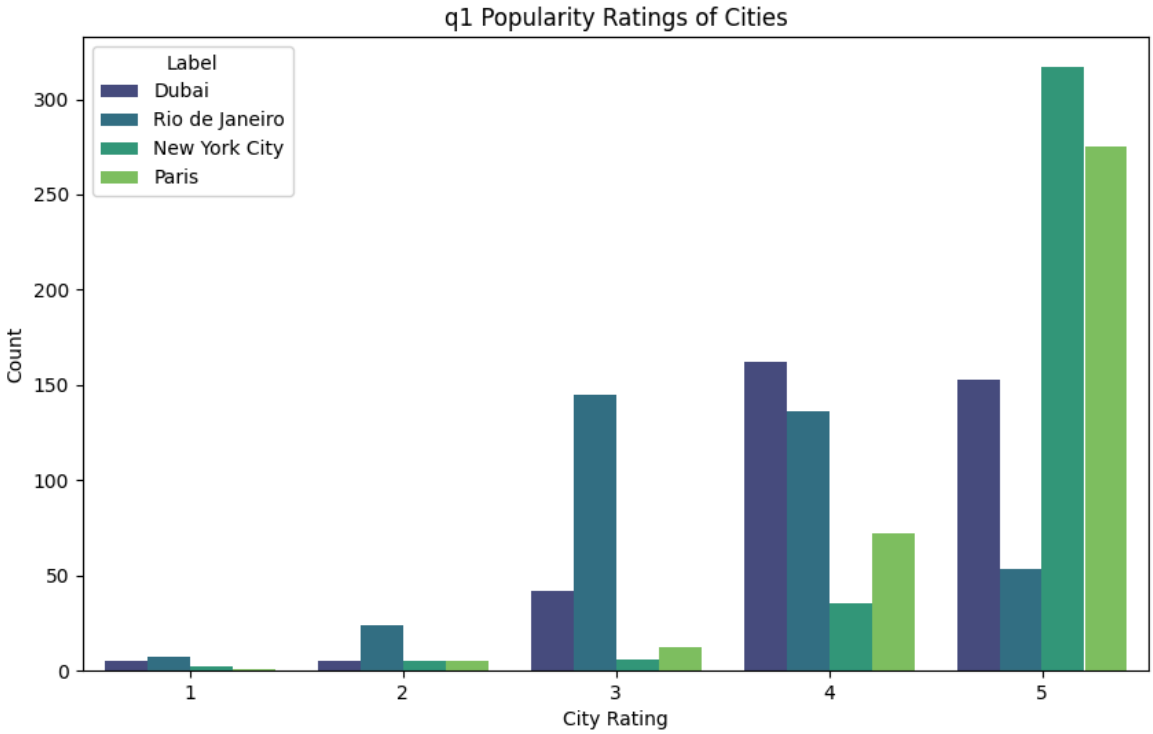
Machine Learning Challenge

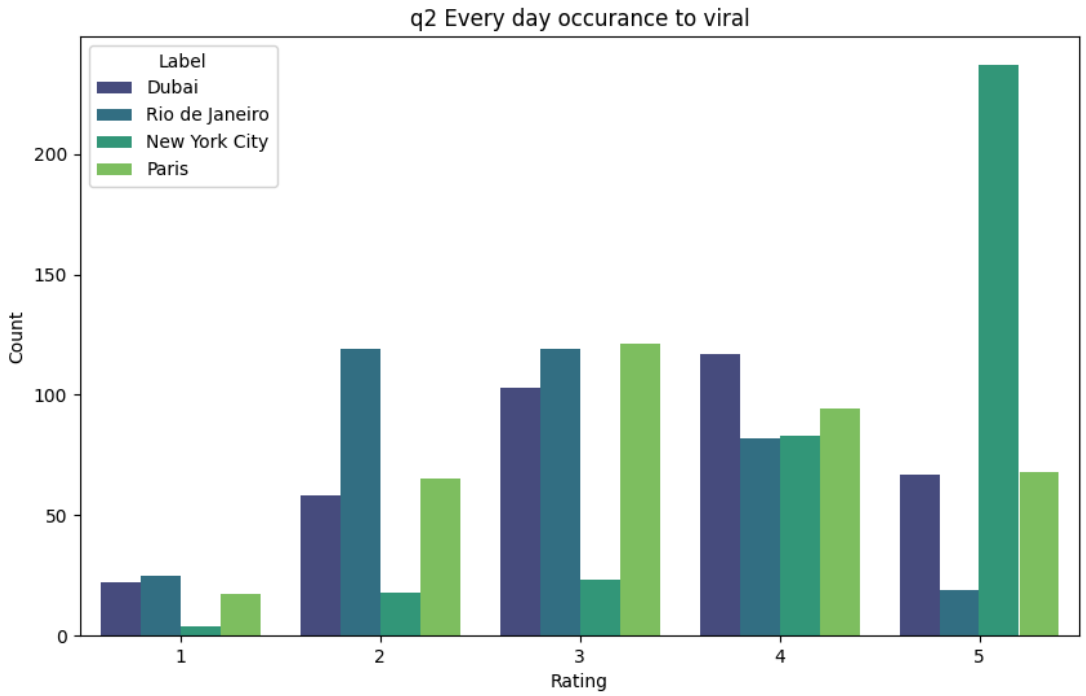
Data Exploration:

