CSC311 Project Report

Kaifeng Li, Kangzhi Gao, Keke Chen, Xiaoyi Jia

April 9, 2024

1 Data Exploration

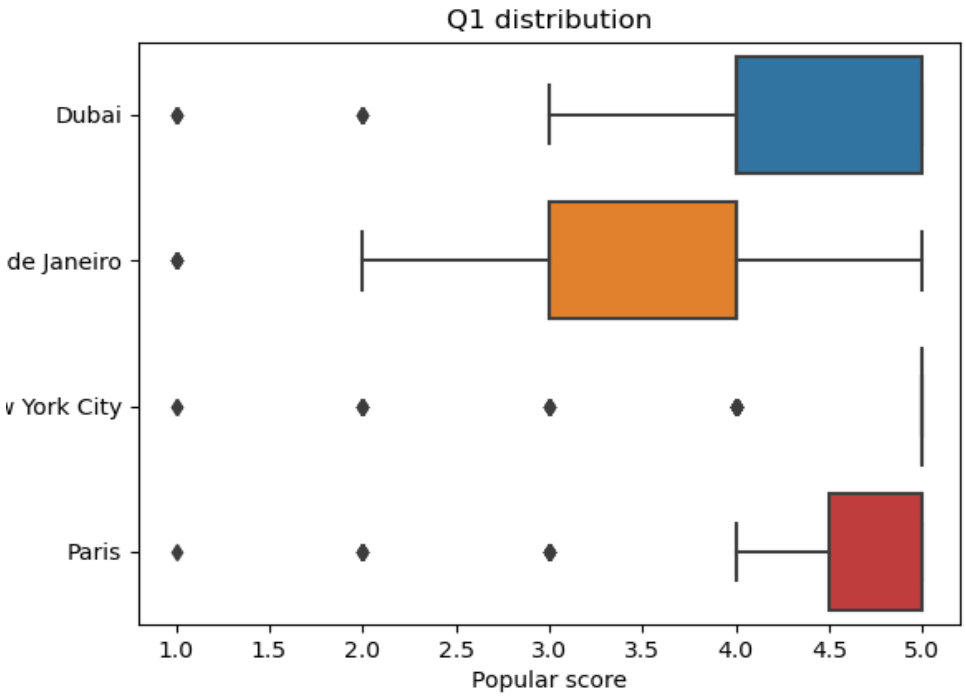
Before we perform any pre-process strategies, we will first take a look at the distribution of each given feature in the dataset. Also, real-world data is very likely to have outliers and missing values. Because our sample size is small, we decided to handle all the missing values by filling them with reasonable values.

	Missing Value count	Outlier count
Popular Score	0	0
Efficient Score	0	0
Uniqueness Score	0	0
Enthusiasm Score	0	0
Traveler Choice	7	0
Key word Rank	0	0
Avg Temp	7	3
Language count	7	29
Style Count	7	153
Quote	37	0

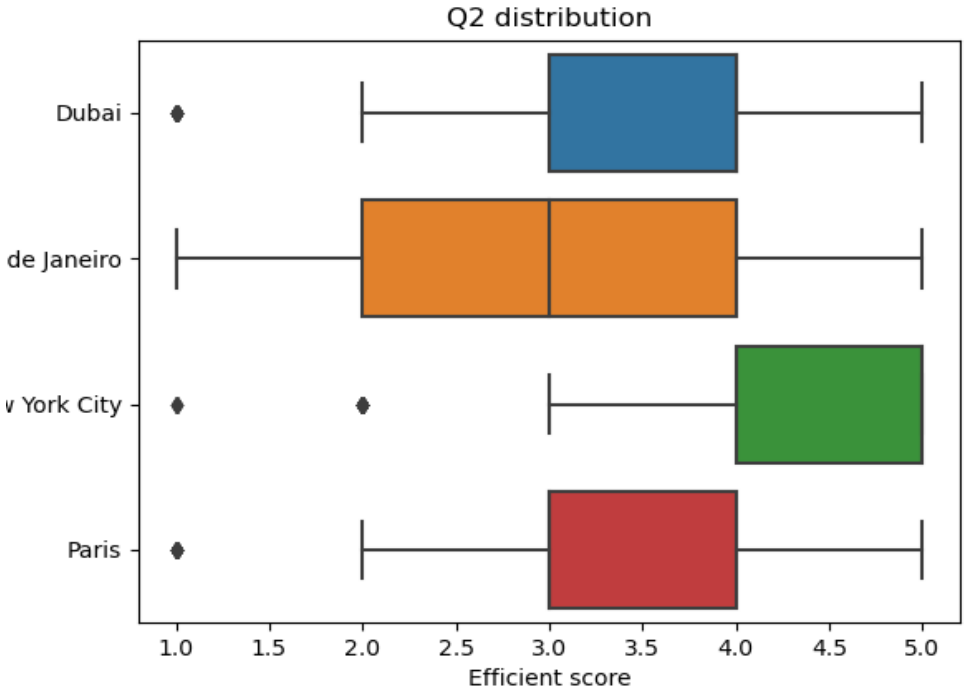
(Figure1) This figure illustrates the total number of missing values and outliers among all questions.

This table shows the amount of missing values and outliers in each feature, and most of the outliers are distributed in questions 7, 8 and 9. So, we will go through all the features, and handle the outliers before we move on the questions 7, 8 and 9.

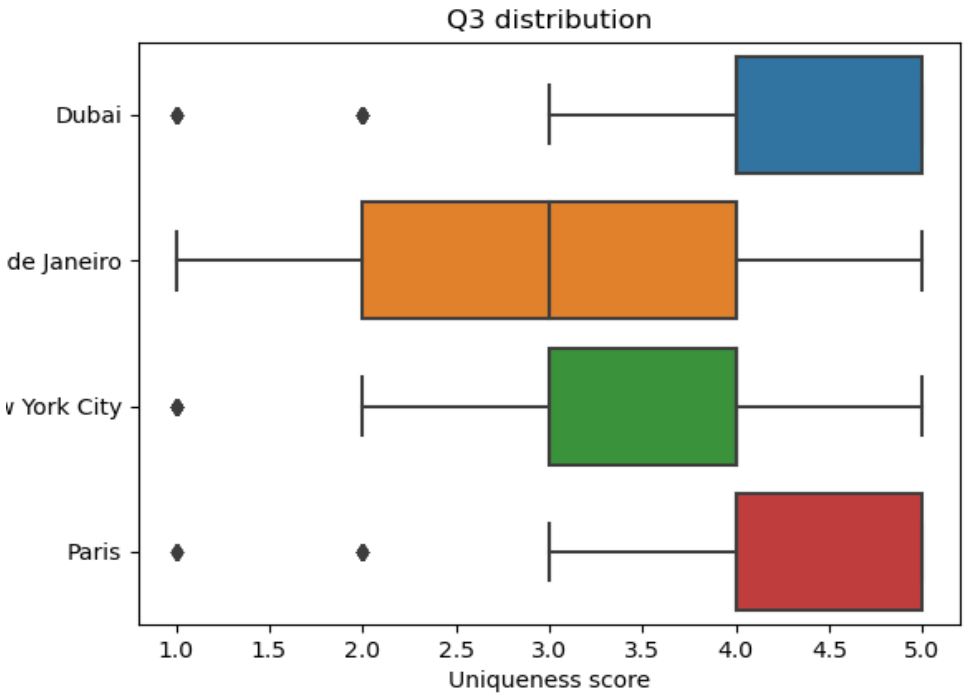
The original dataset contains 10 features. Based on the questionnaire, the first four questions ask people to evaluate the given topic about the city in a score range of 1 - 5. In order to see whether are these features useful for predicting a city, we perform a parallel comparison by creating 4 box plots for each question.



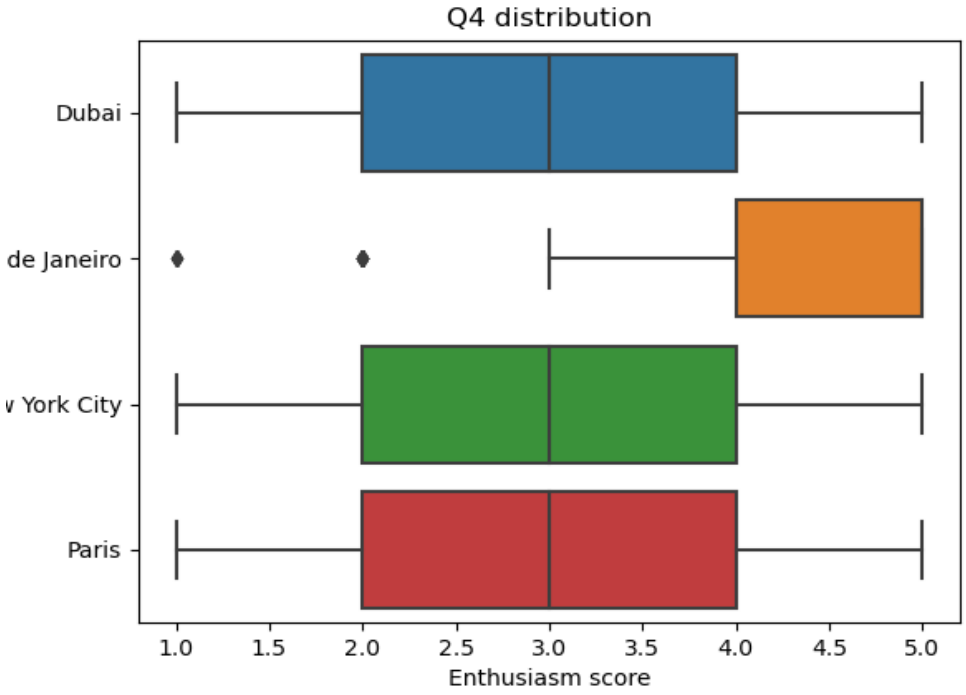
(Figure2) This figure is the box plot distribution of Q1. It depicts the distribution of a quantitative variable(popular score) across different cities.



(Figure3) This figure is box plot the distribution of Q2. It depicts the distribution of a quantitative variable(efficient score) across different cities.



(Figure4) This figure is the box plot distribution of Q3, It depicts the distribution of a quantitative variable(uniqueness score) across different cities.

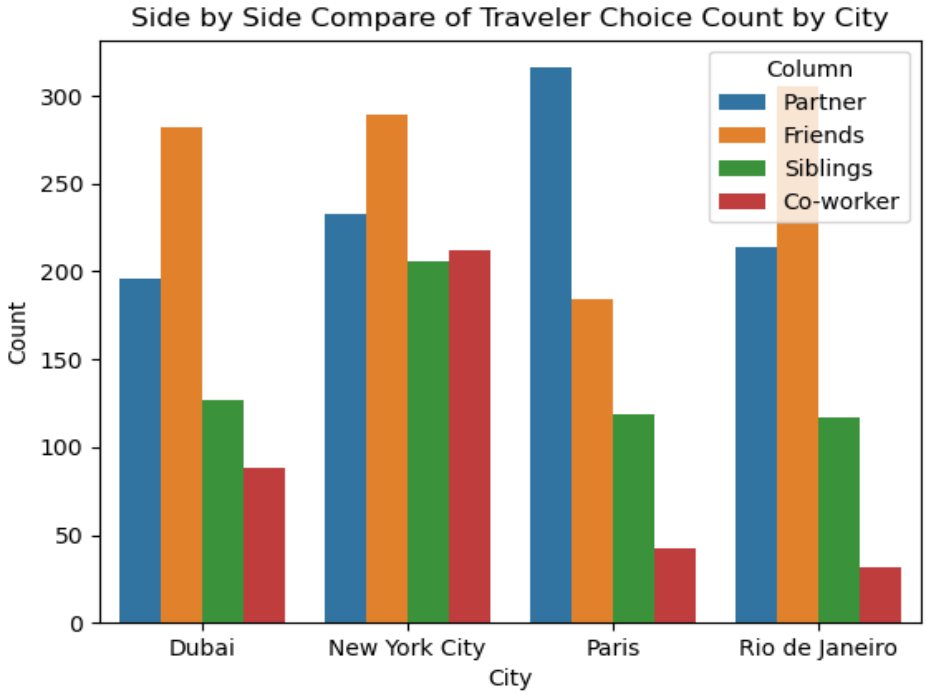


(Figure5) This figure is the box distribution of Q4, It depicts the distribution of a quantitative variable(enthusiasm score) across different cities.

