Machine Learning Challenge Report

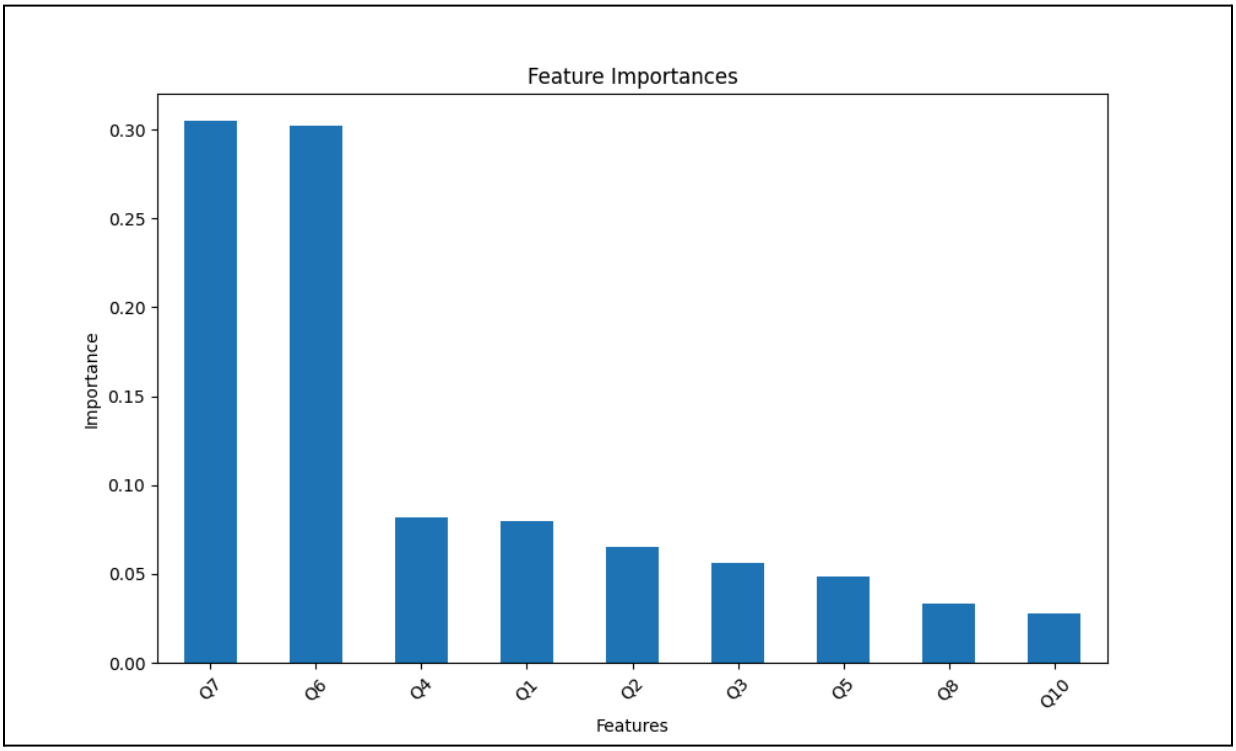
Data

We started by reviewing clean_dataset and analyzing the usefulness of each feature. The spread of the data was disappointing because we found that some features, such as average temperature, architectural uniqueness, and popularity, were not as helpful for making predictions as we had hoped. This was mostly due to unexpected responses in the survey. We decided that we would need to perform feature engineering on most of the features to provide meaningful data to the model.

Distributions for each feature were graphed, they are displayed below. Using sklearn's *feature_importances_* function, we saw that Question 7, which is Average January Temperature, and the categories from Question 6 were the most useful features (Figure 0).