We started exploring data by first attempting to visualize the features that could be put in a graph and creating a frequency chart for the rest. The code for this is in data_visualization.py. We generated bar graphs for questions 1-5, box plots for question 7-9 and a frequency sorting for questions 6 and 10. Generating these allowed us to learn more about our data. First we noticed how the data is distributed in each question,

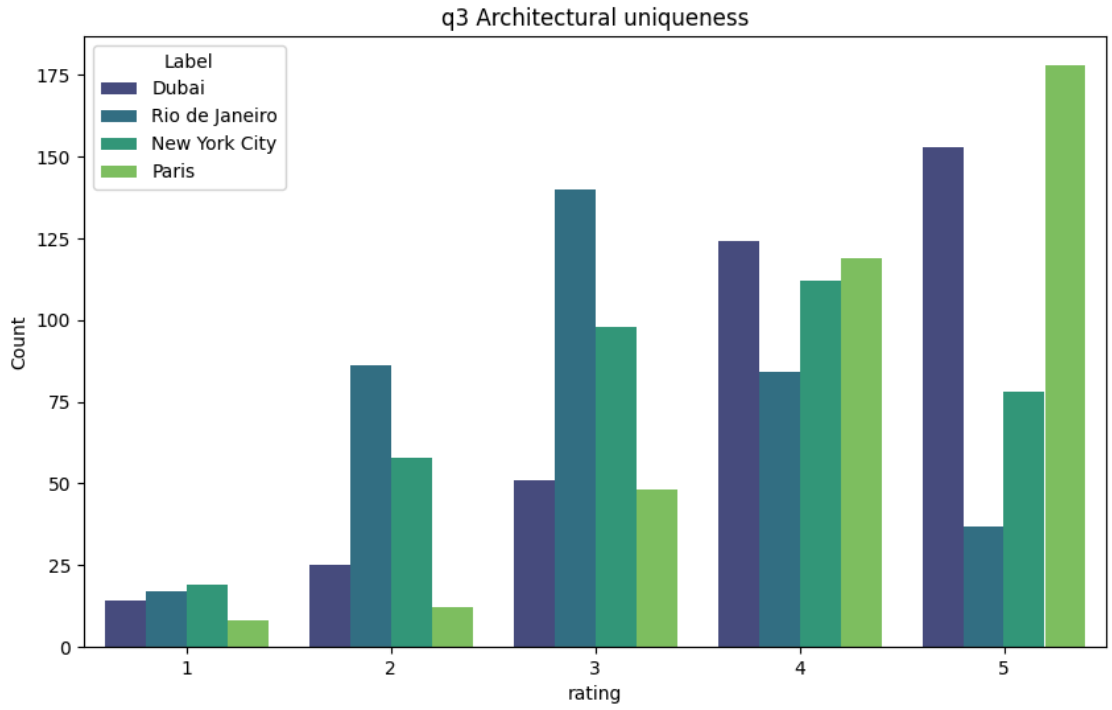
- Q1. The answers seemed to show that every one of these cities are popular as the majority of the answers were in the 4-5 range and New York City was the most common 5 rating answer with 300+ answers.