Features 1, 2 and 3 all show a clear difference in scores in different cities. It means

the participants believe these cities are different in popularity, efficiency and architectural uniqueness. However, the distribution of feature 4 might indicate this feature is not useful, because Dubai, New York and Paris have almost the same score distribution.

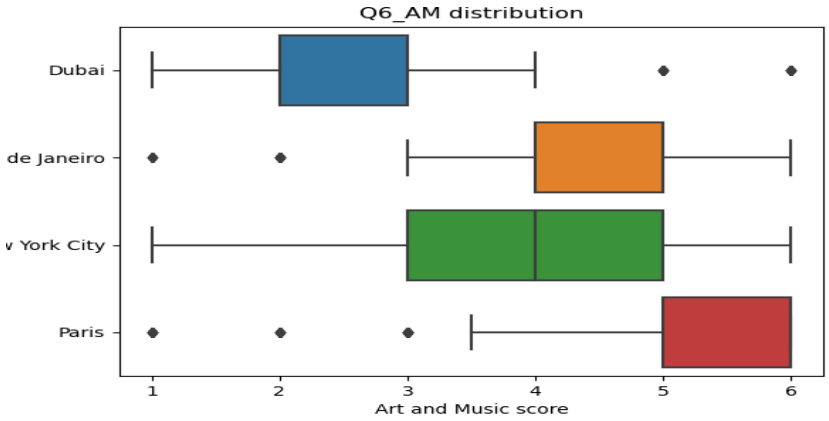
Question 5 is multiple choice that allows participants to choose who they want to travel to the city with. Because it is multiple choice, some participants choose more than one option. So, we can draw a bar plot to check how many people choose the option by different cities.



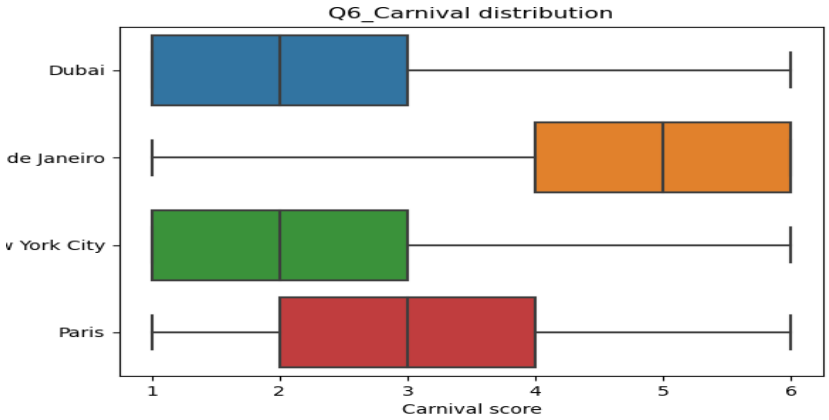
(Figure6) This figure is the bar plot distribution of Q5, It depicts the distribution of a categorical variable(uniqueness score) across different cities.

By looking through all the given traveller choices and comparing the number of participants chosen, we can see the number of people who chose Friends to travel to Dubai, New York and Rio, and the number of people who chose Siblings to travel to Dubai, Paris and Rio are almost equal. The number of people who choose a partner and co-worker might be helpful in distinguishing the city it refers to.

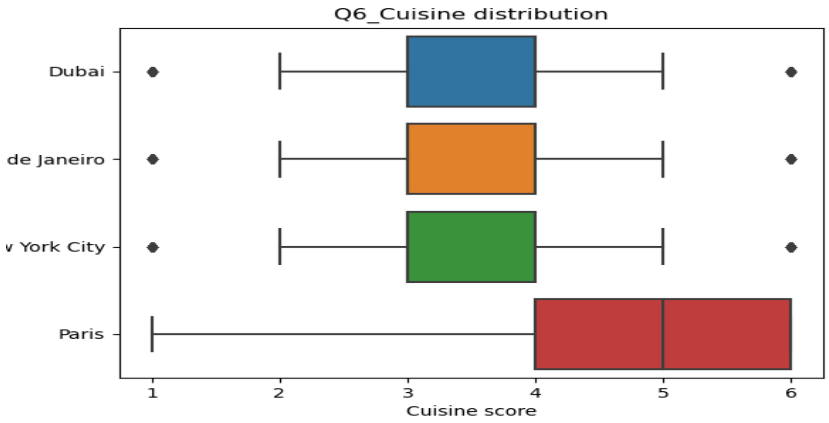
Question 6 requires participants to rank how the keyword related to the city, with 6 keywords provided. Because the numerical value is meaningful for this feature, we first treat it as a numerical value and draw 6 box plots for each keywords.



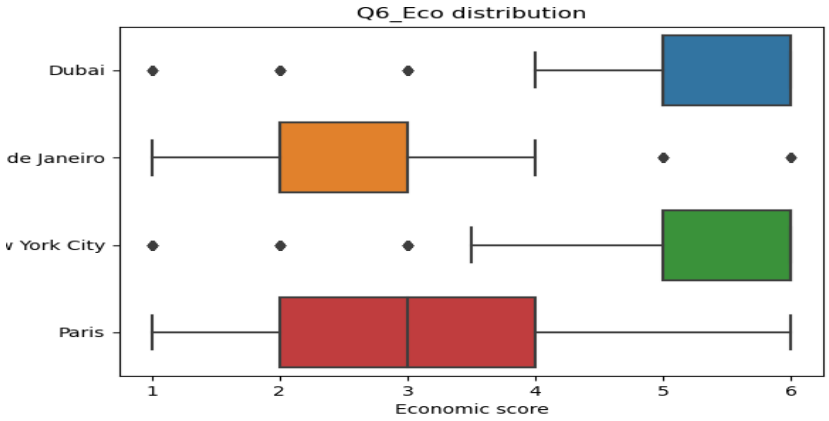
(Figure7) This figure is the box plot distribution of Q6, It depicts the distribution of a categorical variable(art and music score) across different cities.



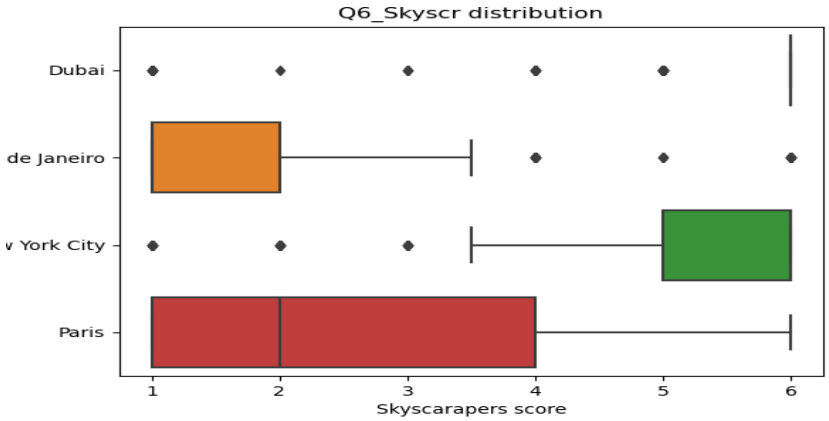
(Figure8) This figure is the box plot distribution of Q6, It depicts the distribution of a categorical variable(carnival score) across different cities.



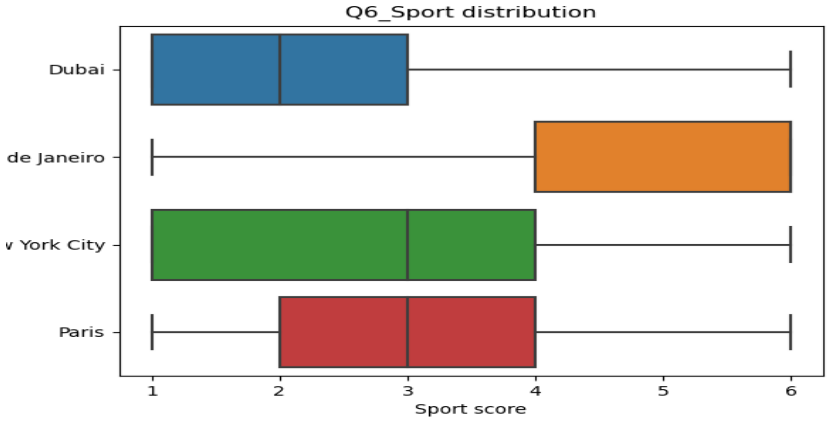
(Figure9) This figure is the box plot distribution of Q6, It depicts the distribution of a categorical variable(cuisine score) across different cities.



(Figure10) This figure is the box plot distribution of Q6, It depicts the distribution of a categorical variable(economic score) across different cities.



(Figure11) This figure is the box plot distribution of Q6, It depicts the distribution of a categorical variable(skyscraper score) across different cities.

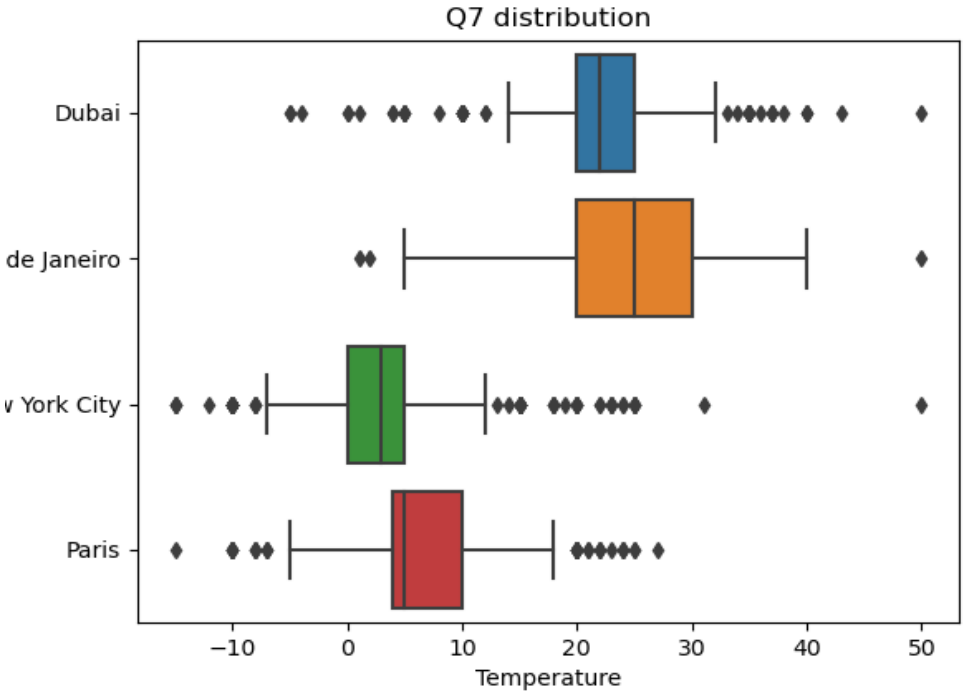


(Figure12) This figure is the box plot distribution of Q6, It depicts the distribution of a categorical variable(Sport score) across different cities.
Except for the keyword Cuisine, people rate this keyword equally for Dubai, Rio and New York, the score of the other 5 keywords indicates a different distribution in each city, which means it might be helpful for future prediction.

