Coffee Quality Analysis and Prediction - Different Models Comparison

Teams Name: Orchid

Jinyin Chai, Qiyang Xie, Ying Shen, Zifei Song

Abstract

This report presents a comparative analysis of several machine learning algorithms applied to the problem of predicting the quality of Arabica coffee beans based on various features we selected, including aroma, flavor, aftertaste, acidity, body, balance, uniformity, clean cup, sweetness, moisture, Quakers, defects, and altitude. The dataset used in this study was obtained from the Coffee Quality Institute (CQI) and contained information on over 1300 coffee samples from different regions around the world. We used linear regression, logistic regression, K-nearest neighbors classification (KNN), and Random Forest to build models that can accurately classify coffee beans into specialty grade or non-specialty grade categories. The performance of each model was evaluated using several metrics, including accuracy, precision, recall, and ROC AUC. Our results showed that Linear regression model and Random Forest model have a better performance in this coffee quality prediction. Overall, our research demonstrates the potential of machine learning algorithms in analyzing and predicting coffee quality. The results can be useful for coffee producers, traders, and customers in making informed decisions about the quality of their products.

I. Introduction

As we all know, coffee is one of the most widely consumed beverages in the world. According to the international coffee organization, there is an estimated 2.25 billion cups of coffee consumed every day (ICO, 2021). The quality of coffee is an important factor for both producers and consumers since it can affect the market value of coffee beans as well as the taste and satisfaction of the end product. As consumers, when faced with a wide variety of coffee, we are always at a loss to choose suitable taste and excellent quality coffee beans. This was the motivation behind our study, which aims

to help consumers better understand the factors that contribute to high-quality coffee by providing a more objective and standardized way of assessing coffee quality so that they can make more informed choices about the coffee they purchase based on their preferences and expectations.

After having a clear motivation, we decided to put it into this data mining research. Specifically, we will use different machine learning algorithms we've learned in this course, including linear regression, logistic regression, K-nearest neighbor classification, and Random Forest, to build prediction models and explore the relationship between various features of coffee beans and the quality scores. By comparing the performance and prediction results of different models, we can find out the most suitable model for this problem.

II. Methodology

The methodology we used for solving this problem included the following steps. First, we did the data preprocessing and visualization. We collected the data and handled missing values and values out of reasonable bounds. Then we fitted the data into four prediction models, which are the Linear Regression model, the Logistic Regression model, the K Nearest Neighbors Classification model, and the Random Forest model. Lastly, we compared the results of the four prediction models and came to the conclusion that which models were better at solving this problem.

III. Results and Interpretations

A. Data Preprocessing and Visualization

To extract the data, we first save the downloaded data to our GitHub. Then, we used the pandas library in Python. We first imported the pandas library and used the read_csv() function to read the CSV file into a pandas data frame.

```
url = "https://raw.githubusercontent.com/JinyinChai/CS6620_Orchid/main/arabica_data_cleaned.csv"
df = pd.read_csv(url, header=None)
new_header = df.iloc[0]
df = df[1:-1]
df.columns = new_header
df
```

Data frame sample (head):

	NaN	Species	Owner	Country.of.Origin	Farm.Name	Lot.Number	Mill	ICO.Number	Company	Altitude	...	Color	Category.Two.Defects	Expiration	Certification.Body
1	1.0	Arabica	metad plc	Ethiopia	metad plc	NaN	metad plc	2014/2015	metad agricultural developmet plc	1950-2200	...	Green	0	April 3rd, 2016	METAD Agricultural Development plc

	Certification.Address	Certification.Contact	unit_of_measurement	altitude_low_meters	altitude_high_meters	altitude_mean_meters
	309fcf77415a3661ae83e027f7e5f05dad786e44	19fef5a731de2db57d16da10287413f5f99bc2dd	m	1950	2200	2075

To do data cleaning, we dropped some columns which contain a lot of null values. We choose only numeric type columns to do analysis for convenience. The final data frame we used for our analysis is 16 columns, including Aroma, Flavor, Acidity, etc. We found that there was some abnormal value for the altitude in our dataset, which exceeded the reasonable altitude, so we decided to drop those rows in this step. In addition, we also checked for missing values in our datasets and filled in reasonable values such as 0 in that cell to make sure there were no missing values in our dataset.

```
[7]  # Choose only numeric type to do analysis
     df = df[['Aroma', 'Flavor', 'Aftertaste', 'Acidity', 'Body', 'Balance', 'Uniformity', 'Clean.Cup', 'Sweetness',
              'Cupper.Points', 'Moisture', 'Quakers', 'Category.One.Defects', 'Category.Two.Defects',
              'altitude_mean_meters', 'Total.Cup.Points']]
     df = df.astype(float)
```