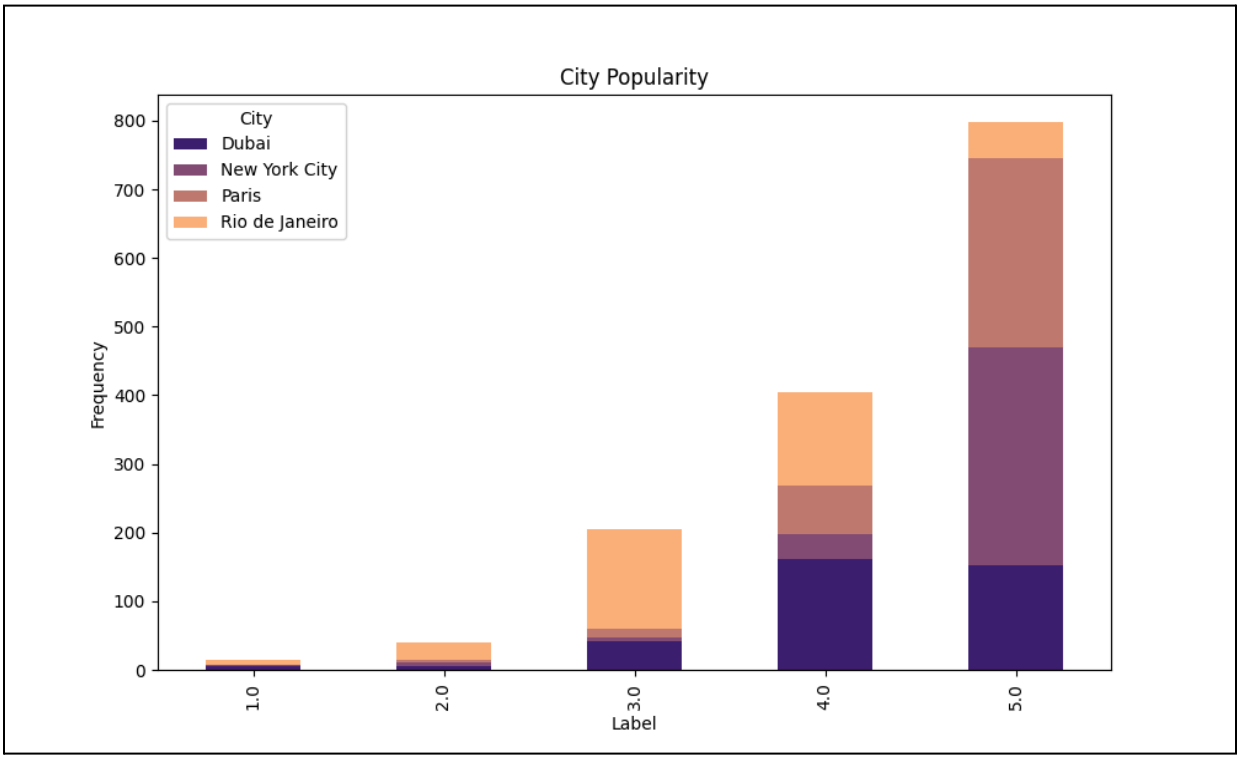
The Average January Temperature (Figure 5) graph shows a large difference between the four cities. Participants consistently chose Rio and Dubai to be warmer and New York City and Paris to be colder, this clustering made this feature more useful than others when training our model. For example, something like Architectural Uniqueness (Figure 3) which has less drastic variations in answers across cities does not help as much as Average Temperature when making a prediction. It is a similar case for the other features, they do not have enough variability to consistently make accurate predictions.

Figure 0: Feature Importance



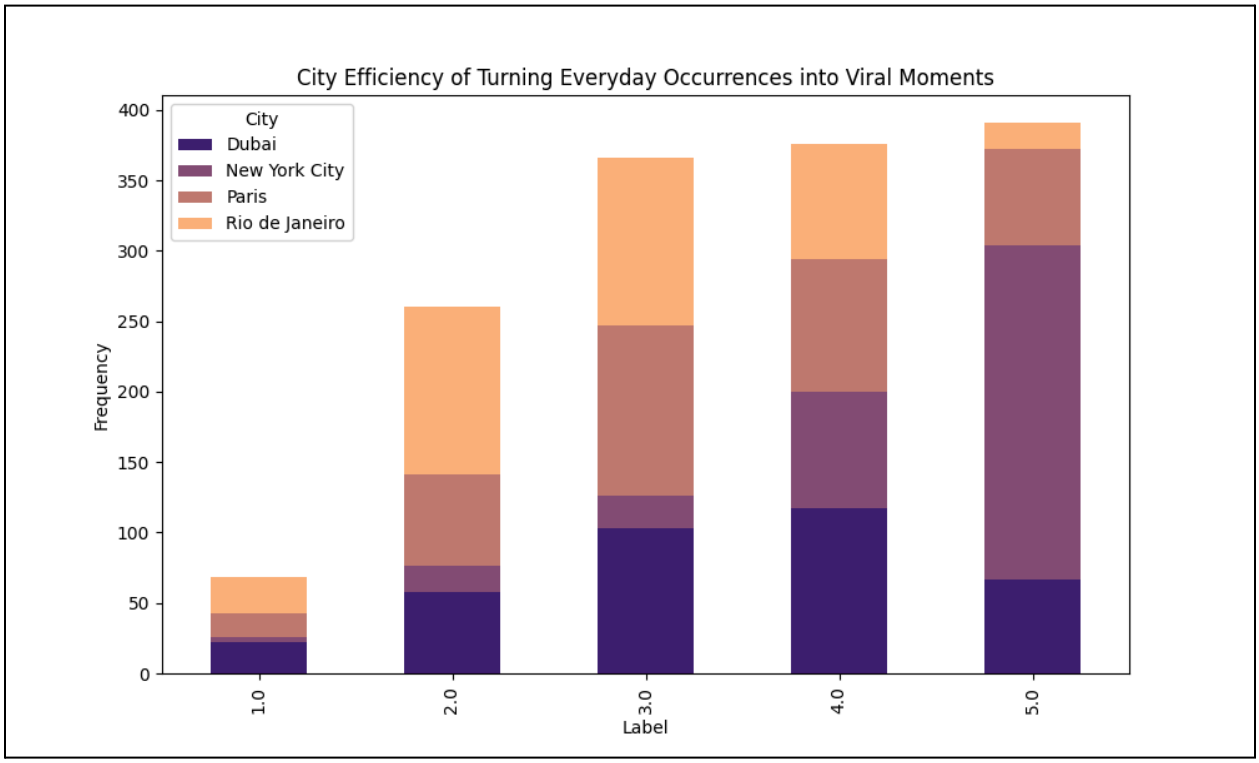
We concluded after preliminary testing for which model to use, that features such as Q6 and Q7 were far more important than we had considered, as is reflected in the graph above.