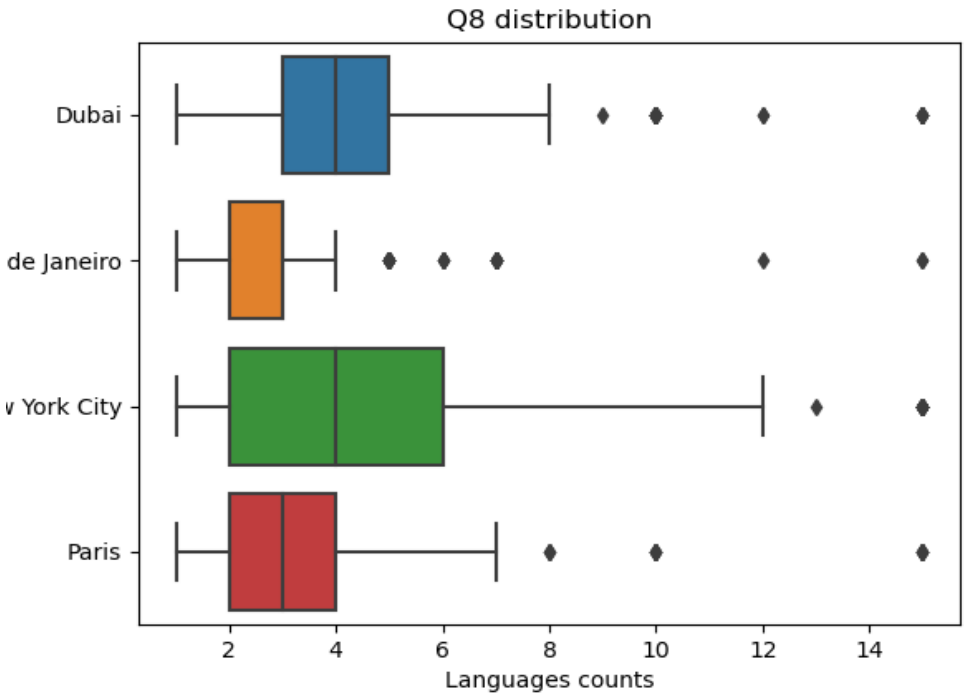
Questions 7, 8 and 9 ask participants to provide the temperature, number of languages and number of different fashion styles that can be recognized on the street in tuition. Because these questions are open-ended, outliers can be really common (according to ¡figure 1¿). So, before we dive into the feature value distribution, we need to carefully handle the outliers. Because the outlier can make the box plot unobservable.

Because the temperature, number of languages and different fashion styles all have a natural range. Instead of dropping the samples with outliers, we round the original value that is really big or small to the bound of the reasonable range. For example, the input of 10000 degrees for temperature will be rounded to 45 degrees. Because we believe a really high-temperature input might indicate the participant believes that this city is extremely hot.

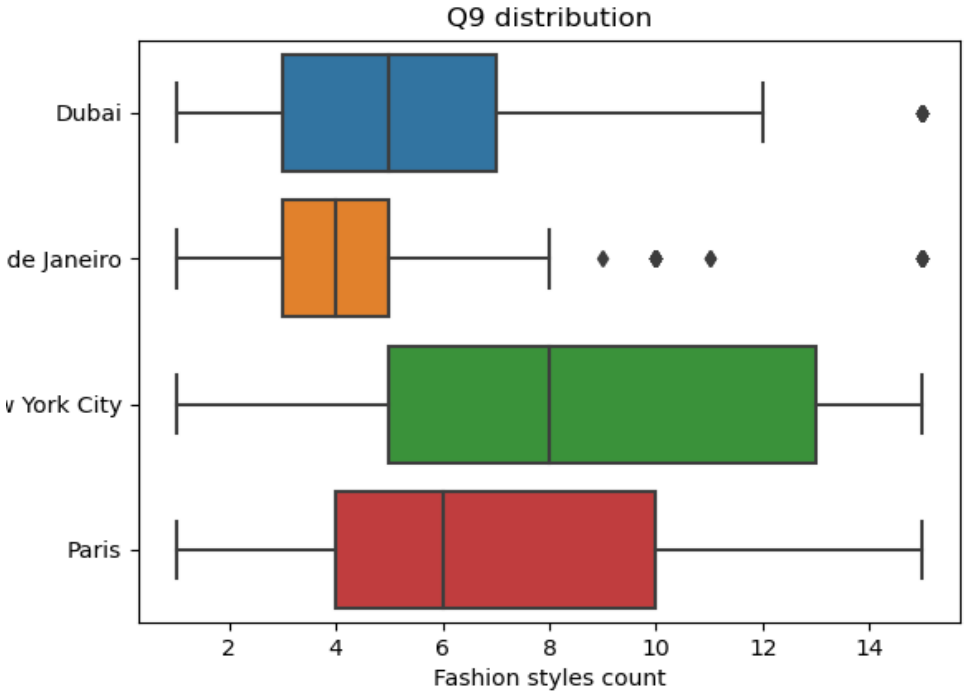
After we handle the outliers among questions 7, 8 and 9, we can draw the box plots for each feature.



(Figure13) This figure is the box plot distribution of Q7. It depicts the distribution of a quantitative variable(temperature) across different cities.



(Figure14) This figure is the box plot distribution of Q8, It depicts the distribution of a quantitative variable(language counts) across different cities.



(Figure15) This figure is the box plot distribution of Q9, It depicts the distribution of a quantitative variable(fashion style counts) across different cities.

The box plots for features 7, 8 and 9 all showed a clear difference in distribution for

different city.

Based on the original dataset, we need to do some cleaning processes to improve the prediction performance, or even be able to use it for training the model.

For questions 1-5, because these features in the dataset only have 5 or 4 different inputs, we use Onehot encoding to transform them into extra features. If there are missing values exist in these categorical features, the missing values of questions 1-4 will be filled by 2, and the missing values of question 5 will be filled as one of the possible choices randomly. We don't randomly choose a value for the missing value of the first 4 questions because the original setting of these features has a meaningful numerical size, and 2 is closer to the "mean". For questions 7-9, we will treat them as numerical explanatory variables. Based on this semester's lab, we know that normalization will improve the prediction accuracy. So, we fill all the missing values by the mean of each feature and normalize them. For question 6, the question asked participants to rank the given 6 keywords by correlation with the city. However, some participants are not ranking them, they rating these keywords. So, in some samples, the values are repeated. So, we have three different strategies for handling this misunderstanding. First, we only take the keyword with the maximum score and the keyword with the minimum score as input to the model. If two or more keywords have the same value, we randomly choose one of them. Second, we treat this feature as 6 numerical variables. We create 6 new variables and each variable represents one of the keywords, the value of each new variable will be the corresponding score in the original feature. The two strategies listed above will share the same missing value handling process, the missing values will be filled by 3.5 (mean), because they all treat the value of the original feature as numerical. Third, we ignore the numerical size for the actual ranking (a bigger number means more related), instead, we make 6 keywords all categorical. By using the Onehot encoding, we create 36 features to indicate different values for each keyword. The missing value will be handled differently. Because we treat the value as categorical, 3.5 (mean) will not be acceptable. We randomly choose from 3 or 4 to fill in the missing values from the original input feature.
Question 10 is the last feature that we need to handle, the question asks participants to write a quote to this city. Because the test data is unseen and we can't access the true label, we can't train BOW while predicting. Instead, we only use the bag of words method to create a vocabulary list and new features to the original dataset based on the vocabulary list. The value of each word represents whether it is present in the quote of the sample (1 = present, 0 otherwise). We decided to use the fixed pre-generated vocabulary list to fit the model and process any new given data because if any training processes are not allowed, the features and weight for each feature should be fixed. The main problem is that if the vocabulary list can't be generated based on the input dataset, some of the words in the original vocabulary list will never be present in the new dataset, and some words in the new dataset are also not in the vocabulary list. This possibly leads to many columns being all zeros, which means the feature is useless for prediction. We will try to use this strategy to train the model and compare the test accuracy. If it makes the model overfit, we will ignore question 10 while fitting the model.

In order to train the prediction model, we need to split the original dataset into training, validation and test sets. To make the choice of hyper-parameter generalize enough, we randomly shuffle the dataset before splitting it. Also, the data-splitting process will take care of transforming the type of dataset from the Pandas data series to the NumPy array, which is more friendly for matrix calculation.

2 Model Evaluation and Exploration

Our project aims to predict which city a person is thinking of, based on their subjective responses to a series of questions. We approached this challenge by exploring three distinct model families: Random Forest, Neural Network, and K-Nearest Neighbors (KNN), each offering unique strengths for handling the nuanced dataset compiled from the survey responses.

2.1 K-Nearest Neighbors (KNN)

We selected the K-Nearest Neighbors(KNN) algorithm for its simplicity and effectiveness in handling the classification problems, especially given the characteristics of our dataset, which includes a mix of continuous variables(e.g., the number of unique languages of a city) and categorical variables(e.g., rating the popularities of a city). One of the primary advantages of KNN is its intuitiveness: it classifies a given data point based on the majority class among its k-nearest neighbors. This approach is particularly beneficial for our project because it allows us to leverage the natural clustering of data points in the feature space, making it highly applicable to scenarios where relationships between data points are indicative of their categorization. We experimented with different number of k(number of neighbors) and distance metrics(Euclidean, Manhattan, and cosine similarity), focusing on maxmizing the model accuracy and avoiding overfitting.

2.2 Neural Network

Another option we considered for our prediction model was a Neural Network, as it allows for the updating of weights and biases through backpropagation during the training process. In our code, we implemented two approaches to construct Neural Networks for our prediction model.The initial method utilizes functions from the scikit-learn library, while the second approach leverages the PyTorch library. Although these methods share similarities, they resulted in different accuracies after the training process.

For the SKLearn library generating the model, we have those steps:

Load and Shuffle Data: Read the dataset and shuffle it to ensure random distribution.

Reserve Test Set: Separate out a subset of the data for final evaluation.

Feature Extraction: Use custom functions to extract text features and combine them with numerical features.

Hyperparameter Grid: Define a set of hyperparameters to tune the model.

Cross-Validation: Implement K-Fold cross-validation to train and validate the model, ensuring generalizability.

Model Training: Within each fold, preprocess the data, train the MLPClassifier, and predict outcomes.

Evaluation: Compute average accuracy across all folds and analyze with a confusion matrix.

Final Test: Evaluate the model on the reserved test set to assess real-world performance.

Which we will further talk about in the model choice section.

2.3 Random Forest

We chose the Random Forest model for its robustness in handling varied data types and its capability to manage overfitting. Given our data's categorical nature and the presence of ordinal scales (e.g., city popularity ratings, architectural uniqueness), Random Forest's ensemble approach allowed us to capture complex patterns without being overly sensitive to noise. We strategically varied the number of trees and their depth, aiming to find an optimal balance that minimized bias and prevented overfitting—a critical consideration given the unseen nature of our test set. Our application of the Random Forest model involved extensive experimentation with these parameters, enabling us to refine our approach based on performance metrics such as accuracy, precision, recall, and the F1 score on cross-validation set.

We use the SKLearn library to generate the random forest model:

Preprocess: Extract features from text using custom functions and combine them with numerical data.

Cross-Validation Setup: Use KFold to split data into training and validation sets.

Model Training: Train RandomForestClassifier on each fold.

Evaluation: Predict and calculate accuracy for each fold, then average it out.

Which we will further talk about in model choice section

3 Model Choice and Hyperparameters

Evaluation Consistency

To guarantee the reliability and consistency of our evaluation metrics for all of our models, we employed K-fold cross-validation, a robust statistical method that maximizes both the training and testing utility of the available dataset. This approach ensures that each observation is used for both training and validation exactly once, which mitigates the impact of dataset partitioning variance on the evaluation metrics.

For our dataset of around 1,400 instances, k = 5 was chosen for K-Fold Cross-Validation for all of our models to ensure computational efficiency while maintaining a balance in model performance evaluation. Those number of folds provides a sufficient amount of data in each training and validation set, thus avoiding overfitting while allowing for a robust assessment of the model. Furthermore, k = 5 is a common practice for datasets of similar size, making it an appropriate choice.