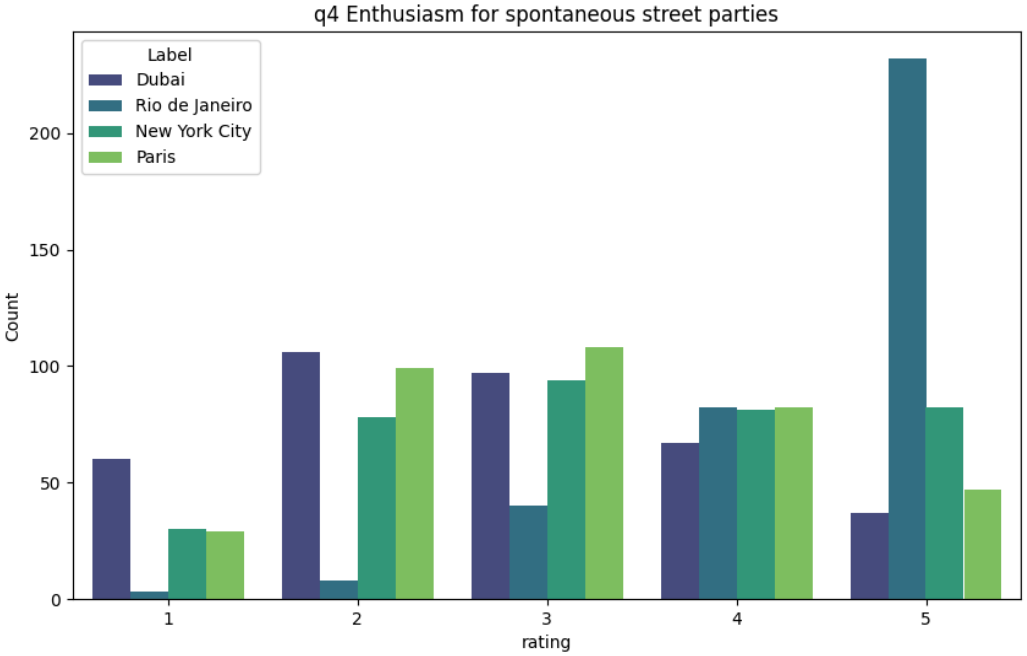
- Q2. New York City was the most common 5 rating answer (250+) where even the sum of the 5 ratings for the other cities would not add up to it. It seemed like the majority agreed New York City is the answer as most answers give it a 5 or 4 while other cities were distributed across 2-5.



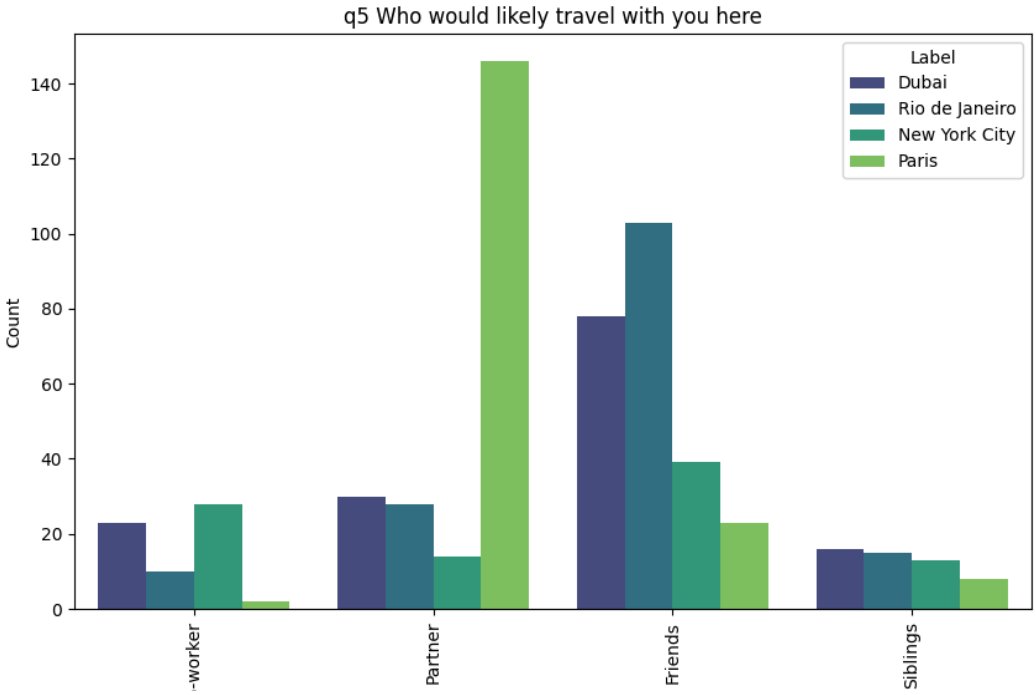
- Q3. In this question the data was distributed all over the ratings 2-5 where there was no clear answer that all the people agreed on for any city. Paris seemed to have the most 5 ratings but it was followed closely by Dubai and Dubai had more 4 ratings.



