relationships between features, making it well-suited for this classification task. Neural networks excel at learning intricate patterns across multiple variables, such as food complexity, pricing, meal timing, and social context. The main downside of using a neural network is that it can be prone to overfitting if not properly regularized. Additionally, neural networks are less interpretable than models like decision trees, making it harder to explain individual predictions. One challenge with using a neural network for this task is the potential lack of data, as 1,644 data points may not be sufficient for the model to generalize well. Neural networks typically require large datasets to learn complex patterns effectively, and with limited data, the model risks overfitting to the training set. Additionally, neural networks involve extensive hyperparameter tuning, such as adjusting the number of layers, neurons per layer, learning rate, and regularization techniques.




The graph shows training and validation accuracy over 80 epochs. We observe that training accuracy steadily increases and plateaus around 95%, while validation accuracy peaks slightly lower, around 90%, before leveling off.

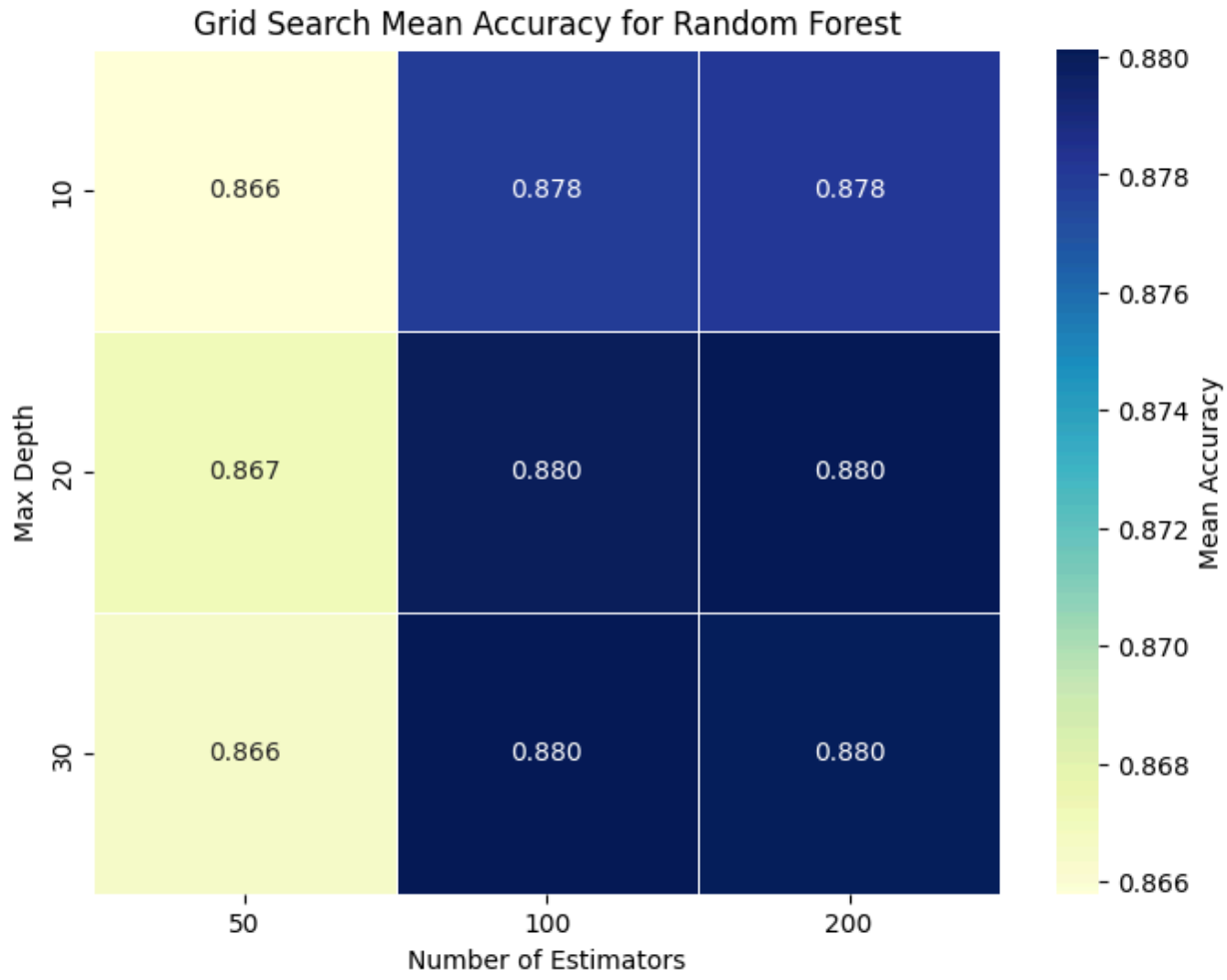
This gap between training and validation accuracy suggests mild overfitting, which is expected given our small dataset (1,644 examples). Neural networks are powerful, but with limited data, they tend to memorize training patterns rather than generalize perfectly. The model learns quickly within the first ~15 epochs, which is typical, most useful features are captured early, and later epochs mainly refine (or overfit) the model.

Despite this, validation accuracy remains high and stable, indicating the model is learning useful patterns, not just noise. Hyperparameter tuning (like dropout or early stopping) could help close the gap, but even as-is, the performance shows that the neural network successfully captures complex relationships in the data.