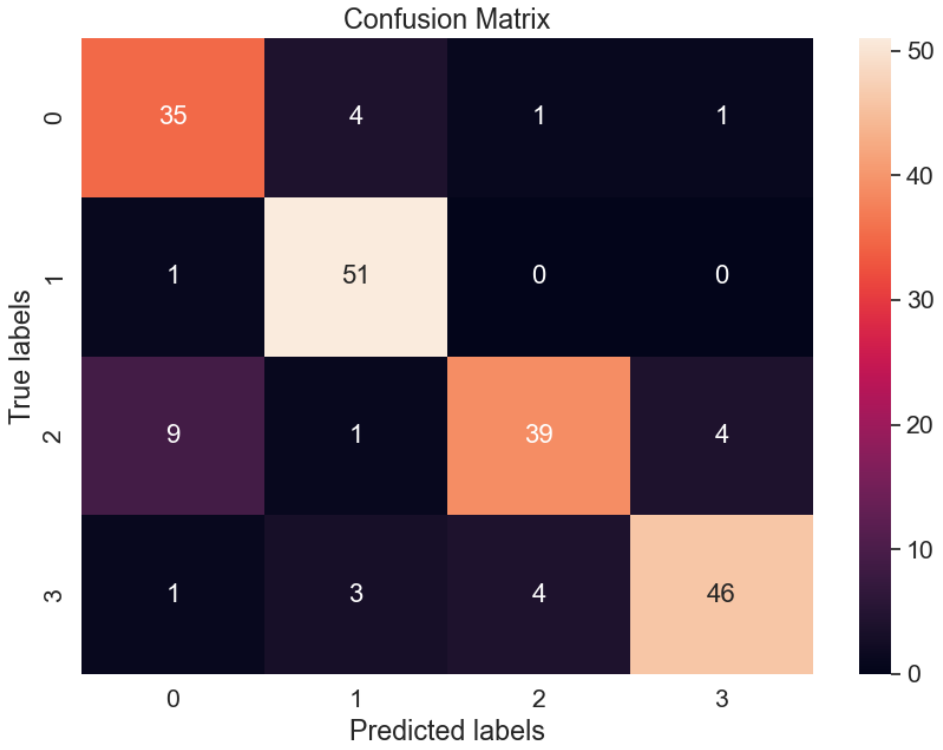
3.1 K-Nearest Neighbors (KNN)

Evaluation Metrics

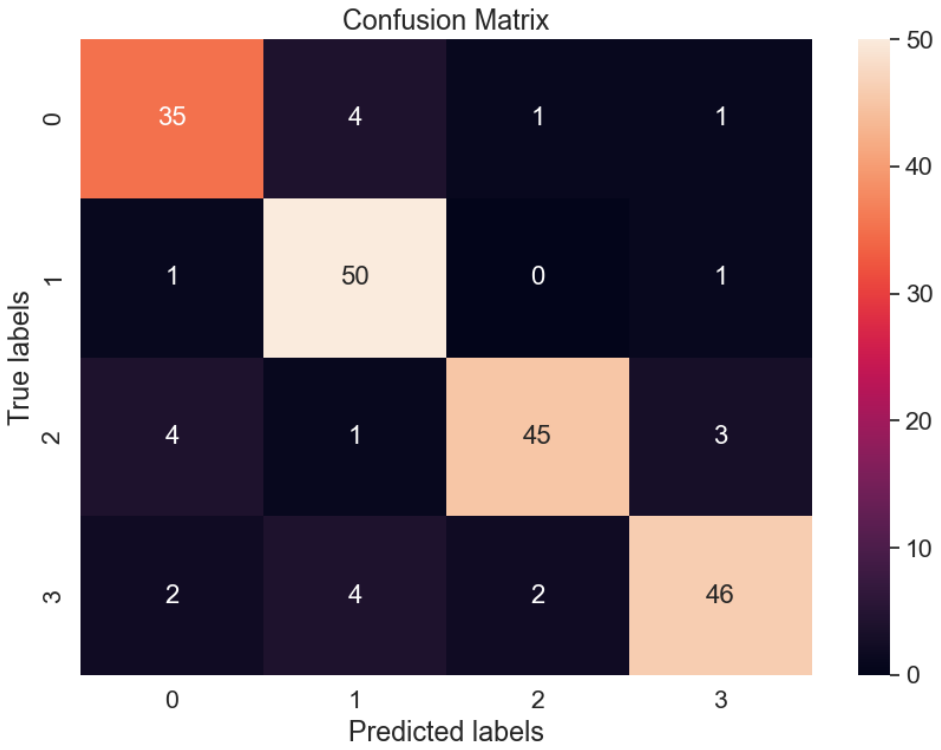
To optimize our KNN model, we focused on selecting the appropriate number of neighbors (k) and the distance metric which is used to identify these neighbours. The choice of k is significant to KNN performance. We experimented with various values of k to strike a balance between model complexity and generalization capability. A smaller k value makes the model sensitive to noise, leading to overfitting, while a larger k value might oversimplify the model, increasing bias. Similarly, we evaluated different distance metrics (Euclidean, Manhattan, and cosine similarity) and chose the most appropriate one for this project. (formula)Euclidean distance measures the distance between two data points. It suffers the curse of dimensity and is sensitive to data outliers, which means the big difference between data points will lead to a total different result. The other metric, Manhattan distance, calculates the sum of the absolute differences between data points. It is more offers robustness in higher dimensions with less sensitivity to outliers. Cosine similarity measures the cosine of the angle between two non-zero vectors in an inner product space, providing a measure of orientation similarity rather than magnitude. It is ideal for text analysis and frequency-based data where the orientation of the vectors is more important than their magnitude and is effective in high-dimensional spaces. Considering our dataset, which contains almost 1500 features, the challenges of high dimensionality and outliers could significantly affect our model's performance. Thus, the metric less affected by these factors is more desirable.
Besides, To evaluate the performance of the KNN model using different k and distance metrics - Euclidean, Manhattan, and Cosine Similarity- we employed two evaluation metrics: the confusion matrix and test accuracy. These two metrics offer a detailed view of the model's predictive capability and overall effectiveness in classifying unseen data.
In each Knn model, We used 5 fold cross validation to split our train and valid data set while fixed a test data set with 200 data sets. During the k-fold process, our data was randomly splitted to 5 ralative equal parts. One part for validation set, the remainings

are the training set. We have a iteration(k) range from 1 to 30. For each iteration we fit our model on the training data and record the validation accuracy for current k. Then we get the k with the highest validation accuracy and store the Knn model with the correspond train data set. Based on the optimistic KNN models we generated for each distance metric, we generated the confusion matrix by computing the prediction using the test data set on the knn models.

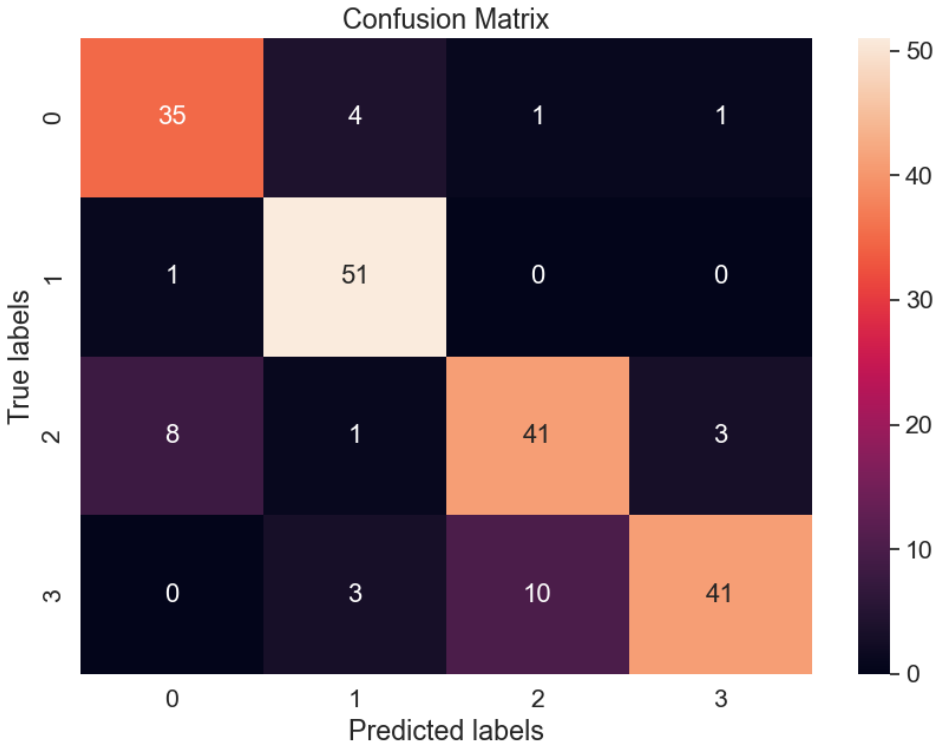
Aggregated Confusion Matrix Analysis



(Euclidean)The confusion matrix (Figure 1) illustrates the number of correct and incorrect predictions made by the classifier with respect to the actual classes in the KNN model with the euclidean distance metric.



(Manhattan)The confusion matrix (Figure 2) illustrates the number of correct and incorrect predictions made by the classifier with respect to the actual classes in the KNN model with the Manhattan distance metric.



(Cosine Similarity)The confusion matrix (Figure 3) illustrates the number of correct

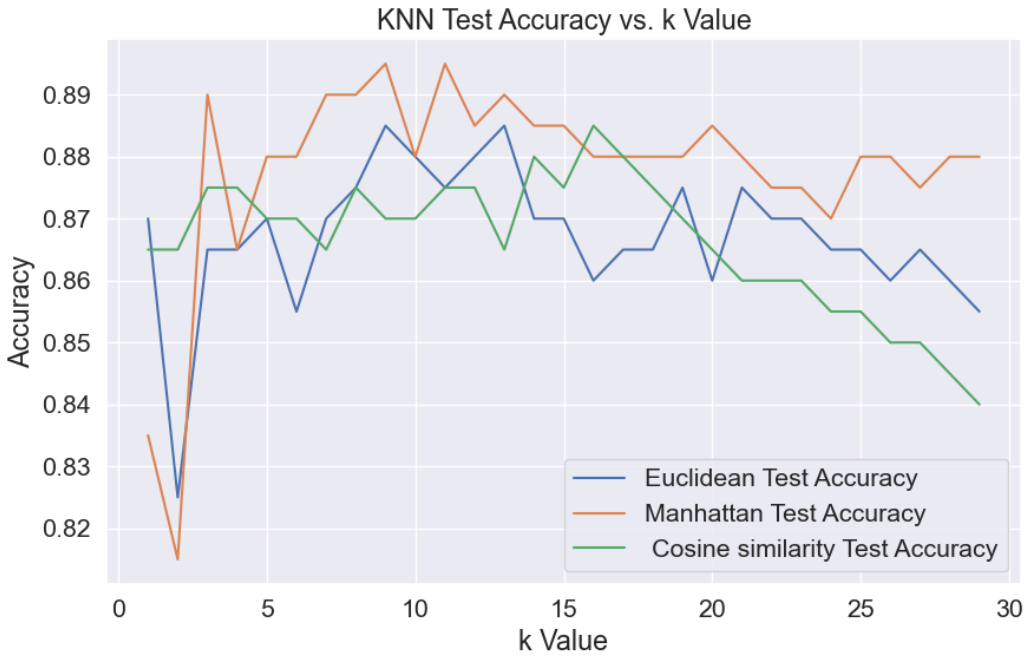
and incorrect predictions made by the classifier with respect to the actual classes in the KNN model with the Cosine similarity metric.

- **True Positives (TP)**: The diagonal elements of the matrix indicate the number of instances that were correctly identified for each class.
- **False Positives (FP)**: The off-diagonal elements in the columns indicate instances that were incorrectly predicted as belonging to a class.
- **False Negatives (FN)**: The off-diagonal elements in the rows indicate instances of a particular class that were predicted as belonging to a different class.
- **True Negatives (TN)**: Although not directly shown, these are the instances that were correctly identified as not belonging to a particular class. This is represented by the sum of all values not in the row and column for that class.

(Figure4)The precision of KNN model with three distance metrics. Based on the results in the figure4, it illustrates that the second model has higher precision than the second

model, which means there are more false positives for each class in the the second one(Manhattan).