- Q4. This question had a clear answer that most people agreed on. Rio had the most (230+) 5 ratings while other cities were distributed fairly similarly across 2-5 (50-100) answers per.

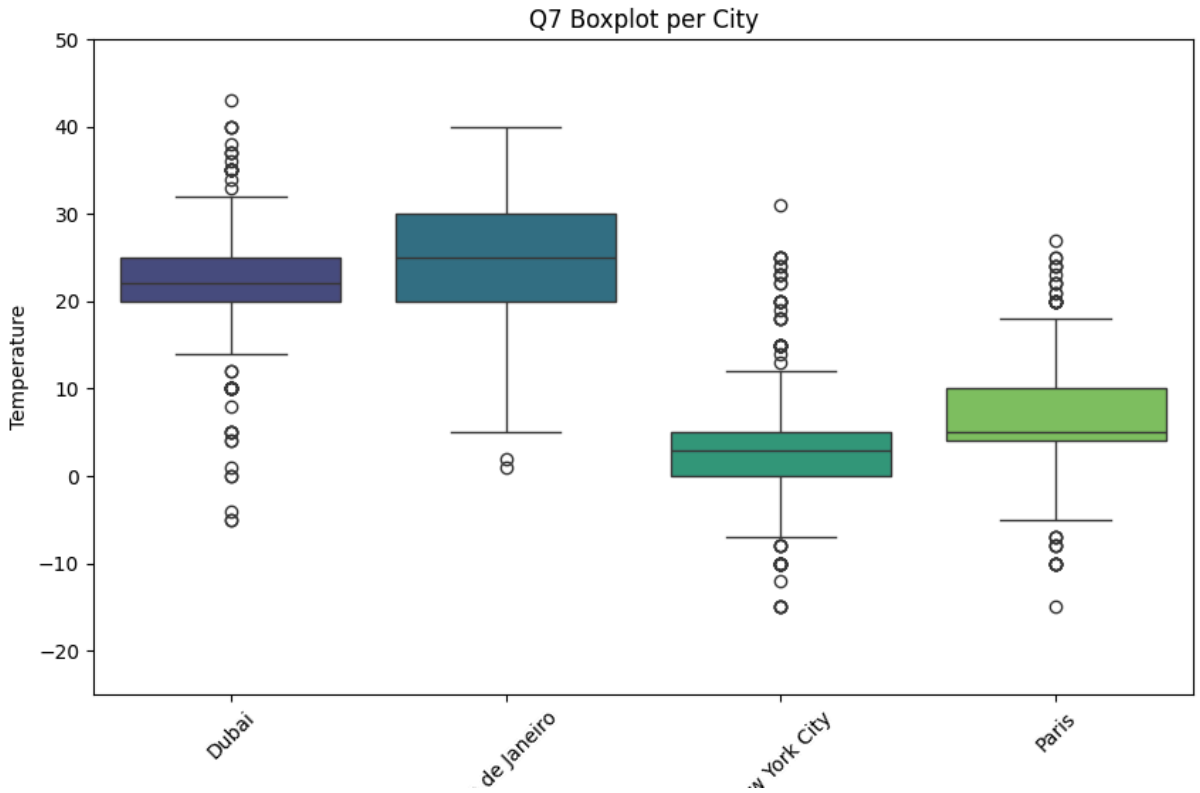


- Q5. This question provided some interesting observations like Paris and partner was the most popular answer 140+ answers. New York and a co-worker was another one with 100+ answers. Similarly for Rio and friends.