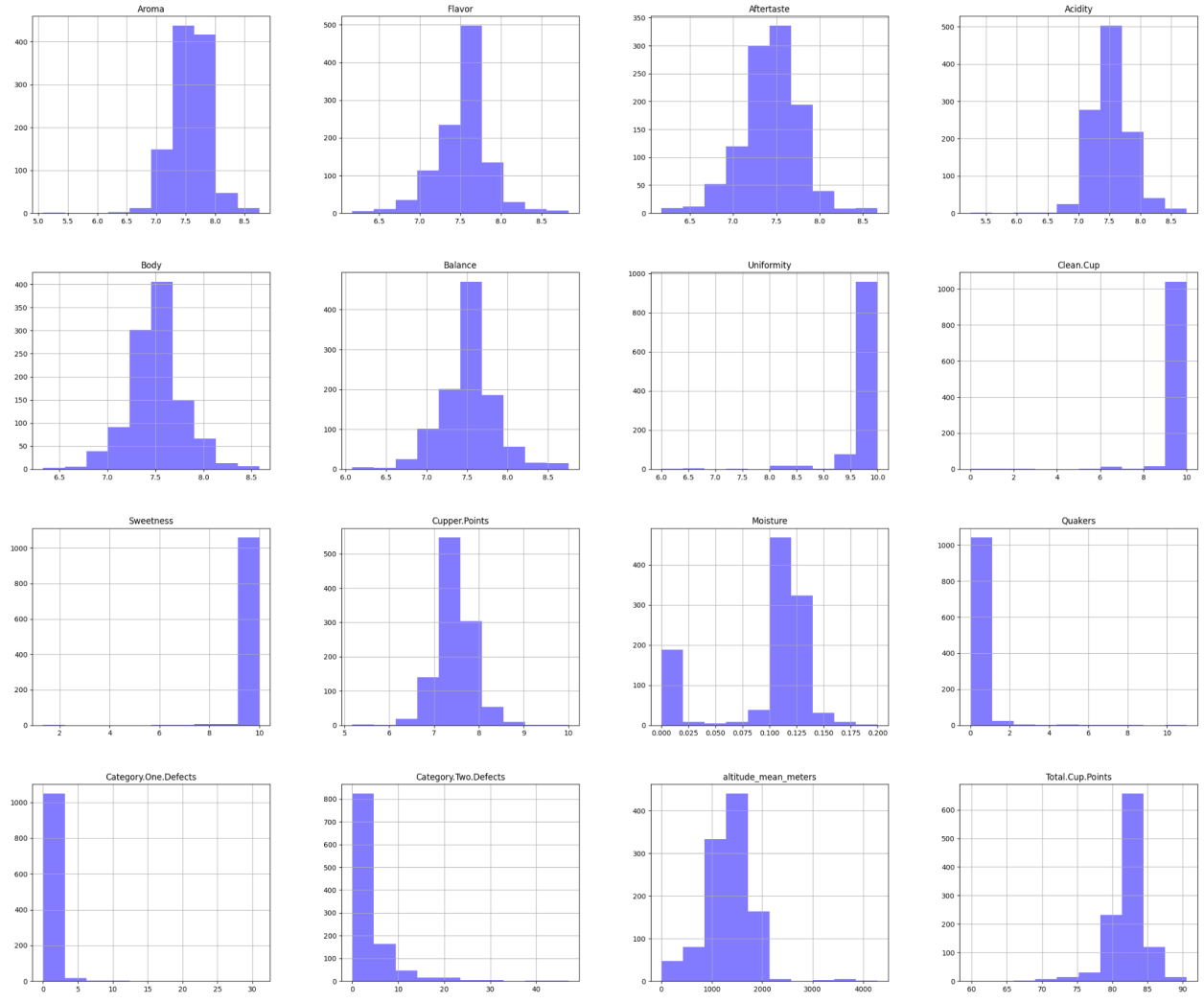
```
[8]  # only use data with reasonable altitude
     df = df[df['altitude_mean_meters']<8850]
```

```
[9]  # fill the only null value of Quakers to 0
     df['Quakers'] = df['Quakers'].fillna(0)
```