Test Accuracy

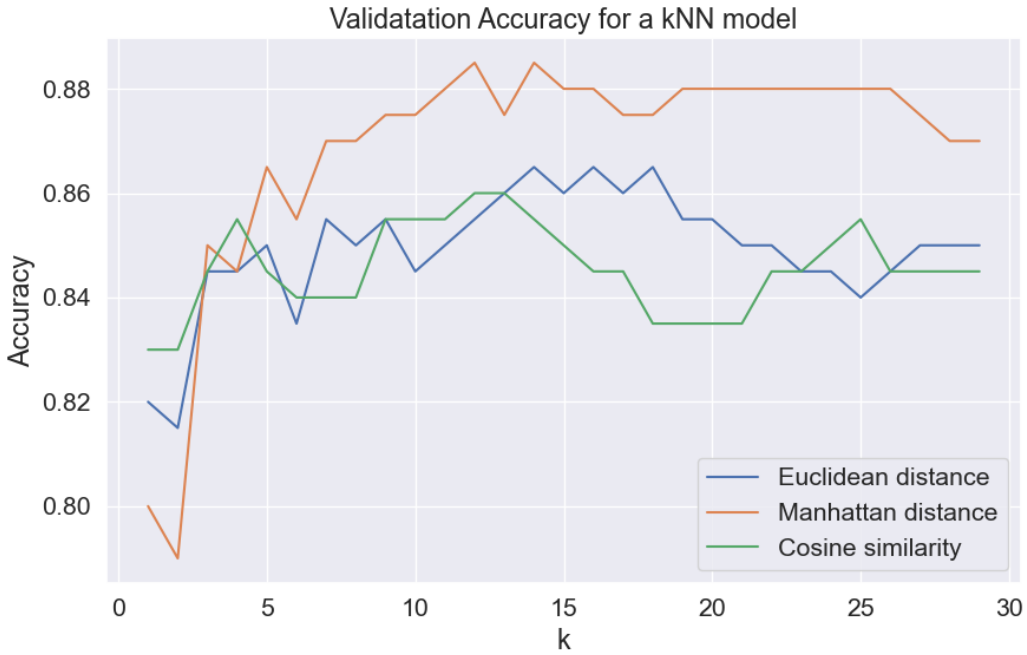


(Figure5) The test accuracy of three models across different k. For each iteration of the cross validation, we use the current k to fit a model using the training dataset and record the test accuracy. There are three lines of test accuracy. The figure illustrates that most Manhattan's test accuracies are higher than the other two models.

Hyperparameter Tuning

In the process of tuning the hyperparameter of the KNN, we adjusted the following parameters:

- **k**:the number of neighbors, range(1, 30)
- **distance metric**: three different distance metrics: Euclidean distance, Manhattan distance, and cosine similarity.



(Figure6) This is a validation accuracy across different k for three knn models. It illustrates that the Knn model with the Manhattan distance have the highest validation accuracy after k = 5 among all the knn models.
Through the $k - fold$ cross validation process, we got the optimistic k for each knn model. The optimistic k for each mdodel is: 13, 11, and 9.The corresponding test accuracies are: 0.88, 0.9, and 0.895. Thus, combining the proporty of Manhattan with the test accuracy of Manhattan and the presicion of Manhattan, the KNN model with Manhattan distance metric is the optimistic among the three models.
However, Knn model is a simple algorithm that relies on distance metrics, cannot model complex patterns as effectively, especially in high-dimensional spaces. And KNN is less efficiency than the other models since KNN needs to compute the distance to each training sets. Besides, the test accuracy of KNN model is still lower than the other models. KNN model is not good enough for this project compare to the other models.

3.2 Neural Network

Evaluation Metrics

The primary metric used to evaluate the model is the accuracy score, which provides a simple yet effective measure of performance, especially when the classes are balanced. However, given that accuracy alone can be misleading sometimes, we also examined precision, recall, and F1-score as reflected in the aggregated classification report.

The Aggregated Confusion Matrix and Classification Report represent the average outcomes from a 5-fold cross-validation process. This aggregation is key to providing a robust evaluation that neutralizes the effects of data partitioning bias. Each data point influences the final metrics equally, allowing for a comprehensive and unbiased assessment of the model's overall predictive accuracy, thus ensuring a consistent and fair evaluation.

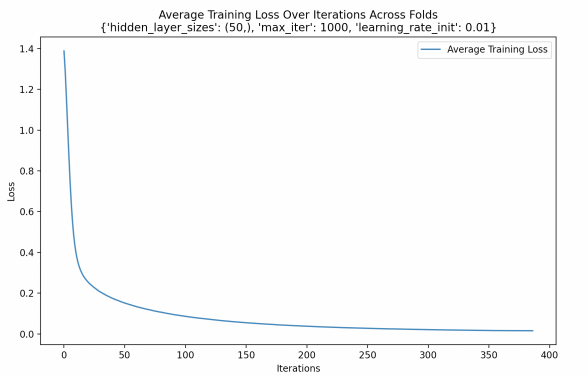
Hyperparameter Tuning

```
hyperparameters_grid = [
    # Variation in hidden layer sizes
    {'hidden_layer_sizes': (50,), 'max_iter': 1000, 'learning_rate_init': 0.01},
    {'hidden_layer_sizes': (150,), 'max_iter': 1000,  'learning_rate_init': 0.01},
    {'hidden_layer_sizes': (50, 100), 'max_iter': 1000,  'learning_rate_init': 0.01},

    # Variation in learning_rate_init
    {'hidden_layer_sizes': (150,), 'max_iter': 1000, 'learning_rate_init': 0.005},
    {'hidden_layer_sizes': (150,), 'max_iter': 1000, 'learning_rate_init': 0.02},

    # Experimenting with both large and more complex networks
    {'hidden_layer_sizes': (250,), 'max_iter': 1000, 'learning_rate_init': 0.01},
    {'hidden_layer_sizes': (100, 150), 'max_iter': 1000, 'learning_rate_init': 0.01},
]
```

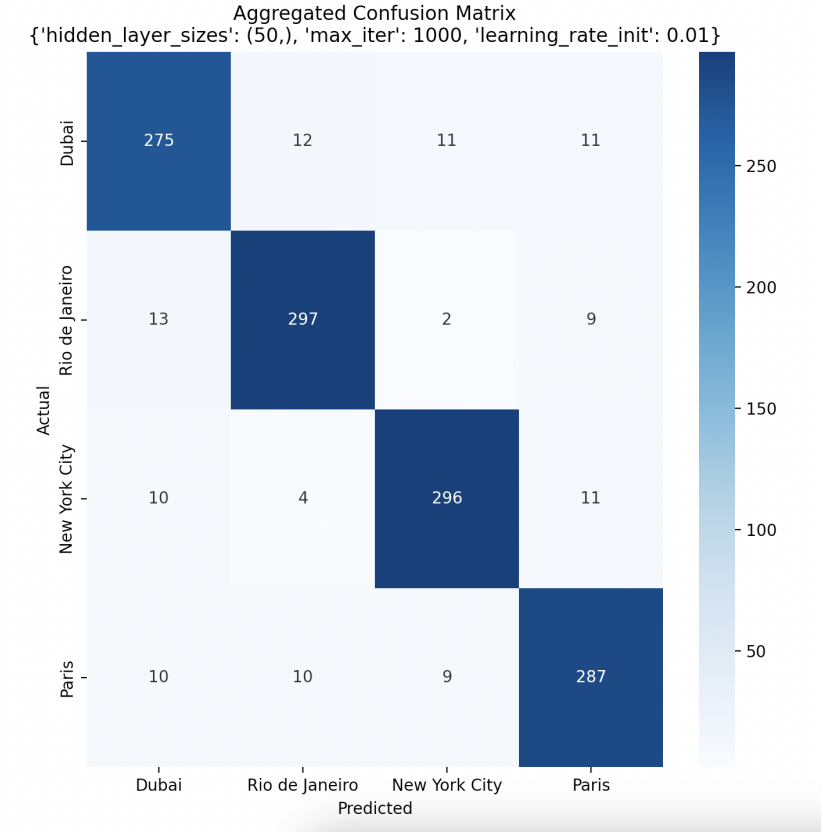
The code utilizes a grid search combined with 5-Fold cross-validation to tune hyperparameters for an MLPClassifier model. For each set of hyperparameters in the grid, it partitions the data into five folds, training a model on four folds and validating it on the remaining one. This process ensures every data split serves as a validation set once. The performance of these models, evaluated across all folds, is then averaged to select the best hyperparameter combination, factoring in metrics like accuracy and loss. The approach, by iteratively training and evaluating models across different data subsets, aims to enhance model generalizability and performance. Notice that for all hyperparameter combinations, we use the same 5 data splits to ensure validation consistency.

Here are some of the hyperparameter combinations we have explored, with the corresponding evaluation metrics:

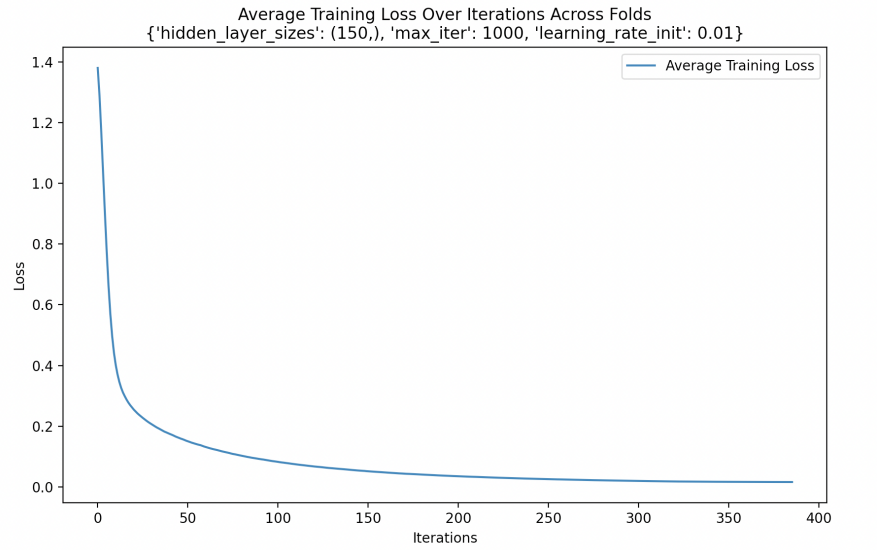
Accuracy report and confusion matrix with hyperparameters: 'hidden_layer_sizes': (50,), 'max_iter': 1000, 'learning_rate_init': 0.01



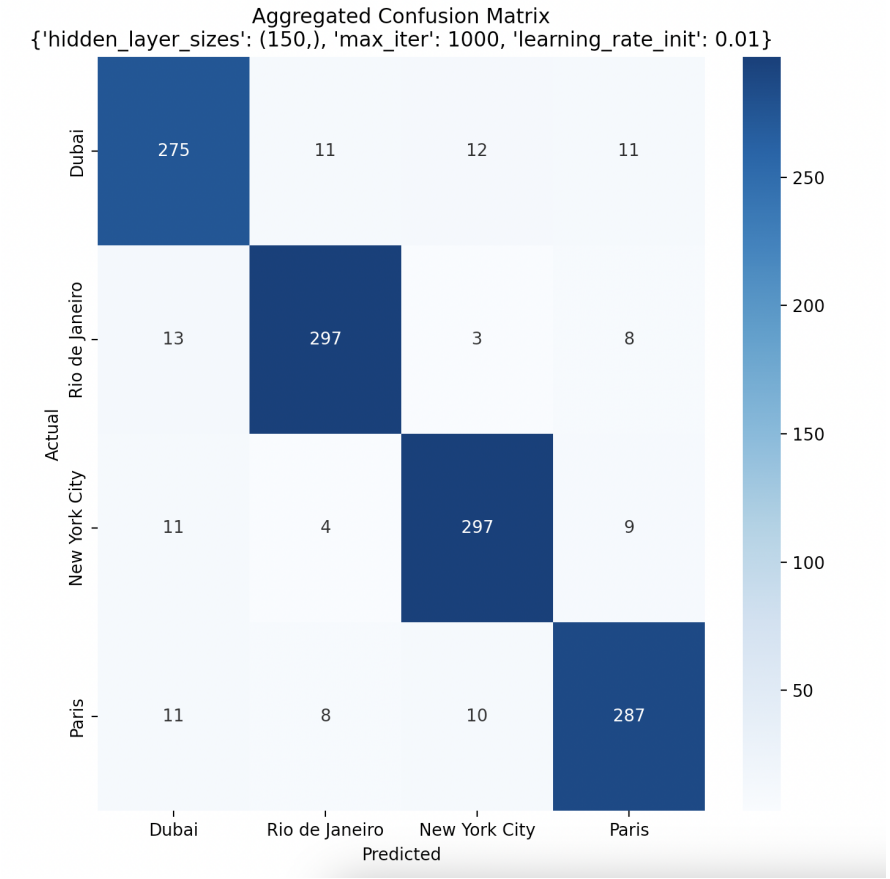
```
Iteration 362, loss = 0.01598164
Iteration 363, loss = 0.01591882
Iteration 364, loss = 0.01584966
Iteration 365, loss = 0.01579582
Iteration 366, loss = 0.01572682
Iteration 367, loss = 0.01566521
Iteration 368, loss = 0.01558481
Iteration 369, loss = 0.01551774
Training loss did not improve more than tol=0.000100 for 10 consecutive epochs. Stopping.
Average K-Fold Accuracy: 0.9092
              precision    recall  f1-score   support

           0       0.88      0.89      0.88       309
           1       0.92      0.92      0.92       321
           2       0.92      0.92      0.92       321
           3       0.91      0.91      0.91       316

    accuracy                           0.91      1267
   macro avg       0.91      0.91      0.91      1267
weighted avg       0.91      0.91      0.91      1267

_
```