Figure 1: City Popularity (Q1)



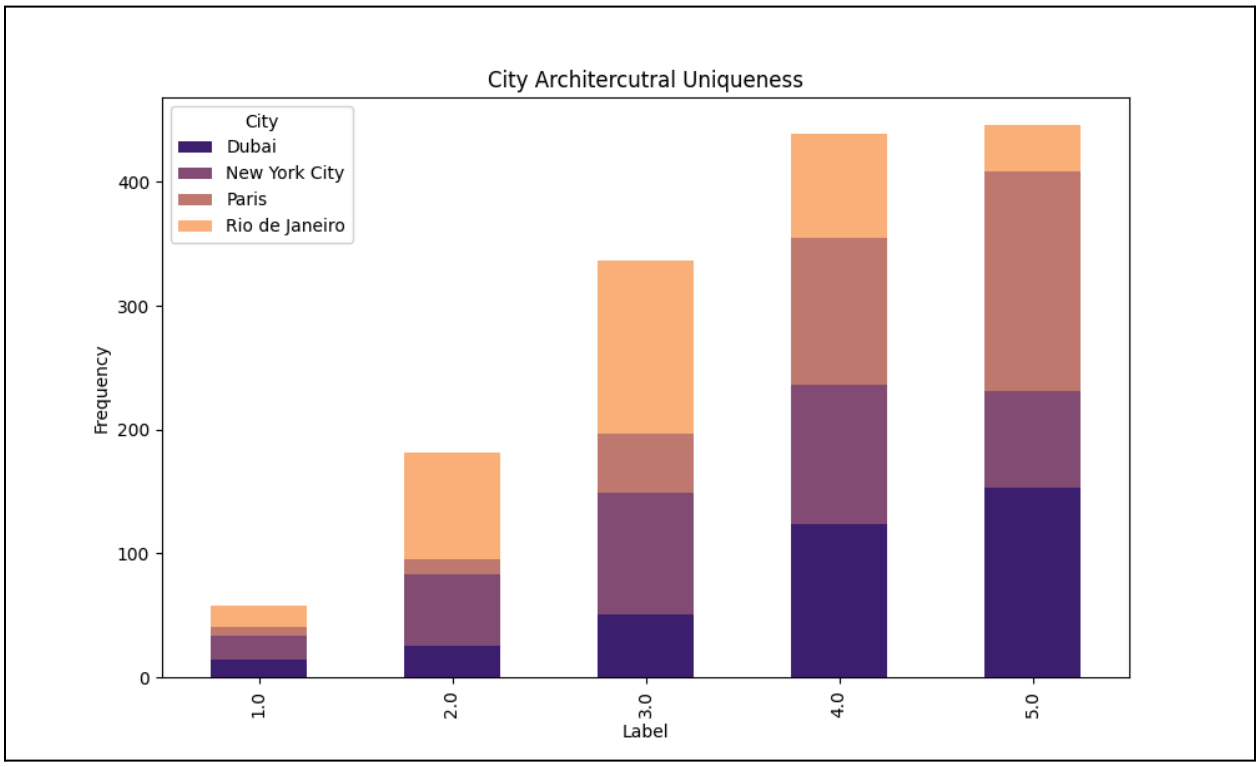
We found the city's popularity was most associated with New York and Paris. As the popularity score dropped, the more likely the city to be named was Rio. This feature is therefore important as a lower popularity score indicates a label of Rio.

Figure 2: City Efficiency in Turning Everyday Occurrences into Viral Moments (Q2)