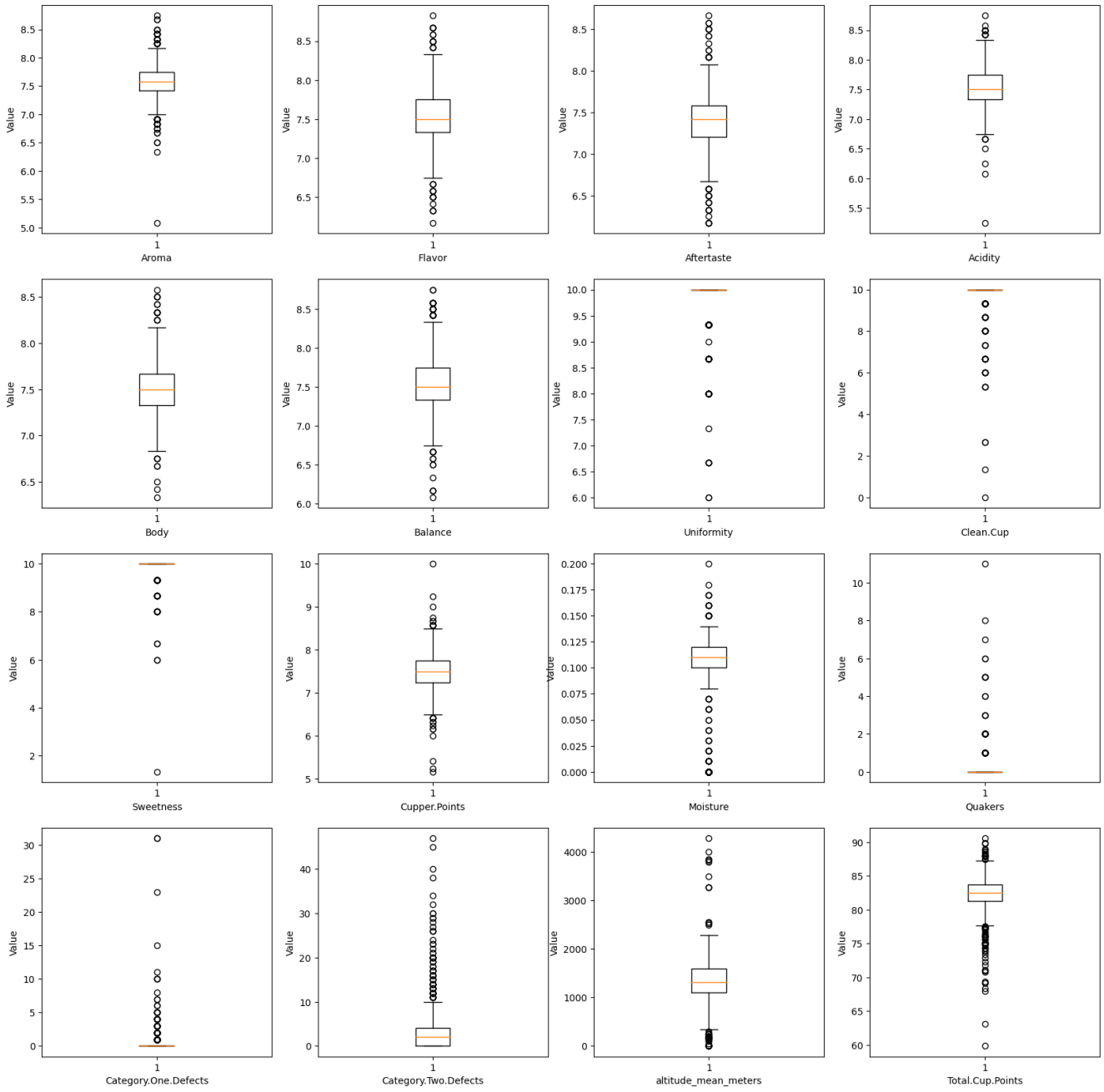
```
     # no NaN in dataset
     df.isnull().sum()
```

```
0
Aroma                0
Flavor               0
Aftertaste           0
Acidity              0
Body                 0
Balance              0
Uniformity           0
Clean.Cup            0
Sweetness            0
Cupper.Points        0
Moisture             0
Quakers              0
```

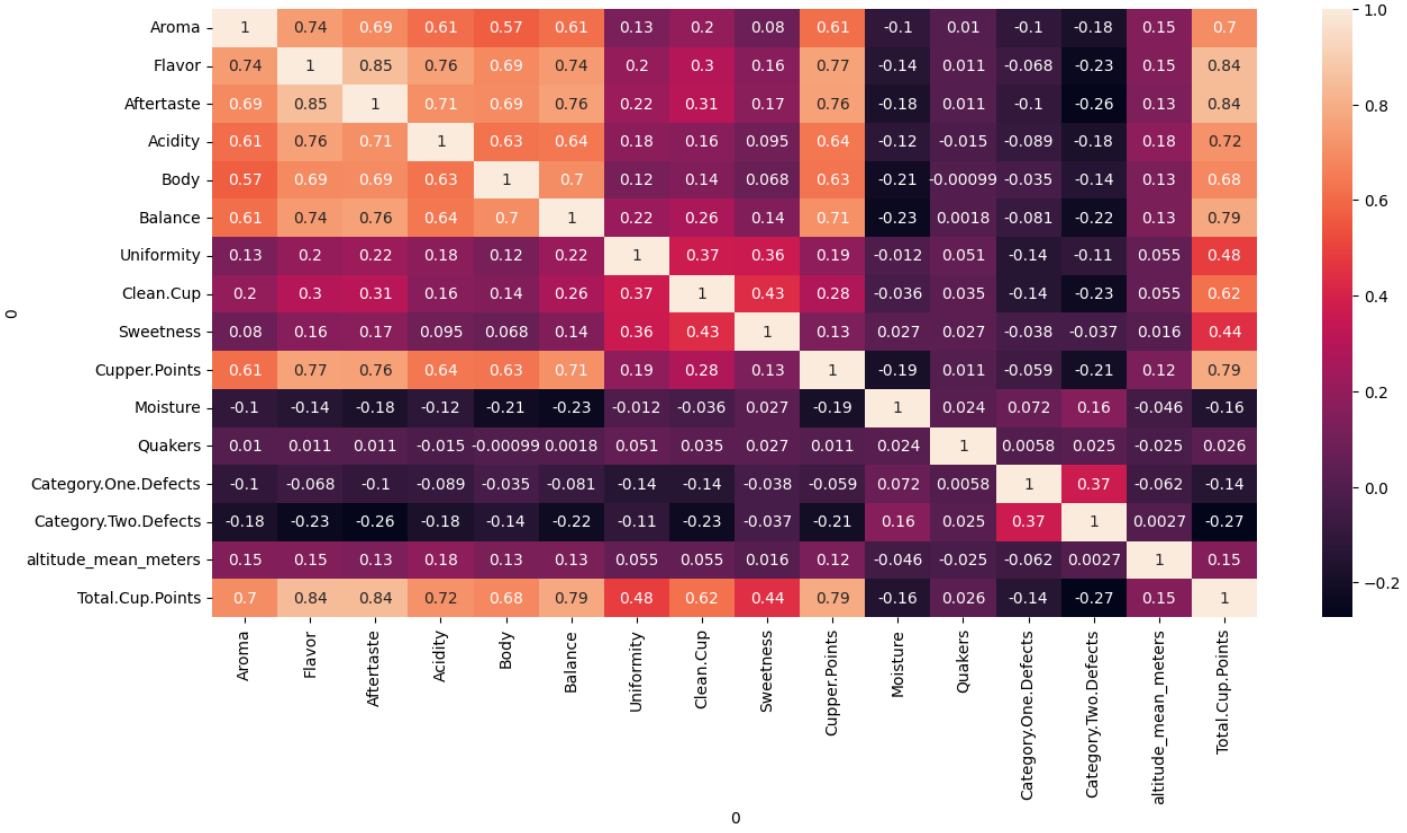
For the data visualization, we first use histograms to see the data distribution and whether there is any abnormality in the data set. From the following graphs, we can see the Aroma, Flavor, Aftertaste, Acidity, Body, Balance, Cupper.Points, and altitude are generally high in the center and low at the edges, which is more continuous. Besides, the graphs of Uniformity, Clean Cup, Sweetness, Quakers, and Category defect are high at the edges and low at the center, which are not very coherent. The Moister feature is the special one with two peaks, one higher at the center and one lower at the left edge.