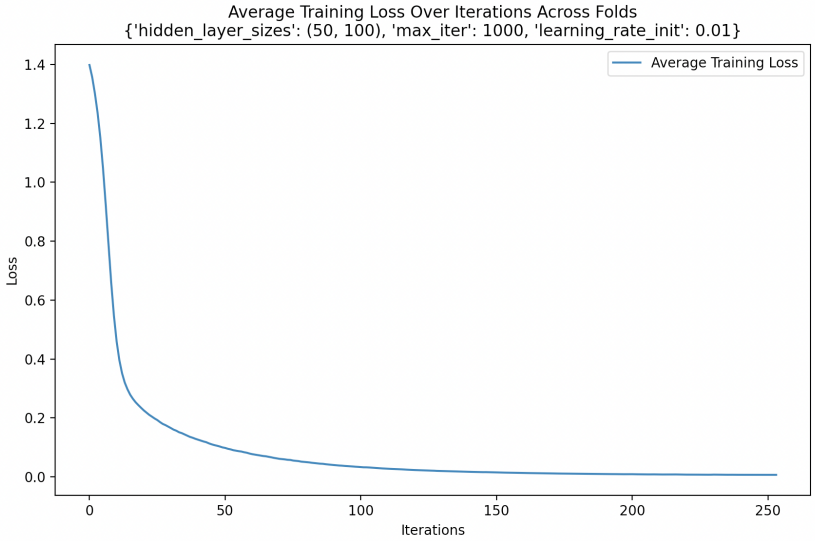
Accuracy report and confusion matrix with hyperparameters: 'hidden_layer_sizes': (150,), 'max_iter': 1000, 'learning_rate_init': 0.01



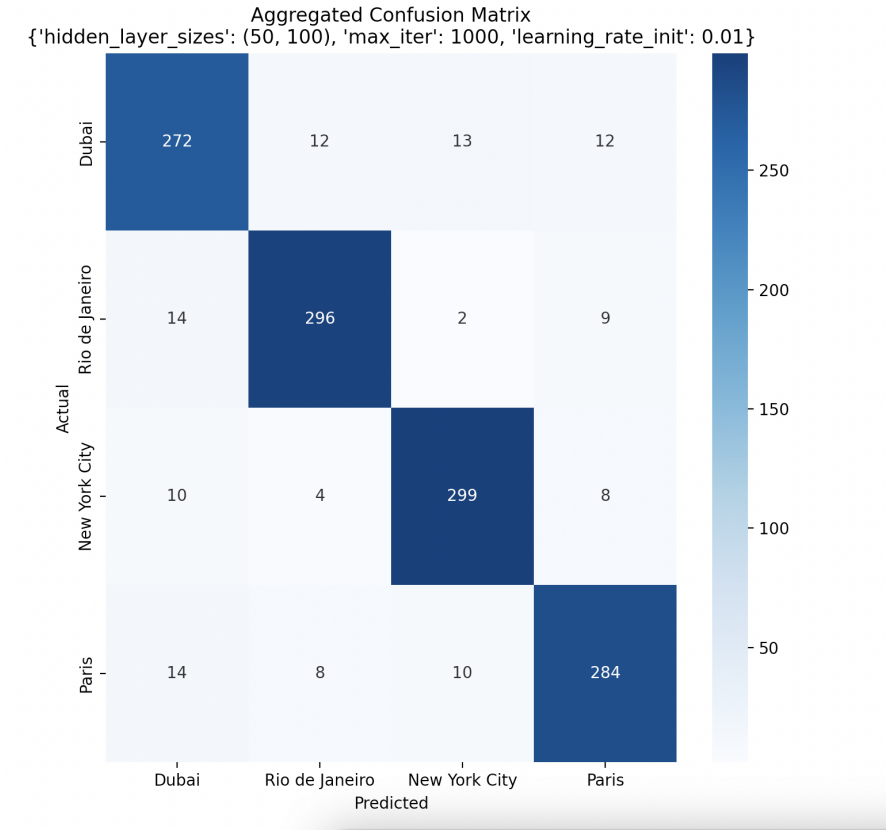

```
Iteration 333, loss = 0.01680443
Iteration 334, loss = 0.01671980
Iteration 335, loss = 0.01664360
Iteration 336, loss = 0.01664637
Training loss did not improve more than tol=0.000100 for 10 consecutive epochs. Stopping.
Average K-Fold Accuracy: 0.9124
              precision    recall  f1-score   support

           0       0.89      0.89      0.89       309
           1       0.93      0.93      0.93       321
           2       0.92      0.93      0.92       321
           3       0.91      0.91      0.91       316

    accuracy                           0.91      1267
   macro avg       0.91      0.91      0.91      1267
weighted avg       0.91      0.91      0.91      1267
```



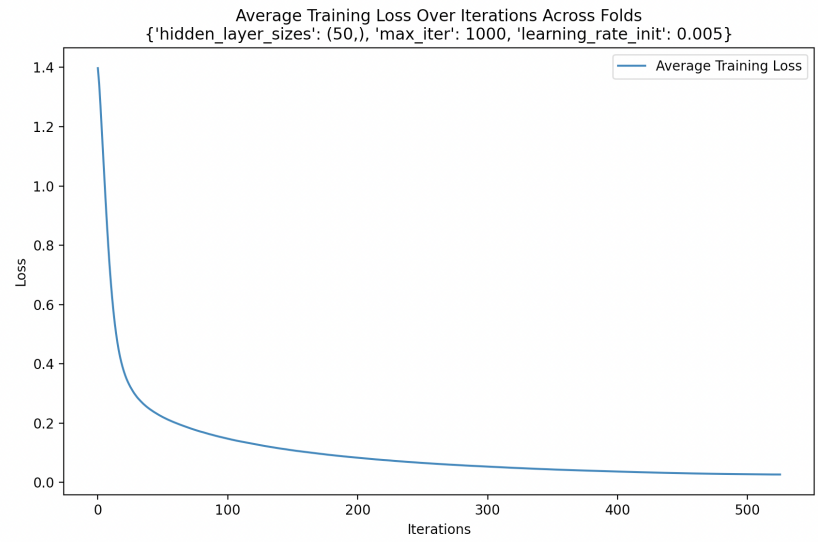
Accuracy report and confusion matrix with hyperparameters: 'hidden_layer_sizes': (50, 100), 'max_iter': 1000, 'learning_rate_init': 0.01



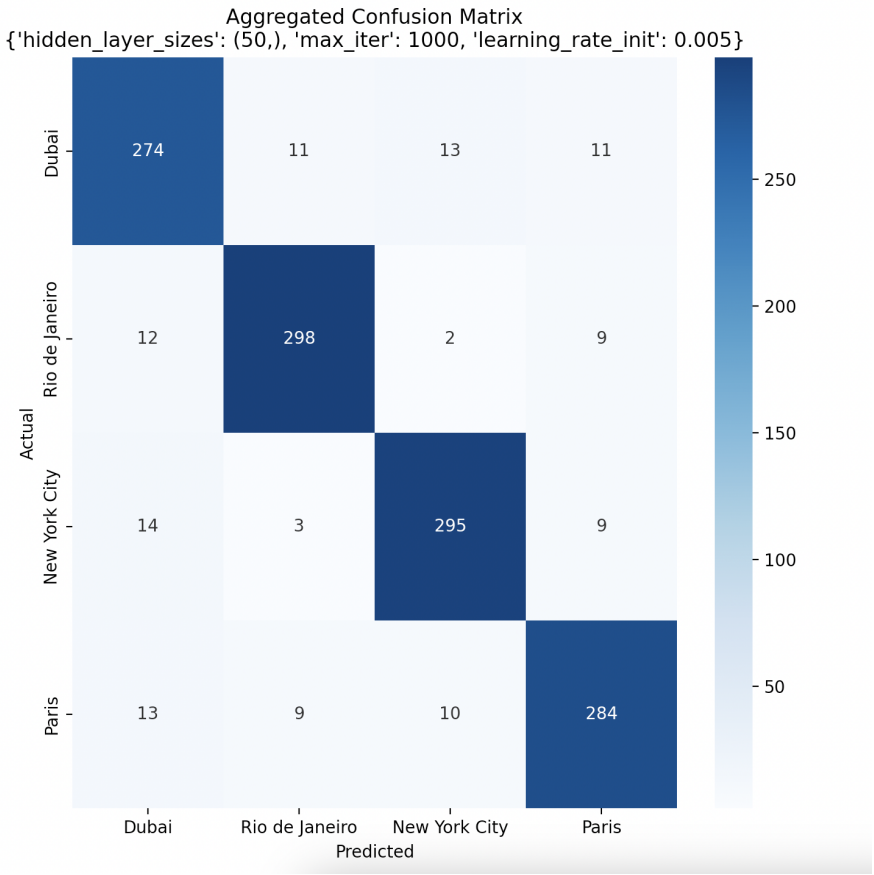
```
Iteration 233, loss = 0.00712636
Iteration 234, loss = 0.00707726
Iteration 235, loss = 0.00712708
Iteration 236, loss = 0.00703742
Training loss did not improve more than tol=0.000100 for 10 consecutive epochs. Stopping.
Average K-Fold Accuracy: 0.9092
              precision    recall  f1-score   support

           0       0.89      0.87      0.88       309
           1       0.93      0.93      0.93       321
           2       0.92      0.93      0.92       321
           3       0.91      0.91      0.91       316

    accuracy                           0.91      1267
   macro avg       0.91      0.91      0.91      1267
weighted avg       0.91      0.91      0.91      1267
```



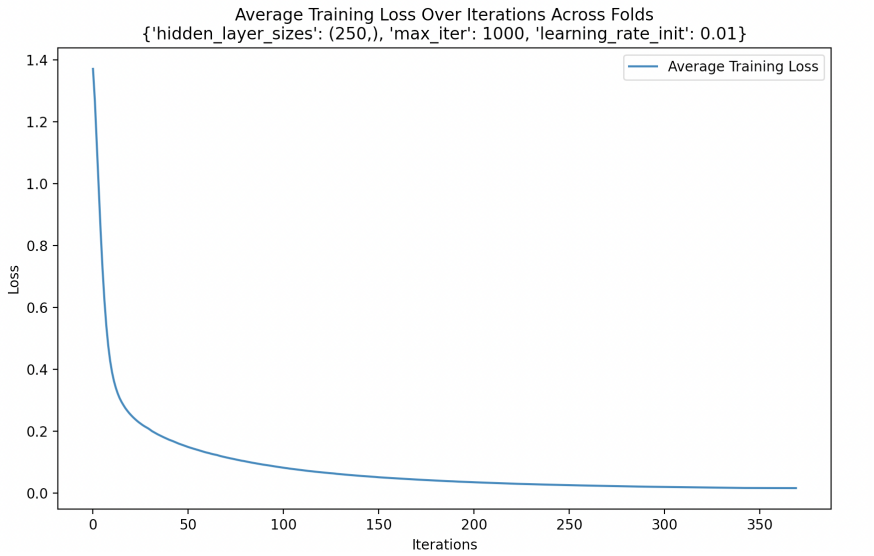
Accuracy report and confusion matrix with hyperparameters: 'hidden_layer_sizes': (50), 'max_iter': 1000, 'learning_rate_init': 0.005