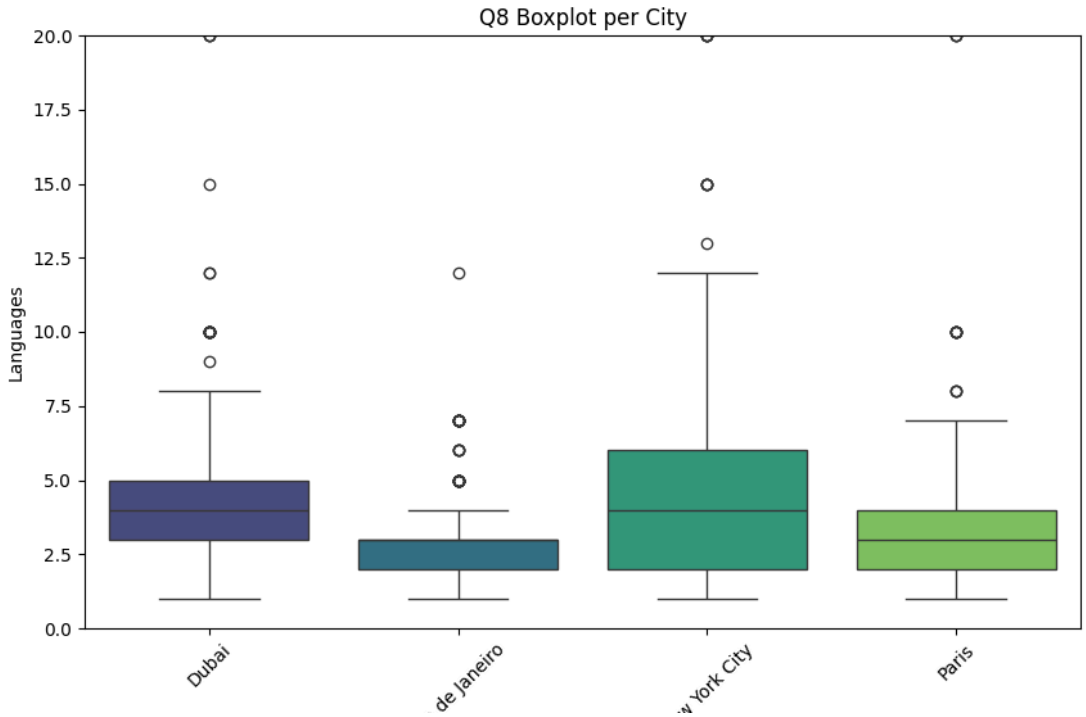


- Q6. The answers here varied a lot and there were a lot of answers that only happened once. Though still we noticed that most answers for dubai had the skyscraper rating at 6, carnival at 1 for New York, cuisine and music 5-6 for Paris and carnival and sports at 5-6 for Rio.

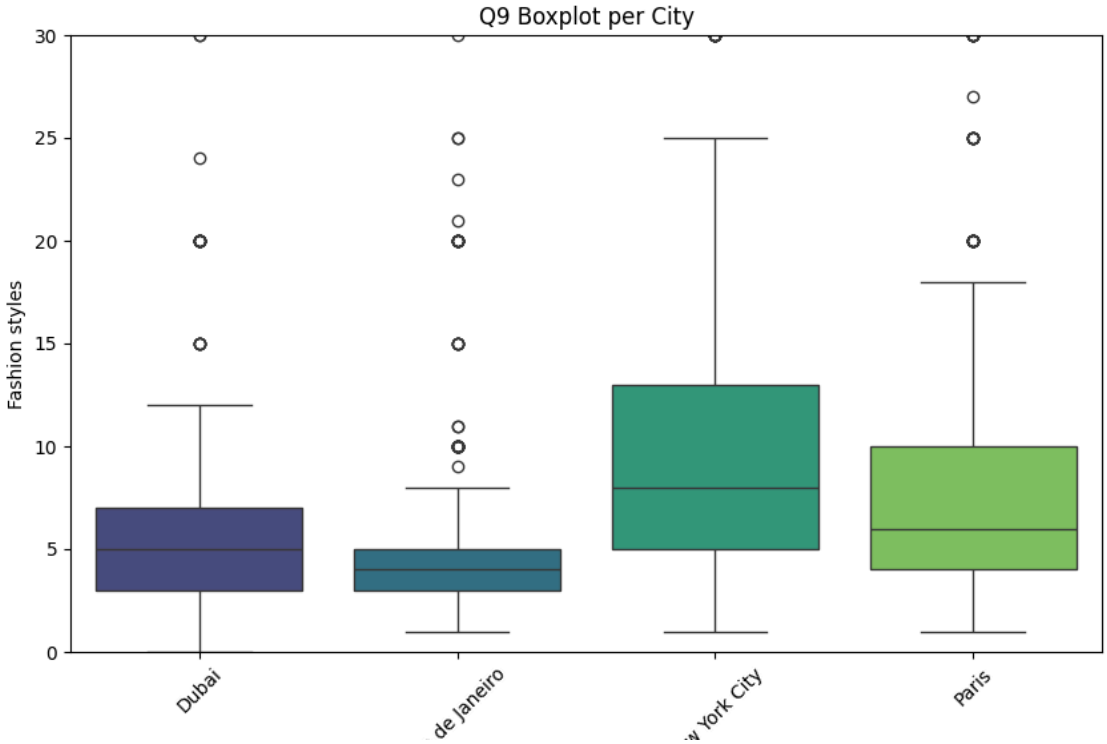
- Q7. The boxplots showed the concentration of answers for each city and they were in different ranges Dubai 20-25, Rio 20-30, New York 0-5 and Paris 5-10. There were also many outliers in both directions of this range and we also noticed a very unrealistic answer of 1000 for one of the cities.