Then we also used box plots to visualize the data. Box plots can help us clearly see each feature's distribution and outlier. From the graphs below, we can see Uniformity, Clean Cup, Sweetness, Quakers, and Category One defect has significant outliers.



We used a heatmap to visualize the correlations between the features in our datasets. Through the heat map, we can more intuitively feel the intensity of the correlation between different features. For example, we can see the correlation between Aroma, Flavor, Aftertaste, Acidity, Body, and Balance are relatively higher than the other features.



B. Linear Regression

Introduction

Linear regression is a statistical technique used to model the relationship between a dependent variable (outcome variable) and one or more independent variables (explanatory variables). Linear regression assumes that the relationship between the variables is linear and that the errors are normally distributed and have constant variance. The goal of linear regression is to find the line of best fit that describes the relationship between the variables.

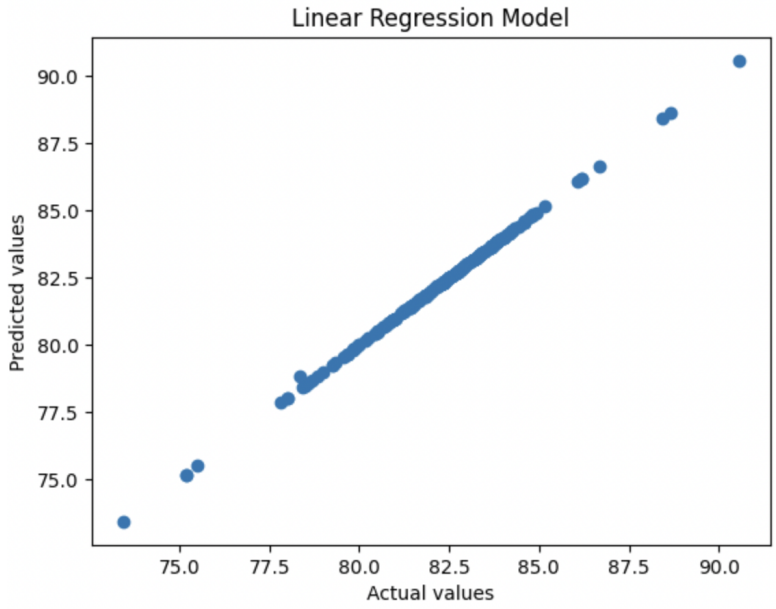
In this part, we use a linear regression model to predict the quality of Arabica coffee beans(evaluated using total cup points) based on 15 features in our dataset.

Model Training and Evaluation

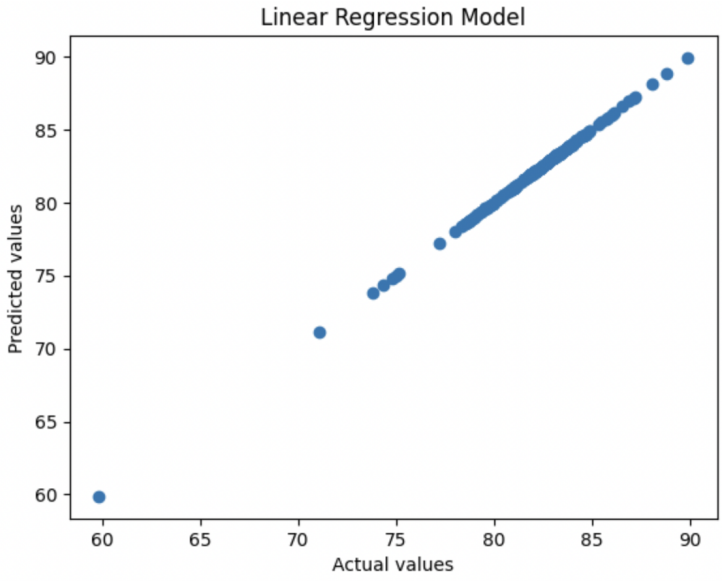
To generate the linear regression model, we use the linear_model.LinearRegression() function which is available via the sci-kit-learn library. To run the model on the dataset, we first split the dataset into training and testing sets(80 and 20 percent, respectively), then we use all the features to fit the model on the training set and predict the target feature Total.Cup.Points. with the fitted model on the testing set.