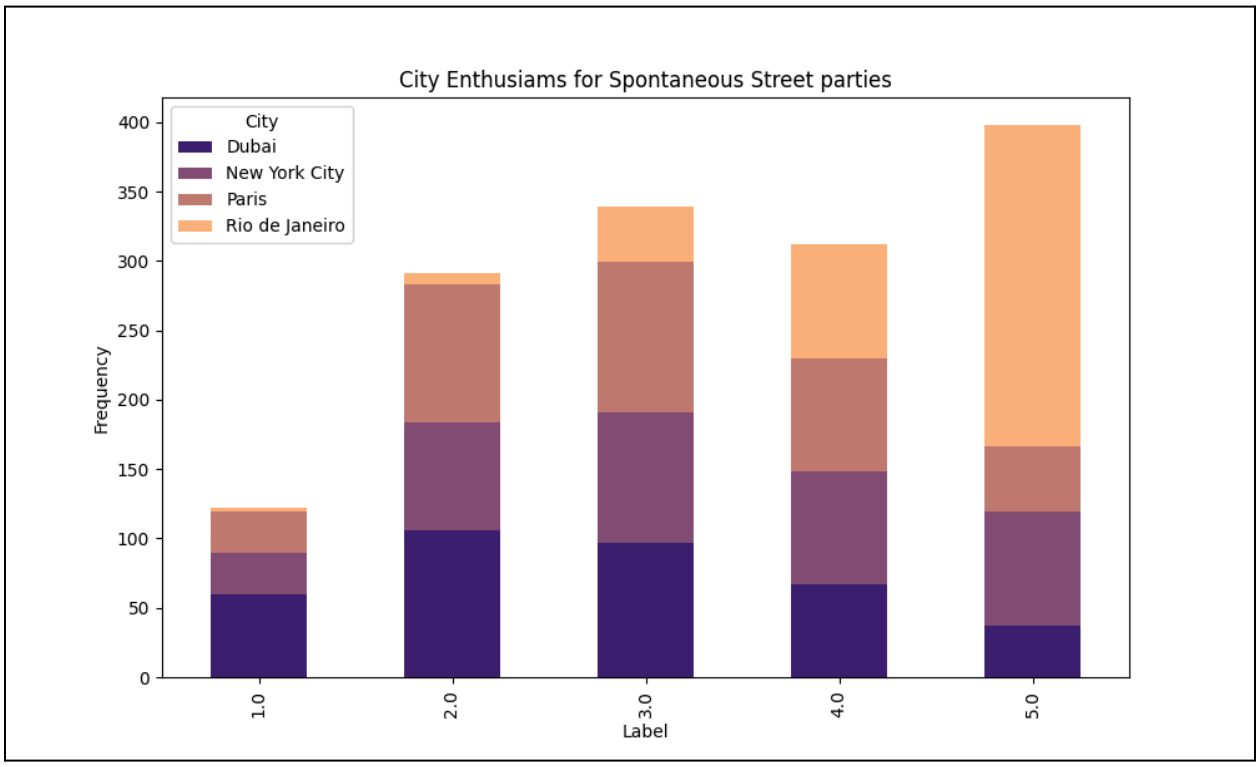
Viral everyday moments were more massively associated with New York City, followed by Paris, with Rio and Dubai consistently being rated lower. New York City and Paris having the highest frequencies of 5.0 ratings can help the model learn their labels, which highlights the importance of this feature.

Figure 3: Architectural Uniqueness (Q3)



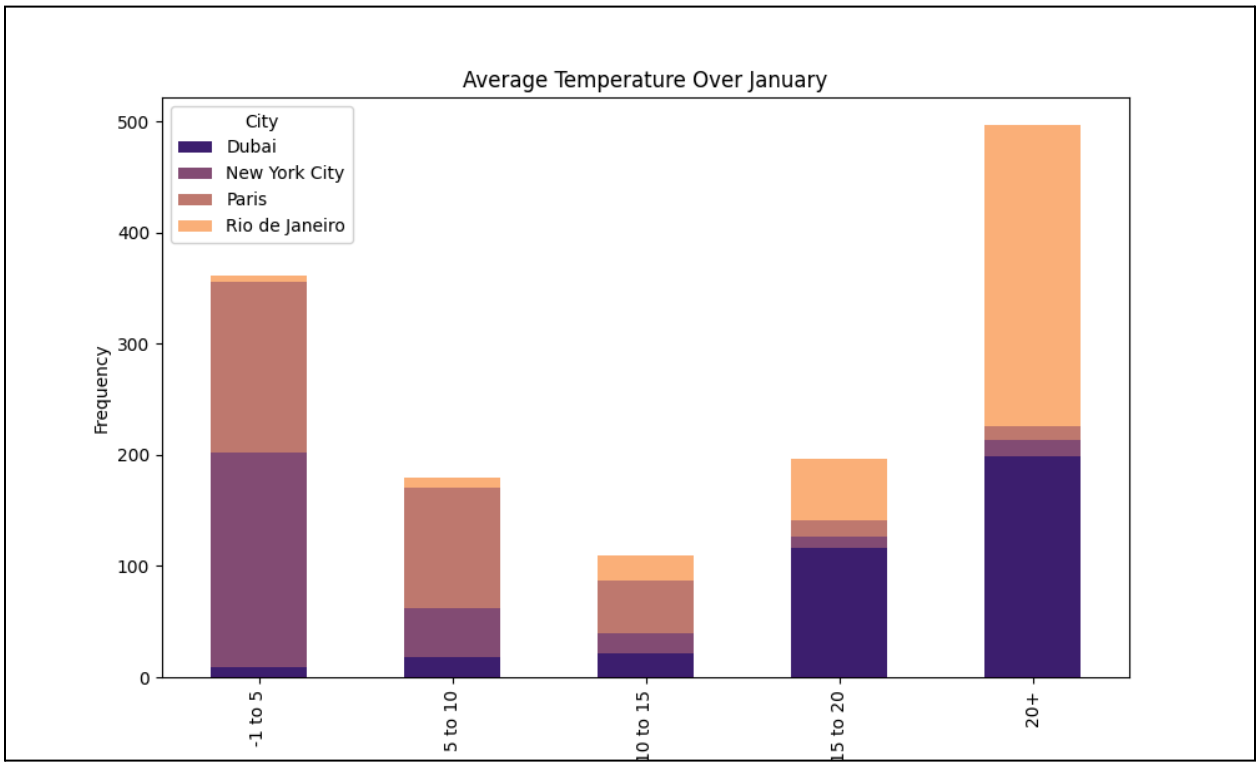
A higher rating of this feature was strongly correlated with Paris and Dubai, cities known for their iconic architectural wonders. It is also to be noted that the more average the rating, the more likely the cities to be named were New York and Rio.