- Q8. The answer to this question was also interesting. We noticed that most people agree that few languages are heard daily in Rio 2-3 with little outliers and more in New York 2-6. Dubai was around 3-5 and Paris 2-4. Though there were also a lot of outliers especially for New York and also sort of unrealistic answers like 20+.



- Q9. Rio had the smallest range in this question with most answers putting it between 2-5 while New York was between 5-15. Dubai was 4-6 and Paris 5-8. Though again this question had a lot of outliers putting the answer at 20+ which again is unrealistic.



- Q10. Similar to question 6 there were a lot of answers that only happened once. But we noticed many answers associated money and riches with Dubai, love with paris, football with Rio and some known quotes for New York

Determining Input Features:

The input features are determined based on the columns present in the dataset (‘Q1’ to ‘Q10’) along with some processing steps to handle categorical data, missing values and text-based data. This is done in the *clean_data()* method.

- ➔ Dropping irrelevant columns: We dropped the ‘id’ column because it was an identifier for who wrote the review and not relevant to the analysis.
- ➔ Handling categorical data:
  - ◆ The 'Q5' column was split into individual categories ('Partner', 'Friend', 'Siblings', and 'Co-worker'), and each category was encoded as a binary variable to indicate whether it was clicked or not in the review. This approach was informed by the bar graph for Q5. It suggested that the presence of these categories being clicked in the review could be a strong indication of which city the review pertained to (‘Partner' for Paris or 'Co-worker' for New York). Hence, we believed it would be appropriate to have them as individual features. The original ‘Q5’ column was dropped.
  - ◆ The ‘Q6’ column was split into individual categories ('Skyscrapers', 'Sport', 'Art and Music', 'Carnival', 'Cuisine', 'Economic') and converted into an integer rank. We noticed during data exploration that certain answers strongly correlate to certain cities such as ‘Skyscrapers’ for ‘Dubai’. Hence, we believed it would be appropriate to have them as individual features. The original ‘Q6’ was dropped.
- ➔ Handling missing values:
  - ◆ The missing values in 'Q1', 'Q2', 'Q3', 'Q4', 'Skyscrapers', 'Sport', 'Art and Music', 'Carnival', 'Cuisine', 'Economic' are filled with the mean of each column. This is a common way to handle missing data.
- ➔ Standardization:
  - ◆ Since columns ‘Q7’, ‘Q8’ and ‘Q9’ do not have a specified scale. To prevent it from dominating the model, these features were standardized with a mean of 0 and standard deviation of 1.
- ➔ Text processing:
  - ◆ Since ‘Q10’ included “text-based data”. We decided to first sanitize these answers by removing any sort of special characters, punctuation and converting it to lowercase. Then, a bag of word representation was created where each row represents a sample and each row represents a word with values indicating whether the word is present in the sample or not. The matrix is concatenated with the data, so each unique word is turned into a feature. The original ‘Q10’ is dropped.