To evaluate the performance of the linear regression model, we calculated Mean Squared Error(MSE) to measure the average of the squared differences between the predicted values and the actual values and Variance Score(R-squared (R^2) value) to measure the proportion of variance in the dependent variable that can be explained by the independent variables. The coefficients are also reported to estimate the relationship between the independent variables (predictors) and the dependent variable (target). The MSE was calculated as 0.0012, and the Variance Score is 0.9997.

We can visualize the model below. An almost straight line indicates that the regression model well predicts the coffee quality.

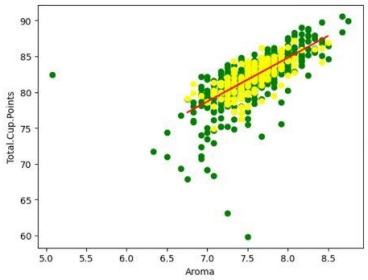


We also performed cross-validation (K-fold) to evaluate the model, for the model generated with all fractures, has a high Cross-validated variance score of 0.99 and a low Standard deviation: of 0.001. The scatter below showed that it forms a straight line, indicating the model well predicted the outcome variable, which is the total cup points in our study.

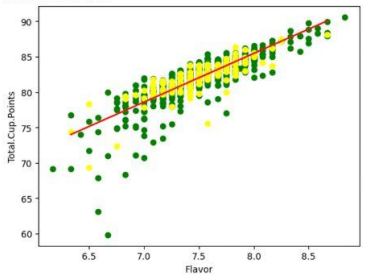


Then, we used each feature alone to fit a linear regression model on the training set. The coefficients, mean squared error and variance score were reported to evaluate the model. The results were visualized below. In the plots, the red line is the linear regression line, the yellow points are the test set, and the green points are the training set. The coefficients, MSE, and variance scores are above the plots.

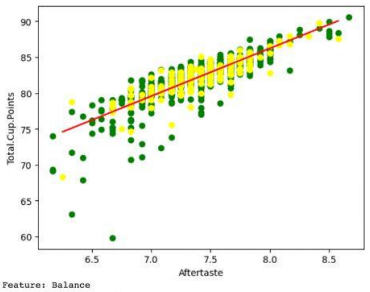
Feature: Aroma
Coefficients: [6.10439329]
Mean squared error: 2.19
Variance score: 0.48



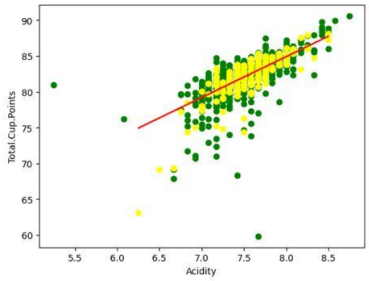
Feature: Flavor
Coefficients: [6.87132971]
Mean squared error: 1.48
Variance score: 0.68



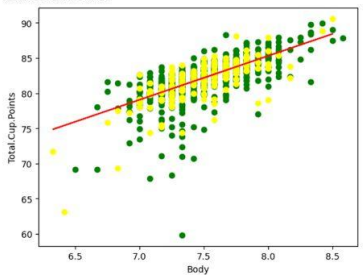
Feature: Aftertaste
Coefficients: [6.64223169]
Mean squared error: 1.55
Variance score: 0.72



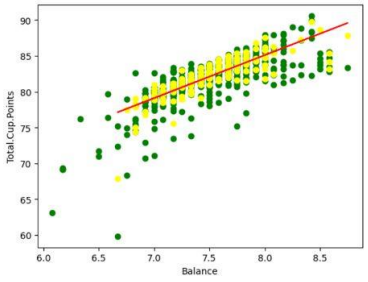
Feature: Acidity
Coefficients: [5.6972416]
Mean squared error: 3.14
Variance score: 0.62



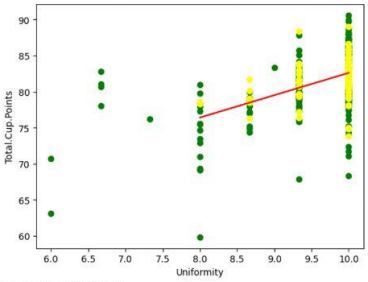
Feature: Body
Coefficients: [6.25366405]
Mean squared error: 3.78
Variance score: 0.54



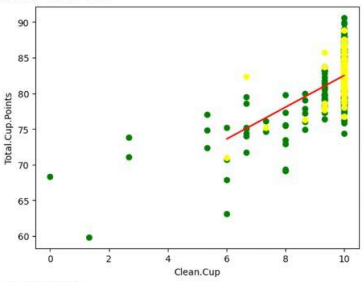
Feature: Balance
Coefficients: [5.98990021]
Mean squared error: 1.84
Variance score: 0.69