Figure 4: City Enthusiasm for Spontaneous Street Parties (Q4)



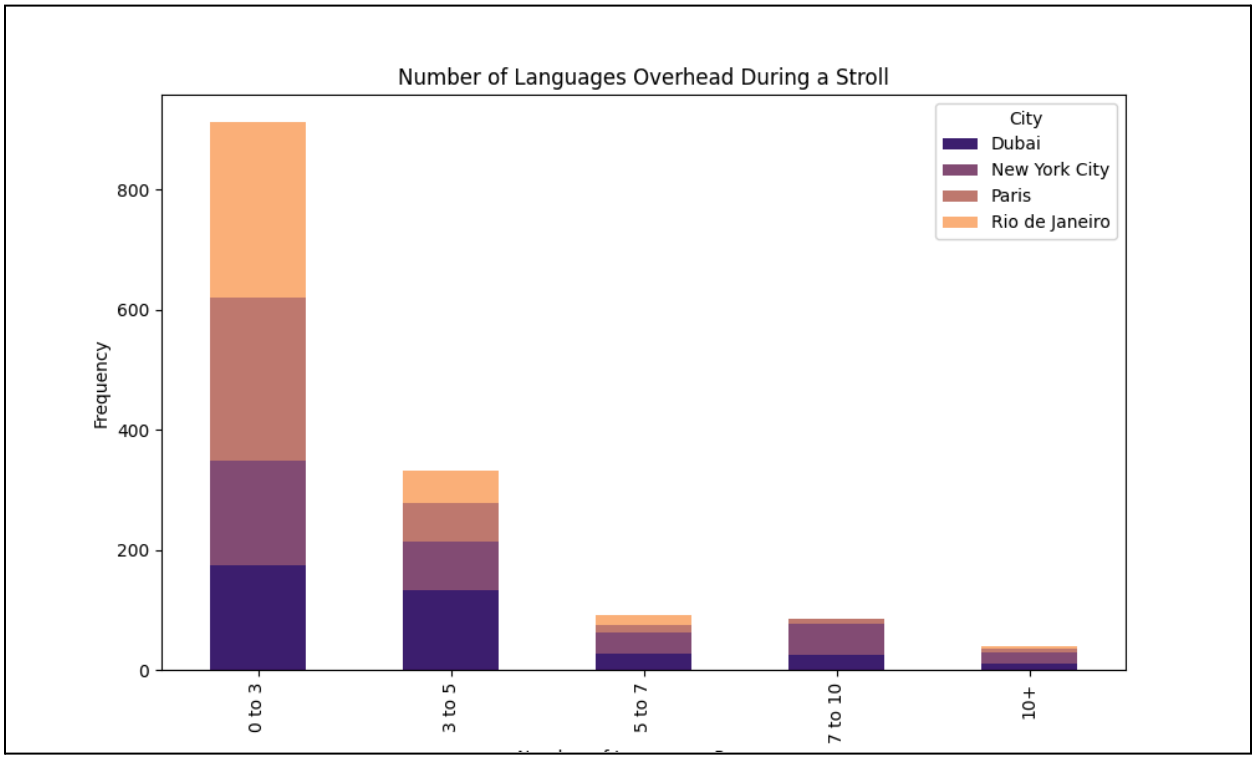
Among responses, Rio has the highest enthusiasm for spontaneous street parties. The other cities are roughly the same, with New York being slightly ahead of Paris and Dubai. This means that this feature would not always be useful when making predictions, apart from identifying Rio as an apparent outlier.