All the features are handled appropriately based on either empirical evidence found during data exploration or based on standard best practices in data preprocessing. The valuable information is retained and no information is overlooked.

Representing Data:
Our goal with representing data was to convert all the questions into a format that could be used in our models. This usually meant converting them to something numerical. For questions 1-4 this just meant converting the numbers into integers since they were only numbers in the range 1-5. Questions 5 and 6 had multiple sub parts hence we used one-hot encoding to break these questions down. So for question 5 if any of the answers were checked then that column would have a 1 else 0. For question 6 every sub part was its own column with an integer. For questions 7-9 we could not just convert to integer because the difference in values from these questions compared to questions 1-4,6 was huge and we know this could have a disproportionate impact on generating weights. So we normalized these weights first using their standard deviation and mean. Also we consider any value more than 2 standard deviations as outliers and replace them with mean. Lastly for question 10 which are the quotes we first do some sanitizing by removing special characters and lower casing all of them then using similar techniques from lab 9 we convert it to a bag of words. The columns generated by the bag of words are appended to the dataset replacing Q10.

Data Splitting:

We shuffle the data to randomize the order of the samples using a random seed of 42. The data samples were grouped by city which is not ideal. Shuffling prevents overfitting and ensures that the model is exposed to diverse sets of data during training. By using the same random seed across all models, the resulting datasets are consistent across different runs. The function splits the shuffled dataset into three parts based on the specified proportions:

- The first 20% of the shuffled data becomes the test data which is used for evaluating the model's performance after training.
- The next 20% becomes the validation data used for hyperparameter tuning and model selection.
- The remaining 60% is used for fitting the model.

We tried splitting data into different batch sizes like 10% for testing, but found that this had the best accuracy. Additionally, allocating 60% of the data for training is an ample amount of data for the model to learn from.

Model Exploration:

1. **Multilayer perceptron:**
   We first chose to configure the MLP model due to its learning capabilities, scalability, flexibility, and the availability of libraries for it. We chose to use the sklearn.neural_network.MLPClassifier model and trained it with specific hyperparameters such

as the number of hidden layers and neurons per layer (hidden_layer_sizes), regularization strength (alpha), and maximum number of iterations for optimization (max_iter) to get the highest test accuracy possible. Ultimately, we could not get higher than a 0.88 test accuracy after experimenting with various hyperparameter options.

2. **Logistic Regression:**
   After experimenting with the MLP model, we decided to explore the logistic regression model since it is effective for classification problems, making it a natural choice. We chose to use the sklearn.linear_model.LogisticRegression model, achieving an accuracy of 0.93. Inspired by this high accuracy, we decided to implement logistic regression ourselves, with the insights gained from the sklearn implementation.
   To proceed with our implementation, since logistic regression requires numeric features, we first encoded the labels into numerical values. We then experimented to optimize the features and hyperparameters of the logistic regression model. This involved tuning parameters such as the number of epochs (iterations) and learning rate. Following this, we identified the optimal learning rate as 0.01 and the best epoch count as 2000.
   Moreover, during feature analysis, we observed that features Q7, Q8, and Q9 had minimal impact on accuracy.Conversely, we found that the inclusion of the bag of words feature Q10 improved accuracy by 4%.
   Despite our efforts in hyperparameter tuning and feature selection, our own logistic regression model could only achieve an accuracy of only 0.89. So, we decided to utilize the weights and biases calculated by the Sklearn model and combine it with our own model for the final model.

3. **KNN**
   Given that there are multiple features and criteria that a sample can be judged based on, we decided to explore the K-Nearest Neighbors (KNN) model because it is a simple yet effective algorithm for classification tasks.

   We applied the KNN algorithm using the **KNeighborsClassifier** from the scikit-learn library. We experimented with various hyperparameters to find the optimal configuration for our dataset. Specifically, we varied the number of neighbors (3, 5, 7, 9), the weighting method (uniform and distance), and the distance metric (Euclidean and Manhattan). To calculate the MSE, we had to encode the labels into numerical values (similar to the Logistic Regression model).