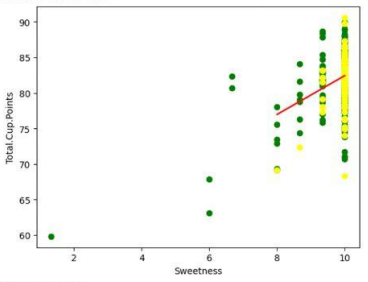
Feature: Uniformity
Coefficients: [3.09669351]
Mean squared error: 4.03
Variance score: 0.07



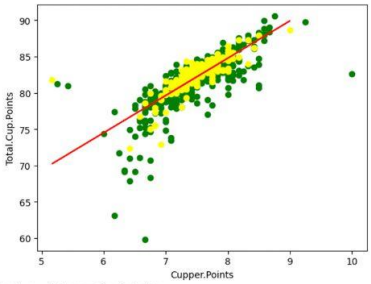
Feature: Clean.Cup
Coefficients: [2.2306102]
Mean squared error: 3.75
Variance score: 0.18



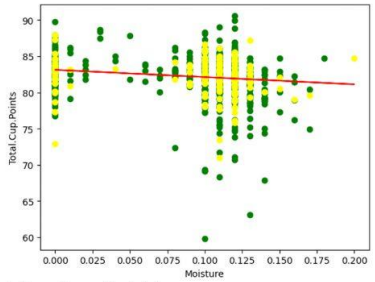
Feature: Sweetness
Coefficients: [2.72896047]
Mean squared error: 6.74
Variance score: 0.12



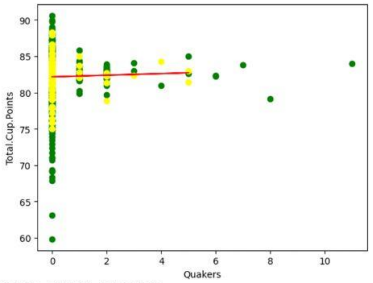
Feature: Cupper.Points
Coefficients: [5.13192492]
Mean squared error: 1.85
Variance score: 0.62



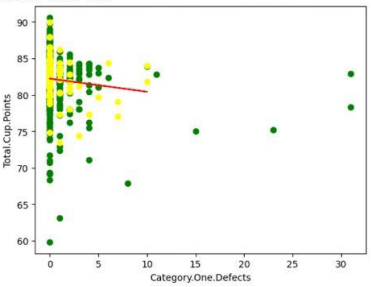
Feature: Moisture
Coefficients: [-10.02246473]
Mean squared error: 5.21
Variance score: 0.01



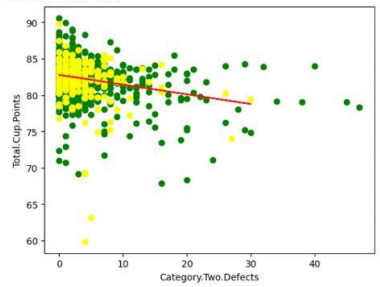
Feature: Quakers
Coefficients: [0.1113541]
Mean squared error: 3.76
Variance score: -0.01



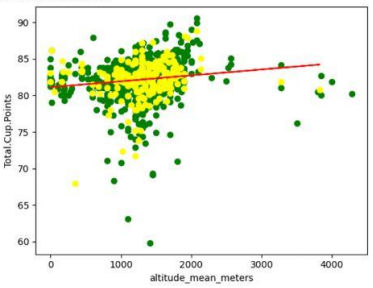
Feature: Category.One.Defects
Coefficients: [-0.18023328]
Mean squared error: 4.15
Variance score: 0.01



Feature: Category.Two.Defects
Coefficients: [-0.13350579]
Mean squared error: 9.77
Variance score: 0.03