Figure 5: Average January Temperature (Q7)



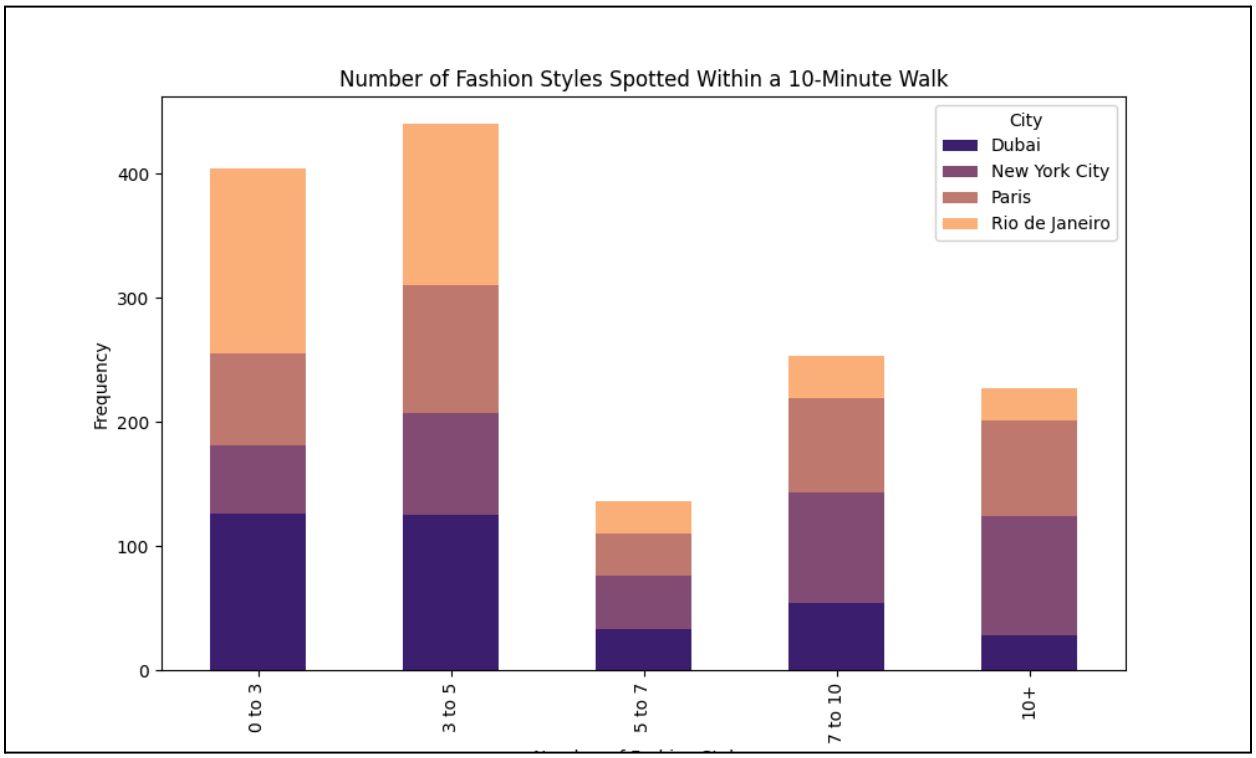
This feature was the most useful and easily identifiable marker of a city. As Cities like Rio and Dubai are located in warmer climates, their average January temperature will be significantly higher. On the other hand, cities such as New York and Paris are located in colder climates so their average temperatures will be lower. It is important to note that this feature is not based on subjective factors such as popularity or interest, as some of the others are, but is instead a material observation.

Figure 6: Number of Languages Overheard During a Stroll (Q8)



This feature proved to not be as useful, as there are no strong correlations that can be drawn from the data. The survey results found that each of the cities was almost equally likely to have a certain number of languages spoken.

Figure 7: Number of Fashion Styles Seen in 10-Minute Walk (Q9)



We iteratively searched for the most important features when doing preliminary analysis of the data and found that the models that outputted the highest prediction accuracy on the test data had Q9 dropped. As one may observe, this is because the distribution of the responses does not follow a pattern. After carefully reviewing the data itself, we decided to omit Q9 entirely from the input features.

While we initially tested the models and data to see if eliminating certain features would increase test accuracy, we found that using all of the features, specifically in the context of the RandomForestClassifier yielded the highest test accuracy. At the beginning of our exploration of the data, we believed that because of the poor quality and misrepresentation of text data, we would remove all text-based features and simply train the model on the numerical features. This was further motivated by the naive assumption that features related to the city's perceived popularity or average temperature would be well learned by the model to make predictions. However, the text-based data simply had to be vectorized or assigned a numerical quantity in order to be used and significantly improved the accuracy of the model.

After analyzing why this may be the case, we concluded that this may be because so many of the text-based features contain references to important attributes of the city, and these were consistently repeated, allowing the model to learn and make predictions efficiently. For example, words such as "tower" or "city" were repeated enough to become consistently associated with their targets, Paris and New York City. More importantly, in more cases than not, the Q10 feature contained explicit references to the target city itself, so predictions could be made by the model with ease.

Integrating these features into our model involved extensive feature engineering. We found that many responses skewed the trend of the model, and would cause issues when the model is learning the training data. Starting by reading the CSV file of data and putting it into a Pandas dataframe, we then call cleaning functions to fill in incomplete data with the mean of the column and to correct the format of certain features for later use. In this preliminary step, we chose to vectorize the text-based data to make it easier to interpret and use when exploring which

model yields the highest test accuracy, and also later when we went about implementing a model. The data is now ready to be used by the model to learn and make predictions.