   After exploring different combinations of hyperparameters, we found the best-performing KNN model to be the one with 9 neighbors, using the Manhattan distance metric and uniform weighting. It had the following results:

   Training Accuracy: 0.8877551020408163
   Validation Accuracy: 0.8600682593856656

Test Accuracy: 0.8532423208191127
Precision micro: 0.8532423208191127
Confusion Matrix:
[[63 7 1 3]
 [4 57 8 0]
 [6 9 63 0]
 [3 0 2 67]]
Mean Squared Error (MSE): 0.4709897610921502

This was a considerable improvement on our initial KNN model without any hypertuning which had a test accuracy of 0.812.

We were satisfied with the results of the KNN model, but chose not to proceed with it due to the high MSE value compared to the LR and MLP models. We believe that the other models are more well-suited to performing on large datasets.

Naive Bayes

After exploring MLP and logistic regression with varying degrees of success, we decided to explore Naive Bayes to answer the following question: Could a simpler model, under certain conditions, perform better than its complex counterparts? To do this, we opted for Gaussian Naive Bayes due to the nature of the dataset which was mostly numerical. Our initial implementation of a GNB model using only numpy and pandas, yielded a training accuracy of 0.2496 and a validation accuracy of 0.2517. After fixing divide-by-zero issues handling features with little to no variance among the samples within a class, the validation accuracy was raised to 0.5748. Further refinement involved altering the PDF and predict functions to compute the logarithm of probabilities directly, leveraging the fact that log(a/b) = log(a) - log(b). This was by far the most effective change, boosting the validation accuracy to 0.7007. After hitting a wall in terms of the manual model we transitioned over to sklearn's GNB model. Using their GridSearchCV tool to automatically search for an ideal variance floor, we ended up with a final model accuracy of 0.8844. While aiming for model simplicity and efficiency, we faced the problem of feature selection. We decided to remove features that contributed to model complexity without significantly affecting the accuracy, such as Q6 and Q10. Although the accuracies are lower in comparison to Logistic regression, the process allowed us to investigate the underlying structure of our dataset and rule out the existence of simple patterns that could be captured by a more straightforward model.

Model Choice and Hyperparameters:

1. **Evaluation Metric:**
   To ensure that the models are comparable we utilized the same data cleaning and splitting for all the models. This way the format of the data given to each model was the same. The training, validation and test sets generated for each model was generated using the same random seed split in the clean_data.py file.
   To compare the models we relied on testing and validation accuracy and MSE. Where we compared the rates for all the models in our model exploration part. Before we did this comparison we tuned the parameters of all the explored models to ensure that we were getting the best possible result with that model given the data. We discovered the following:
   1. Logistic regression got a test accuracy of 0.9351535836177475 and MSE 0.3743156983572648
   2. MLP got a test accuracy 0.8808510638297873 and MSE 0.39404255319148934
   3. KNN got a test accuracy of 0.8532423208191127 and MSE 0.4709897610921502
   4. Naive Bayes got a test accuracy of 0.8844 and MSE 0.3912

   Comparing all these we decided that logistic regression had a significantly higher rate and also a good MSE which is why we decided to go with logistic regression.
2. **Hyperparameters:**
   After experimenting with different hyperparameters for the models, we found that the ones that gave us the highest accuracy and least MSE were:
   1. Logistic regression: {C=1.0, penalty=l2, solver=newton-cg, max_iter=1000}
   2. MLP: {hidden_layer_sizes=(300, 200, 100), alpha=5, max_iter=1000}
   3. KNN: {n-neighbors=9, metric=manhattan, weights=uniform}
   4. Naive Bayes {'var_smoothing': 0.0008111308307896872}
3. **Final Model:**
   With the information collected above, we opted for Logistic Regression as our model. We computed the weights and biases for each label (New York City, Dubai, Rio de Janeiro, and Paris) using sklearn's LogisticRegression model with the aforementioned hyperparameters. Leveraging these learned parameters, we can determine the probability of an input belonging to any of the four categories. Subsequently, we assign the input to the category with the highest probability.

Prediction:

We are using all the available data to make predictions and the parameters are fine tuned after experimenting repeatedly with different values in our logistic regression model which is what pred.py is implementing. The model is tested on different shuffles of test and training set as well as on different sizes of the split. The result of all these testing is an accuracy of 0.93 with a low MSE. We believe due to how this was achieved that our model is not overfitting and clearly it's not underfitting. Given all of this we believe there is no reason for our model to underperform on the test set so we predict it will have an accuracy of 0.93.