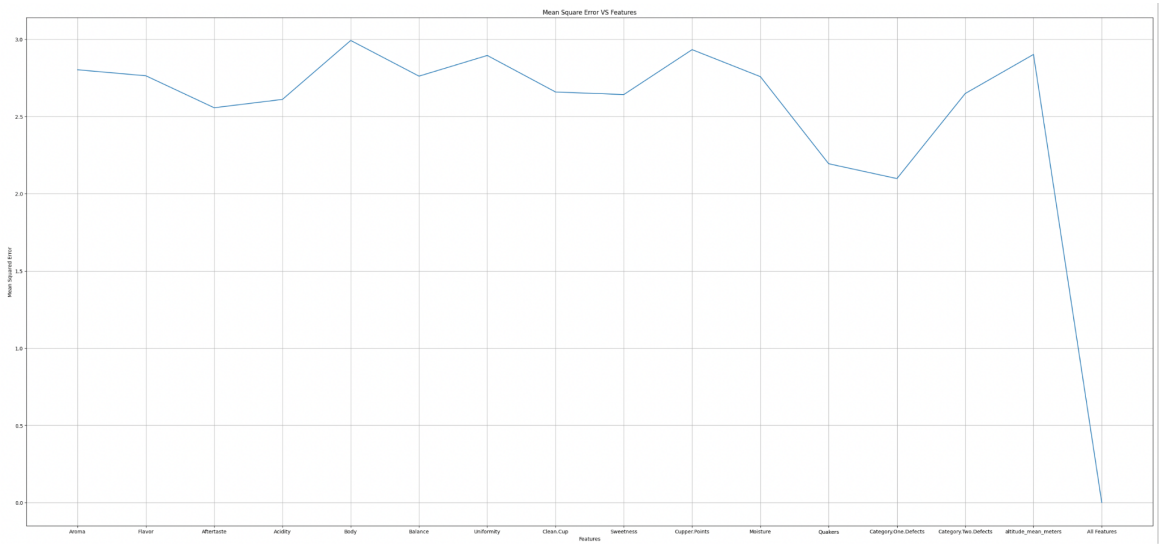


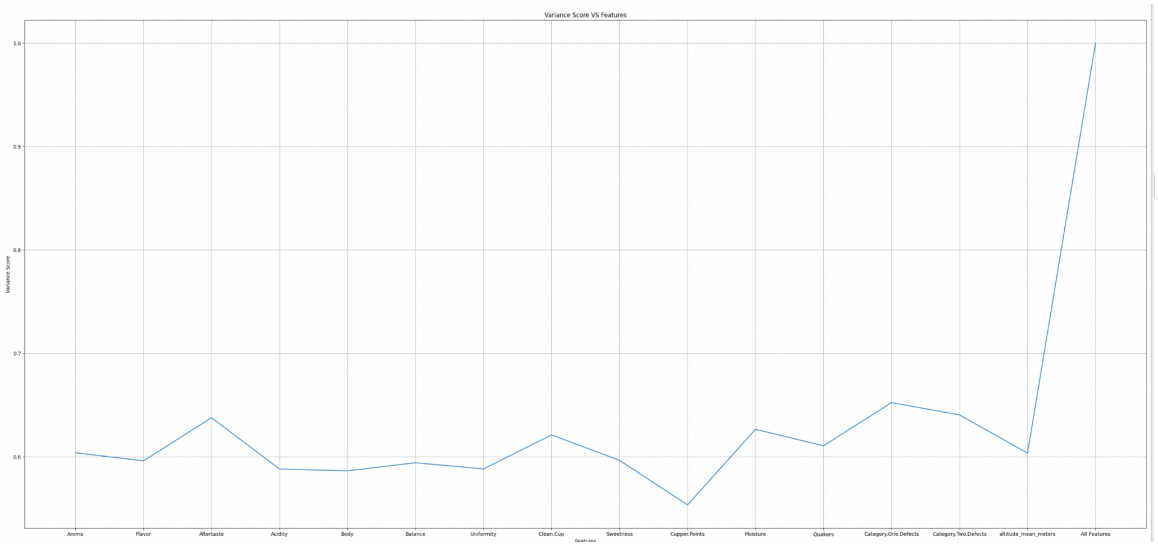
Feature: altitude_mean_meters
Coefficients: [0.00080967]
Mean squared error: 6.34
Variance score: 0.03



Then 10 iterations were performed to generate the linear regression model using single features and all features to get the average coefficients, mean squared error, and variance score. The results show that the model generated using a single feature had a mean squared error of 2.24 - 3.27 and a variance score of 0.57 - 0.67. The model generated using all features had the lowest MSE of 0.0005 and the highest variance score of 0.9999.

To compare the model performance, the MSE vs features and variance score vs features plots were generated. From the two plots, “Aftertaste” and “Category.One.Defects” are the two features that are most predictive for the target feature, as it has a lower average MSE and higher average variance score. Some features(such as “Body” and “Cupper.Points”) have very high mean squared error and low variance scores, showing that they don’t fit the linear regression model to predict the target feature. We can see that it is better to use all the features for a linear regression model to predict the target feature, as “All features” has the lowest average mean squared error and highest average variance score.





C. Logistic Regression

Introduction

Logistic regression is a statistical method used to analyze the relationship between a binary dependent variable and one or more independent variables. The dependent variable is a categorical variable that takes on one of two values, often represented as 0 or 1. The independent variables, also known as predictor variables or features, can be either categorical or continuous.

The goal of logistic regression is to find the best-fitting model that can predict the probability of the dependent variable taking on the value 1, given the values of the independent variables. The output of logistic regression is a predicted probability of the dependent variable, which can be transformed into a binary outcome using a decision threshold.

In this part, we applied logistic regression to predict the quality of Arabica coffee beans based on features in our dataset. We used a binary dependent variable, specialty grade or non-specialty grade, and several predictor variables to build a logistic regression model that can accurately classify coffee beans into these two categories. The results of this analysis can provide useful insights for coffee producers and traders in optimizing the quality of their products.

Model Training and Evaluation

We used logistic regression to build a model that can predict the quality of Arabica coffee beans based on several features. The dataset was randomly split into a training set (80% of the data) and a test set (20% of the data) using stratified sampling to ensure that the proportion of specialty grade and non-specialty grade samples was similar in both sets.