To avoid biases in training, test, and validation sets after splitting, we started by splitting the data into four sets for each city. Then we took ten percent from each set and concatenated them into one set to create a test set. Another ten percent was concatenated to create the validation set, and the remaining eighty percent of the data became the training set. We then applied a random shuffle of the rows of each dataset using Pandas' built-in features with a set seed to ensure consistency across testing.

Model

Initial Approach and Exploration

We decided it would be best to use scikit-learn's pre-built models to gain an understanding of what features would be most useful when considering prediction accuracy and to decide which model is appropriate for this specific classification task.

We experimented with random forest, decision tree, and naive Bayes models. After converting text-based features, namely Q5, 6, and 10, into numerical representations, our RandomForestClassifier had a test accuracy of 87 percent. For our DecisionTreeClassifier, we explored the impact that dropping text-based features would have on performance. This yielded a significantly lower test accuracy of 68 percent. The naive Bayes model, which required a bag-of-words to implement, had an accuracy of 74 percent on text-based features alone.

Based on our preliminary findings, we established that text-based features are valuable to a model's prediction. As a part of our feature engineering process, we converted them to numeric values for ease of usage. We then set out to build our own implementations of the

RandomForestClassifier and naive Bayes Models, using code from our labs and independent research.

We started by creating a DecisionTree class which contained the methods *build_tree, entropy, information_gain,* and *split_data*. To implement *entropy* and *information_gain* in particular, we relied on information from the Lecture 2 slides and readings to guide our design decisions.

Some challenges we faced when implementing this class was testing to see if everything was working properly before we moved on to implementing other methods and had trouble with the *split_data* method in particular, as we found it difficult to find the optimal split and use the information gain function here too, even if text-based data was vectorized.

We leveraged code from the labs to implement a naive Bayes classifier using bag-of-words. Using the lab code gave us a good structure to convert the binary classifier into a multi-classifier. Since the lab implementation of naive Bayes was entirely hand-coded there were no major challenges in constructing a new model.

Model Choice and Hyperparameters

To ensure that the evaluation of each model was comparable we ran the model several times on a new random test set with a set seed and averaged their accuracies; This gave us a more realistic measurement of each model's accuracy. We chose the accuracy metric from sklearn as our metric for evaluation since it can be calculated for every model, it is easy to calculate with each model and our overall goal for making these models was to produce one with the highest accuracy. Other metrics like Type 1 and Type 2 errors didn’t make sense in this

context since false-positive and false-negative when predicting a city has no meaning, neither was more important than the other; they simply represented a wrong output.

We identified the RandomForestClassifier to consistently yield the highest accuracy on the test set. Our group arrived at this conclusion by testing all of the other models against a consistently drawn test set, as mentioned before, we ensured that in the training, validation, and test sets for all models involved equally drawing data from each of the four label cities.

We used prediction accuracy on the test set and hyperparameters used as the most important evaluation metric when deciding to use the RandomForestClassifier. Compared to all of the other models, the RandomForestClassifier had the highest test accuracy when considering a consistent test set across the evaluation of all models. Hence, the RandomForestClassifier was established to be the best for the classification problem we were trying to solve. We also performed a grid search among several models to find the optimal set of hyperparameters with each model, and in that case as well, the RandomForestClassifer yielded the best results. In addition, we also considered using the naive Bayes model for Q10 and ensembling with the random RandomForestClassifier, however, this had a lower accuracy than the RandomForestClassifier on its own.

Below we will provide examples of hyperparameter combinations looked at during the grid search:

Example 1 (Random Forest):

Accuracy of RandomForestClassifier: 0.8671328671328671
Parameters of the trained RandomForestClassifier:
bootstrap: True
ccp_alpha: 0.0
class_weight: None
criterion: gini
max_depth: None
max_features: sqrt
max_leaf_nodes: None
max_samples: None
min_impurity_decrease: 0.0
min_samples_leaf: 1
min_samples_split: 2
min_weight_fraction_leaf: 0.0
n_estimators: 100
n_jobs: None
oob_score: False
random_state: None
verbose: 0
warm_start: False

Example 2 (best hyperparameters) (Random Forest):

Accuracy of the best Random Forest model: 0.8741258741258742
Parameters of the trained RandomForestClassifier:
bootstrap: False
criterion: gini
max_depth: 8
max_features: log2
max_leaf_nodes: 77
min_impurity_decrease: 0.0
min_samples_leaf: 3
min_samples_split: 9
min_weight_fraction_leaf: 0.0
n_estimators: 336
warm_start: True

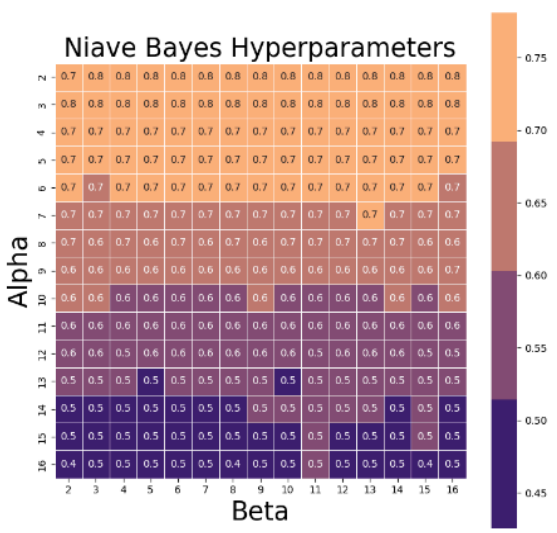
This is compared to the test accuracy and hyperparameters of the DecisionTreeClassifier and