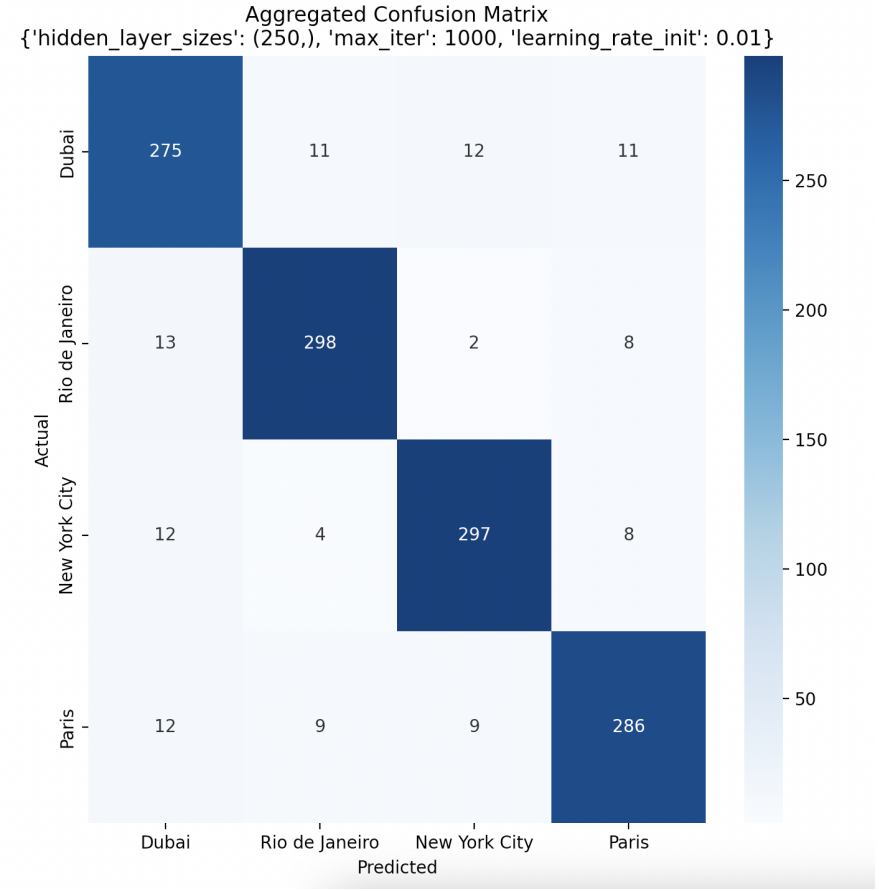

```
Iteration 479, loss = 0.02793927
Iteration 480, loss = 0.02787024
Iteration 481, loss = 0.02778037
Iteration 482, loss = 0.02769720
Iteration 483, loss = 0.02760754
Training loss did not improve more than tol=0.000100 for 10 consecutive epochs. Stopping.
Average K-Fold Accuracy: 0.9084
              precision    recall  f1-score   support

           0       0.88      0.89      0.88       309
           1       0.93      0.93      0.93       321
           2       0.92      0.92      0.92       321
           3       0.91      0.90      0.90       316

    accuracy                           0.91      1267
   macro avg       0.91      0.91      0.91      1267
weighted avg       0.91      0.91      0.91      1267

_
```



Accuracy report and confusion matrix with hyperparameters: 'hidden_layer_sizes': (250), 'max_iter': 1000, 'learning_rate_init': 0.01



```
Iteration 344, loss = 0.01548282
Iteration 345, loss = 0.01541171
Iteration 346, loss = 0.01532157
Iteration 347, loss = 0.01525077
Iteration 348, loss = 0.01518031
Training loss did not improve more than tol=0.000100 for 10 consecutive epochs. Stopping.
Average K-Fold Accuracy: 0.9124
              precision    recall  f1-score   support

           0       0.88      0.89      0.89       309
           1       0.93      0.93      0.93       321
           2       0.93      0.93      0.93       321
           3       0.91      0.91      0.91       316

    accuracy                           0.91      1267
   macro avg       0.91      0.91      0.91      1267
weighted avg       0.91      0.91      0.91      1267
```



Note that we have also tried the learning rate 0.005 to 0.02, a learning rate of 0.005 takes too many iterations to converge, whereas a learning rate of 0.02 jumps too quickly. Therefore we have chosen the learning rate of 0.01.

Furthermore, SKLearn will stop the iteration when the training loss does not improve for 10 consecutive epochs, therefore we set a maximum iteration.

Choice of hyperparameter and justifications:

Confusion Matrix Analysis:

True Positives: High diagonal values indicate a strong ability of the classifier to correctly identify each class.

False Positives and Negatives: Relatively low off-diagonal values show few instances where the model incorrectly predicts the actual class.

Balance: Comparable figures across classes suggest no particular bias toward any class.

Classification Report Analysis:

Precision: High precision values denote a low count of false positives.

Recall: High recall values reflect the classifier's effectiveness in identifying true positives.

F1-Score: Consistent F1-scores across classes point to a balanced recall and precision.

Support: Even distribution confirms a balanced dataset. Overall metrics, with macro and weighted averages at around 0.91 and accuracy of the model at 0.9124, underline a high-performing and well-generalizing classifier.

From the confusion matrix and the different layer we can see, the one with the hyperparameter as **'hidden_layer_sizes': (150,), 'max_iter': 1000, 'learning_rate_init': 0.01** give a higher overall performance. Other hyperparameters are either too simple to capture the complexity of the data or too complex and capture some noise, making them not as good a hyperparameter combinations as the one we have chosen.

The hyperparameter combination have been selected based on a balance between complexity and computational efficiency. The hidden layer size of 150 neurons is substantial enough to capture complex patterns in the data but not so large as to make the model prone to overfitting or excessive training times. A maximum of 1000 iterations provides the model with ample opportunity to converge to a solution, and a learning rate of 0.01 is a middle ground that avoids the slow convergence of smaller learning rates and the instability of larger ones.

The reported average K-Fold accuracy of 0.9124 is quite high, indicating that the model performs well across different subsets of the data. This level of accuracy demonstrates that the model is able to generalize from the training data to unseen data, which is critical for real-world applications.

Looking at the precision, recall, and f1-score for each class, all values are close to or above 0.9, which suggests that the model has a good balance between sensitivity and specificity across all classes. This is supported by the confusion matrix, which shows relatively few misclassifications.

The confusion matrix further reveals that the model is mostly accurate with a relatively even distribution of errors across classes, implying that there's no significant bias towards any particular class. Most errors appear to be between the classes 'Dubai' and 'Paris', as well as 'New York City' and 'Rio de Janeiro', suggesting some possible similarities between these locations that the model is confusing.

In conclusion, the chosen hyperparameters provide a good balance between learning capacity and generalizability, as evidenced by the high accuracy and balanced perfor-

mance across different classes. Future steps might include further exploring the feature space or model architecture to address the specific misclassifications observed.

Neural Network Model Evaluation:

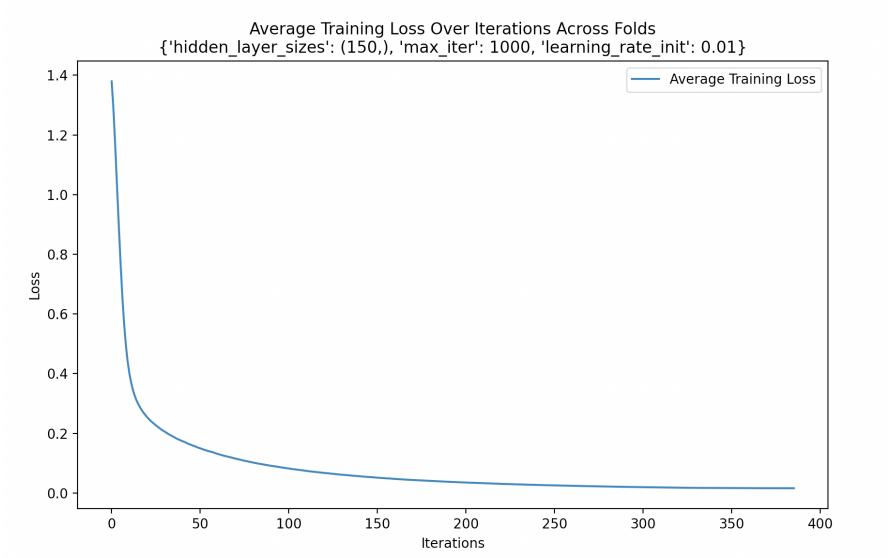
For our final model evaluation, the main evaluation metric I used to assess my model is accuracy, which measures the proportion of correctly predicted instances out of all predictions made. I chose accuracy because it provides a straightforward and intuitive understanding of the model's overall effectiveness across all classes.

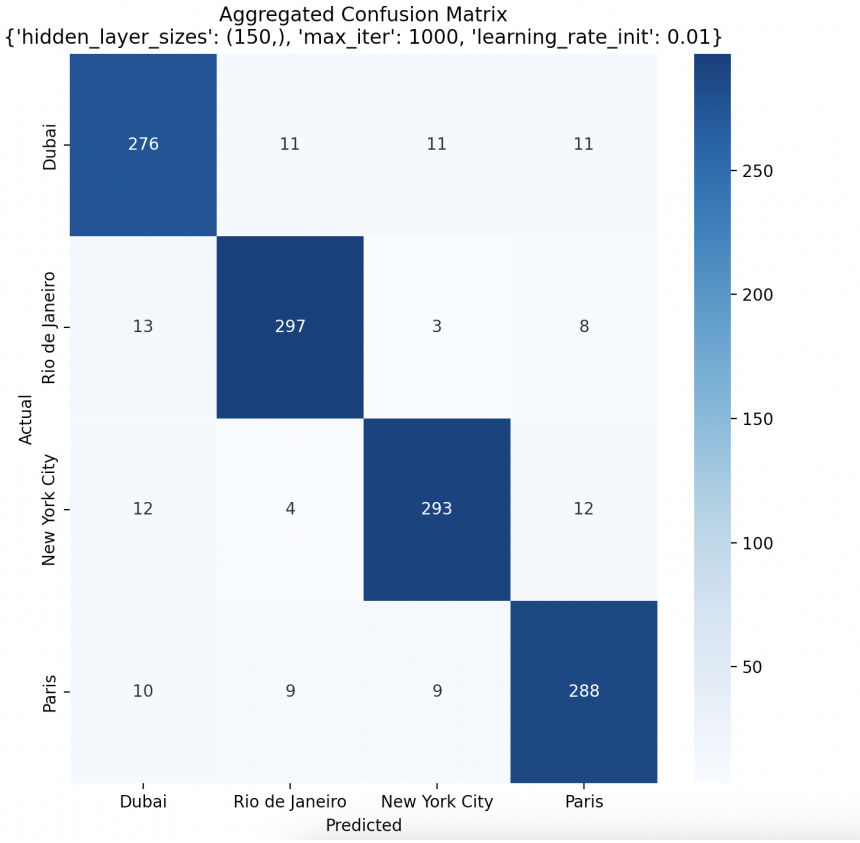
To ensure the robustness of the evaluation, I continue employing K-Fold cross-validation with 5 splits, which maximizes the use of my dataset for both training and validation while also ensuring that every data point contributes to the validation process exactly once. This method mitigates the risk of overfitting, as the model's performance is averaged over several train-test splits, providing a more reliable estimate of its ability to generalize to unseen data.

Additionally, to evaluate the model's detailed performance across the different classes, I analyzed the confusion matrix, which provides insight into the true positives, false positives, false negatives, and true negatives for each class. This allows me to observe not just the overall accuracy, but also the precision and recall for each class, giving a more nuanced view of the model's predictive capabilities. The classification report further complements this by offering a breakdown of precision, recall, and F1-score for each class, which is particularly useful when dealing with imbalanced classes.