**Random forest:**

Random forest is a strong choice for this classification task because it effectively handles both numerical and categorical data, making it well-suited for features like food complexity, pricing, meal timing, and social context. Its ability to capture non-linear relationships and feature interactions without requiring extensive preprocessing is a major advantage. Additionally, since

random forests are an ensemble method, they reduce the risk of overfitting compared to single decision trees, which is particularly useful given the limited dataset size. The built-in feature importance scores also make it easier to interpret which factors contribute the most to predictions.




The heatmap shows the mean validation accuracy for different combinations of max depth and number of estimators in the random forest model. Accuracy stays fairly consistent across the grid, ranging from 0.866 to 0.880, with the highest values occurring when using 100 or 200 estimators and a max depth of 10. This indicates that while tuning helps slightly, the model performs reliably well across most settings. The ensemble structure of random forests helps reduce overfitting even with a small dataset, and the results suggest that increasing tree count provides more benefit than increasing depth. However, even at its best, the random forest model reached a peak test accuracy of around 91% during this grid search. Later, when we ran a broader grid search with more aggressive tuning and the full feature set, the test accuracy reached around 92%, suggesting that the initial heatmap does not fully capture the model's potential. This also shows that random forests can still benefit from careful tuning, especially when using all available features.

# Model Choice and Hyperparameters

## Model Selection:

The final model for pred.py was chosen based on a comparative analysis of different machine learning models, including Decision Trees, Random Forests, k-Nearest Neighbors (KNN), and Neural Networks. The selection process involved training and evaluating these models using the same dataset splits to ensure comparability of results.

- **k-Nearest Neighbors (KNN)**: A KNN model was tested with various values of k. While it performed decently on training data, it struggled with high-dimensional feature spaces, leading to lower accuracy and higher log loss of higher than 2.2.

- **Neural Networks**: We built a neural network with an input layer, two hidden layers (ReLU activation), and a Softmax output layer. We then tuned the architecture by adjusting depth, width, activation functions, and dropout. Optimization involved tuning the learning rate, comparing Adam and SGD optimizers, and experimenting with batch sizes. Regularization techniques such as L2 weight decay, early stopping, and dropout adjustments were applied. Finally, we tuned the hyperparameters using Optuna. Despite these efforts, it did not outperform ensemble-based models, achieving a best test accuracy of around 86% with a log loss of 0.68.

- **Random Forests**: A RandomForestClassifier was trained and optimized using GridSearchCV. It showed superior performance in comparison to KNN and Neural networks by achieving a test set accuracy of 92% and log loss of 0.35. This robustness and reduced overfitting made it the final choice for our prediction script.

## Evaluation Metrics:

To ensure a fair comparison, the following evaluation metrics were used:

- **Log Loss**: This metric measures the uncertainty of predictions by evaluating the predicted probability distribution. Since the dataset involves classification, log loss was crucial in determining how confident the model was in its predictions, helping to identify whether the model assigned appropriate probabilities to each class rather than just making correct or incorrect predictions.

- **Accuracy on Test Set**: The proportion of correctly classified instances. While accuracy provides an overall performance measure, it was complemented by log loss to capture finer details of the model's predictive confidence.

Both metrics were calculated on validation and test sets to ensure that the models were not overfitting and could generalize well to unseen data.

### Hyperparameter Tuning:

The following hyperparameters were tuned for different models:

**Decision Trees:**

- max_depth: Tested values [1, 5, 10, 15, 20, 25, 30, 50, 100]
- min_samples_split: Tested values [2, 4, 8, 16, 32, 64, 128, 256, 512, 1024]
- criterion: ['gini', 'entropy']

The best-performing decision tree had max_depth=8 and criterion="entropy", but it was still prone to overfitting compared to ensemble models.

**Random Forests (Final Model):**
Hyperparameters were optimized using GridSearchCV with the following values:

- n_estimators: [50, 100, 200]
- max_depth: [None, 10, 20, 30]
- min_samples_split: [2, 5, 10]
- min_samples_leaf: [1, 2, 4]
- criterion: ['gini', 'entropy', 'log_loss']

After tuning, the best-performing model had the following hyperparameters:

- n_estimators=200
- max_depth=30
- min_samples_split=2
- min_samples_leaf=1
- criterion='entropy'

This configuration resulted in the highest validation accuracy and the lowest log loss among all tested models.

### Final Model Implementation in `pred.py`

The final model implemented in pred.py is a RandomForestClassifier initialized with the best hyperparameters identified through GridSearchCV. The model was trained on the training dataset and evaluated using log loss and accuracy on both validation and test sets. We put every tree in the forest into one CSV file and used that in our prediction script.

## Prediction

## Performance Estimate:

The final Random Forest model achieved a test set accuracy of 92% and a log loss of 0.35 so we estimate our performance on the new test set should be around the same.

## Reasoning:

The performance estimate is derived from the results obtained during model evaluation on the test set. The RandomForestClassifier was trained using optimal hyperparameters selected via GridSearchCV, ensuring the best balance between bias and variance. Empirical results demonstrate that the model generalizes well, as indicated by the minimal discrepancy between validation and test set performance where the val accuracy was around 86-87%.

## Workload Distribution:

Andy: I did Neural Networks, Random Forest, Prediction Script
Frank: I did the q5, q3, q2 data processing, tested different datasets with KNN, and helped Andy with running simulations of Random Forest.
Michael: I did the code and report portion of data exploration and representation, generated and tested Random Forest and Decision Trees.

Some paragraphs in this report were refined using ChatGPT to improve clarity and conciseness; all final content was critically reviewed and authored by our team.