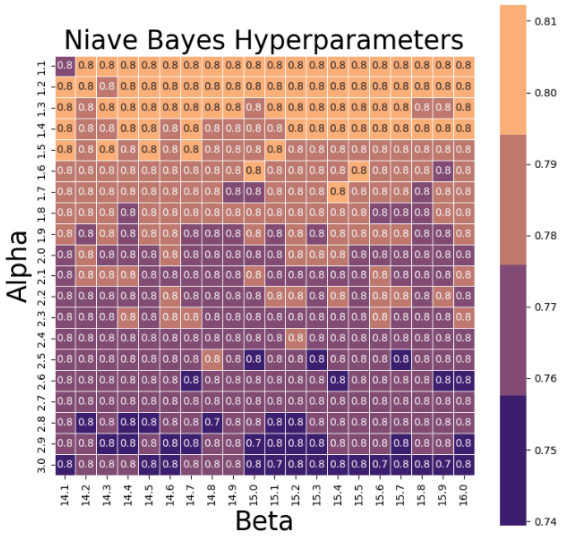
Decision Tree Hyperparameters:
Accuracy of DecisionTree: 0.7867132867132867
Parameters of the trained RandomForestClassifier:
ccp_alpha: 0.0
class_weight: None
criterion: gini
max_depth: None
max_features: None
max_leaf_nodes: None
min_impurity_decrease: 0.0
min_samples_leaf: 1
min_samples_split: 2
min_weight_fraction_leaf: 0.0
random_state: None
splitter: best

KNeighbors Hyperparameters:
Accuracy of KNeighbors Classifier: 0.43356643356643354
Parameters of the trained KNeighbors Classifier:
algorithm: auto
leaf_size: 30
metric: minkowski
metric_params: None
n_jobs: None
n_neighbors: 5
p: 2
weights: uniform

Naive Bayes:

The naive Bayes model has two hyperparameters, alpha and beta. Alpha is an additional value given to every occurrence of a word and proves a small probability of unseen words. Beta is an additional value given to the probability of every class. Initially alpha and beta both were 2. After a quick grid search, we found that alpha equals 2 and beta equals 15 produced the highest accuracy of 78%. Then we did some more fine-tuning in that region, we found that alpha equals 1.1 and beta equals 14.2 produced the highest accuracy of 81%.





Final Model Choice

Our final model choice that we submitted in the *pred.py* file was a RandomForestClassifier model with hyperparameters listed: 'bootstrap': False, 'criterion': 'gini', 'max_depth': 8, 'max_features': 'log2', 'max_leaf_nodes': 77, 'min_impurity_decrease': 0.0, 'min_samples_leaf': 3, 'min_samples_split': 9, 'min_weight_fraction_leaf': 0.0, 'n_estimators': 336, 'warm_start': True.

The bulk of the code is stored serialized in a pickle file, which we submitted along with *pred.py*. We import the pickle module in *pred.py* and run the contents of the file under the *predict_all* function.

Prediction

We expect our model to perform at a test accuracy of 87 percent. Our quote for this figure is based on the numerous tests we conducted using the optimal parameters on our test data. As was listed above, the following are a couple of instances of testing:

Test 1 - Test Accuracy of RandomForestClassifier: 0.8661971830985915

Test 2 - Test Accuracy of RandomForestClassifier: 0.8741258741258742

Test 3 - Test Accuracy of RandomForestClassifier: 0.8697183098591549

Therefore, from rigorous testing and analysis of the trends of the test accuracy, we can conclude that on a given test set, our model will perform with 87% accuracy, as this is the mean of the set of test accuracies we found when testing.

Workload Distribution

Anirudh (Rudy) Navin Arbatti

- Explored model possibilities using sci-kit learn to evaluate DecisionTreeClassifier's performance on data
- Designed and implemented initial DecisionTreeClass when experimenting with hand coding a decision tree
- Helped complete bulk of report and maintained documentation of the project

Nikhil Iyer:

- Created data cleaning script to clean the dataset
- Applied feature engineering to the dataset
- Created visualizations and graphs for the features
- Created testing script to evaluate multiple models across all combinations of features
- Fine-tuned the hyperparameters of the selected models

Devin Afonso Mendes:

- Create Naive Bayes Model
- Hyperparameter tuning for Naive Bayes Model
- Initial plotting of the data
- Helped create pred.py code

Musab Muhammad:

- Explored a possible Linear Regression model
- Helped design DecisionTree class by implementing an information gain method
- Explanation of Figures in Paragraphs 1 and 2 of Data
- Figure 2 and 4 descriptions