Moreover, to ensure that my model is evaluated on completely unseen data, I reserved a set of 200 data points that were completely held out from both training and cross-validation processes. The performance on this final test set acts as the ultimate test of the model's generalizability and provides a final confirmation of its expected performance in real-world scenarios.

```
Iteration 332, loss = 0.01708949
Iteration 333, loss = 0.01703691
Iteration 334, loss = 0.01695130
Iteration 335, loss = 0.01687232
Iteration 336, loss = 0.01687313
Training loss did not improve more than tol=0.000100 for 10 consecutive epochs. Stopping.
Average K-Fold Accuracy: 0.9108
              precision    recall  f1-score   support

           0       0.89      0.89      0.89       309
           1       0.93      0.93      0.93       321
           2       0.93      0.91      0.92       321
           3       0.90      0.91      0.91       316

    accuracy                           0.91      1267
   macro avg       0.91      0.91      0.91      1267
weighted avg       0.91      0.91      0.91      1267
```





Training Output Analysis:

Loss: The training loss values are decreasing, which indicates that the model is learning and improving during training. The model stops after the training loss fails to improve significantly over 10 consecutive epochs, suggesting that it has converged to a stable solution.

Average K-Fold Accuracy: The reported average K-Fold accuracy is approximately 91.08%, a strong indication that the model is performing well across different subsets

of the dataset. This suggests good generalization.

Confusion Matrix Analysis:

Correct Predictions: The majority of the predictions lie on the diagonal, which represents true positives, with values such as 276 for 'Dubai' and 297 for 'Rio de Janeiro.' This confirms that the classifier is mostly accurate.

Remaining Misclassifications: There are some off-diagonal numbers, such as 'Dubai' being confused with 'Paris' 11 times, indicating areas where the model might be improved. However, these numbers are relatively low compared to the true positives, which is positive.

Class Balance: The confusion matrix shows a balanced number of predictions across different classes, meaning the model does not exhibit a significant bias towards any particular class.

Classification Report Analysis:

Precision: The precision is high for all classes (ranging from 0.89 to 0.93), which means that when the model predicts a class, it is usually correct.

Recall: The recall is also high for all classes (ranging from 0.89 to 0.93), showing that the model is good at identifying all true instances of each class.

F1-Score: The F1-Scores are consistently high (ranging from 0.89 to 0.93), suggesting a balanced precision and recall. This is important in applications where both false positives and false negatives are to be minimized.

Support: The support, which is the number of true instances in each class, is fairly balanced, indicating that the dataset is well-distributed among the classes, and the model's performance is not skewed by class imbalance. Macro and Weighted Averages:

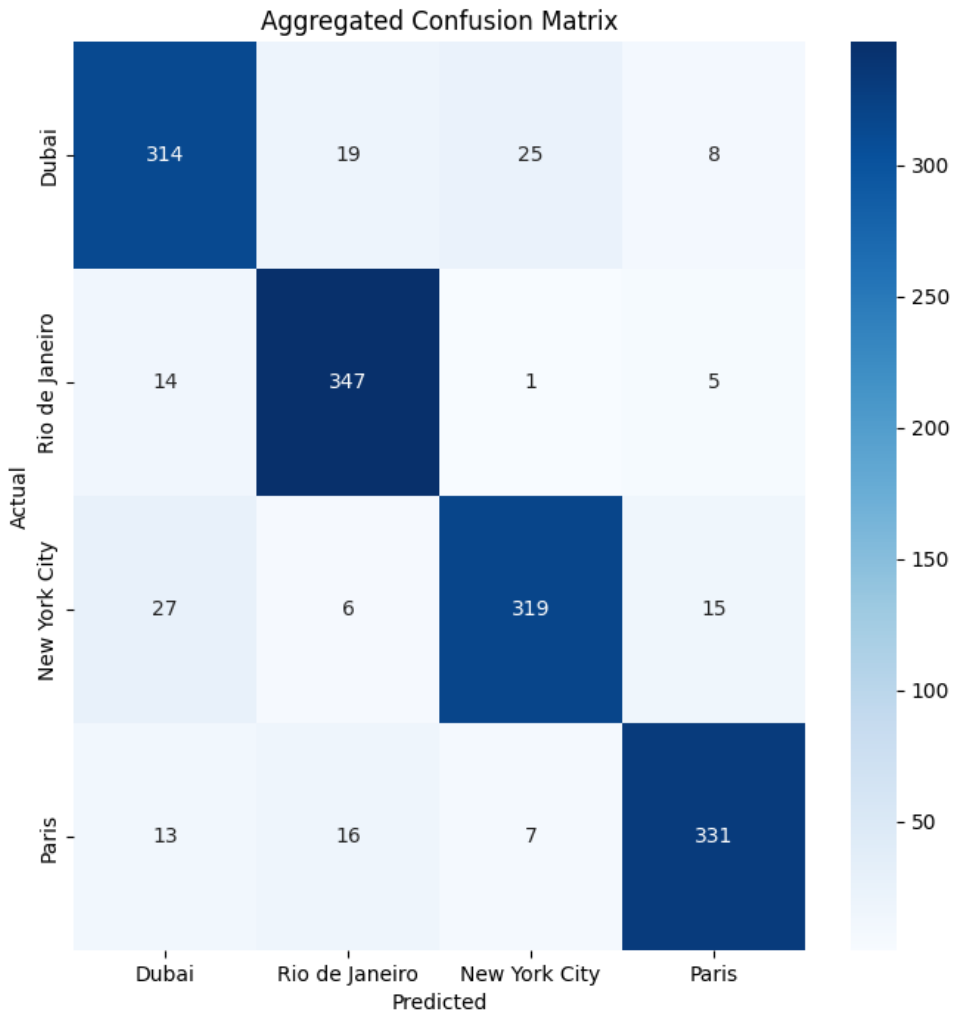
The macro average takes the average of the precision and recall for each class without considering support. The weighted average takes the class imbalance into account. Both these averages are at 0.91, which aligns with the high individual class scores, reinforcing the model's consistent performance across all classes.

The 200 data that is reserved before which completely held out from both training and cross-validation processes produced an accuracy of 0.9132. Which indicates a good performance of the model.

In summary, the model exhibits strong and balanced predictive capabilities across all classes, with no indication of overfitting. The good performance across all metrics, along with the high average K-Fold accuracy, suggests that the model's performance is strong.

3.3 Random Forest

3.3.1 Aggregated Confusion Matrix Analysis



The confusion matrix illustrates the number of correct and incorrect predictions made by the classifier with respect to the actual classes.

- **True Positives (TP)**: The diagonal elements of the matrix (314 for Dubai, 340 for Rio de Janeiro, 327 for New York City, 340 for Paris) indicate the number of instances that were correctly identified for each class.
- **False Positives (FP)**: The off-diagonal elements in the columns indicate instances that were incorrectly predicted as belonging to a class.
- **False Negatives (FN)**: The off-diagonal elements in the rows indicate instances of a particular class that were predicted as belonging to a different class.
- **True Negatives (TN)**: Although not directly shown, these are the instances that were correctly identified as not belonging to a particular class. This is represented

by the sum of all values not in the row and column for that class.

The high values on the diagonal indicate a high number of correct predictions. The model has performed similarly across all classes, which suggests that it is not biased toward any particular class.

3.4 Hyperparameter Tuning

In tuning the hyperparameters of the Random Forest, we adjusted the following parameters:

- **Number of Trees**: 200
- **max_depth**: 100
- **max_depth**: 256

we used grid search to tune hyperparameters. The below lists are possible values for each hyperparameter of the Random Forest Model In a Random Forest model, the hyperparameters are number of trees, max_depth of trees and $min_samples_split$. Then we will try all combinations of these hyperparameter choices with cross-validation.

```
num_trees = [50, 100, 150, 200, 250, 300]
max_depth = [1, 5, 10, 15, 20, 25, 30, 50, 100]
min_samples_split = [32, 64, 128, 256, 512, 1024]
```

Aggregated Classification Report:

	precision	recall	f1-score	support
0	0.85	0.86	0.86	366
1	0.89	0.95	0.92	367
2	0.91	0.87	0.89	367
3	0.92	0.90	0.91	367
accuracy			0.89	1467
macro avg	0.89	0.89	0.89	1467
weighted avg	0.89	0.89	0.89	1467

Average K-Fold Accuracy: 0.8937

3.5 Pred.py description:

Vocabulary and Network Parameters Initialization:

I begin by loading a pre-defined list of words, vocab, which I have selected as relevant features for the model based on prior explorations of the dataset. Following this, I load

the neural network's learned weights and biases from saved text files. These parameters define two layers of the network – the hidden layer and the output layer.

Text Processing and Feature Extraction:

My script includes a function insert_feature, designed to process raw text data. It takes each text entry, tokenizes it, removes punctuation, and checks against the vocab list to create a binary feature vector for each sample. If a word from the vocab is present in the text, the corresponding feature is marked as 1.0; otherwise, it's 0.

Data Preprocessing Function - process_data:

This function serves multiple purposes:

It reads the raw dataset from a CSV file, ensuring that I work with the latest data. I handle missing data in categorical questions (Q1 to Q4, Q6) using mode imputation or by assigning a default value where necessary. I convert all categories in questions Q1 to Q4 and Q6 into one-hot encoded vectors since these categorical inputs need to be in a format that's suitable for neural network processing. For other questions (Q5, Q7, Q8, Q9), I employ various preprocessing steps, such as random choice imputation for Q5 and normalization for the numerical questions to ensure they're on a similar scale.

Neural Network Forward Pass - predict_all:

Within this function, the actual computation for the neural network predictions happens:

The processed data is fed into the network, starting with the first layer where matrix multiplication is performed between the data and weights_first_layer, adding bias_first_layer, and then applying the ReLU activation function. The activated first layer outputs are then passed on to the second layer with another round of matrix multiplication with weights_sec_layer, adding bias_sec_layer. Using the softmax function, I convert the raw scores (logits) from the second layer into probabilities to interpret them as the likelihood of each class.

Making Predictions: Once I have the probabilities, I select the class with the highest probability as the model's prediction for each sample. This process is vectorized to efficiently handle all samples at once.

Model Evaluation: If I'm evaluating the model's performance, I compare its predictions against actual labels to calculate the accuracy. This gives me a clear indication of how well my model is performing. This is just for testing.

3.6 Final Prediction Model Description

Along with our `pred.py` file, there are four `.txt` files storing the weights and biases from our optimal Neural Network. We stored them in separate files due to their large size. We import them by using the `np.loadtxt` function to convert them into NumPy arrays. In our `pred.py` file, we first process the input data into the same format we discussed earlier. Then, by replicating the similar calculation methods from the lecture and the lab using Numpy library functions, we can get a list of prediction result (e.g 'Dubai', 'Rio de Janeiro', 'New York City', 'Paris').

4 Prediction

Prediction: we expect the model to perform with an accuracy of approximately 90% on the test set.

Reasoning: The model achieved an average K-Fold cross-validation accuracy of 91.08% during training, indicating a consistent performance across various subsets of the training data. This consistency suggests the model has generalized well beyond the training examples and can handle unseen data effectively.

Furthermore, the precision, recall, and F1-scores across the different classes are all above 0.89, with very similar values for macro and weighted averages, demonstrating balanced performance that is not overly influenced by any particular class. This balanced performance, combined with the fact that the training loss stabilized (indicating convergence without overfitting), gives empirical evidence to support the accuracy prediction for the test set.

The chosen hyperparameters, including the size of the hidden layer and learning rate, have been empirically validated to yield a model that captures the complexity of the data without capturing noise - further reinforcing the expectation of similar performance on unseen data.

However, the prediction could be slightly conservative due to potential over-optimism in the cross-validation process. So a slight reduction might be more realistic. Considering this, and to provide a point estimate, I would predict the model's real-life accuracy to be around 90%. This accounts for potential variations and uncertainties when the model encounters truly unseen data, which might not perfectly follow the same statistical distribution as the training set.

5 Workload Distribution

Kaifeng Li took care of Process Data and Data Exploration, accomplish the BOW algroithm and assist KNN cross validation algorithm 25%
Kangzhi Gao was responsible for processing data, Neural Network model training and pred.py generation. 25%
Keke Chen: Neural Network model, Random Forest model, Neural Network model choice report, final prediction. 25%
Xiaoyi Jia took care of KNN model, Data Exploration 25%