Then, we fitted the logistic regression model to the training data using all 15 features to predict the target variable, which is the quality of the coffee beans. We reported the coefficients, mean squared error, and variance score for the model on the test set. The results show that the model had a mean squared error of 0.11 and a variance score of 0.89.

```
[ ] # Use all the features (1-15) to fit the logistic regression model for feature 16 using the training set
    from sklearn.linear_model import LogisticRegression
    logi_reg = LogisticRegression(max_iter=3000)
    logi_reg.fit(X_train, y_train)
    y_pred = logi_reg.predict(X_test)
```

```
# Report the coefficients, mean squared error and variance score for the model on the test set.
from sklearn.metrics import mean_squared_error
print("Coefficients:", logi_reg.coef_)
print("Mean squared error: %.2f" % mean_squared_error(y_test, y_pred))
print("Variance score: %.2f" % logi_reg.score(X_test , y_test))
```

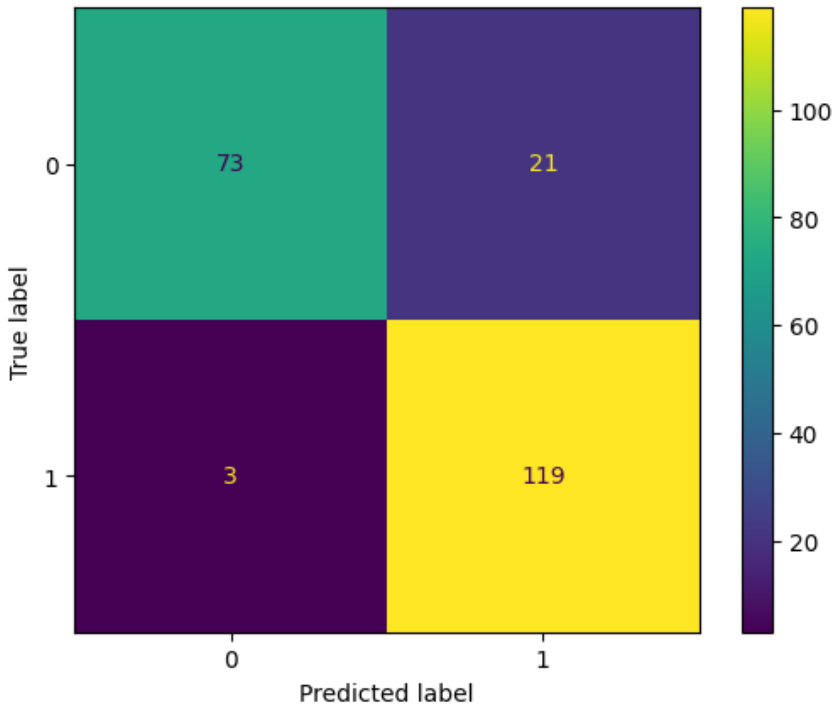
```
Coefficients: [[ 9.30890273e-02  1.66853975e+00  1.66605763e+00  7.47204502e-01
  -1.08346654e+00  8.41268096e-01 -9.66692837e-03 -2.45060377e-01
  -4.08031139e+00  2.05329318e+00 -2.45664818e-01  2.32035323e-01
   1.28397526e-02 -3.15171714e-02  2.32765105e-04]]
Mean squared error: 0.11
Variance score: 0.89
```

To evaluate the performance of the model, we used several metrics, including accuracy, precision, recall, classification report, and the confusion matrix.

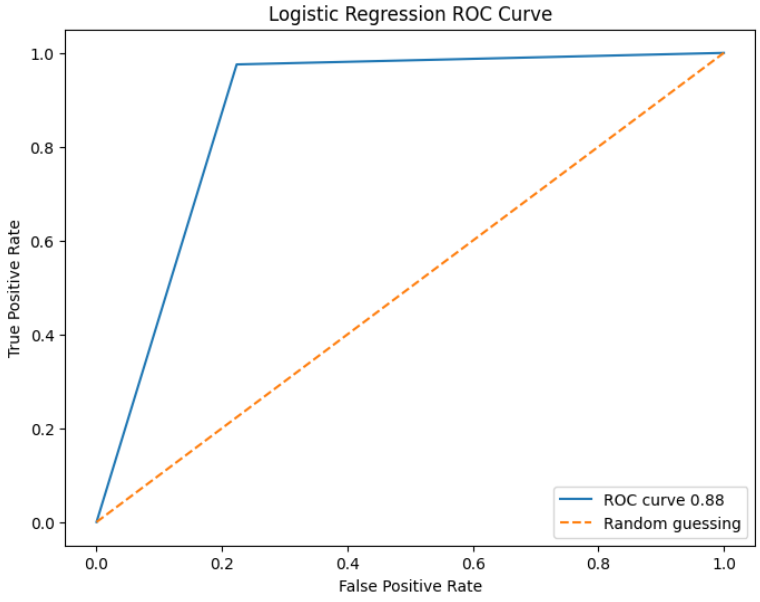
```python
# Repot the accuracy, precision score, recall value, classfication report, and confusion matrix for the model
accuracy_score_1 = accuracy_score(y_test, y_pred)
precision_score_1 = precision_score(y_test, y_pred)
recall_score_1 = recall_score(y_test, y_pred)
classification_report_1 = classification_report(y_test, y_pred)
confusion_matrix_1 = confusion_matrix(y_test, y_pred)
print("The accuracy of Logistic Regression on the test data is:", accuracy_score_1)
print("The precision score is:", precision_score_1)
print("The recall value is:", recall_score_1)
print("The classification report is:\n", classification_report_1)
print("The confusion matrix for this experiment is:", confusion_matrix_1)
```

```
The accuracy of Model 1 on the test data is: 0.8888888888888888
The precision score is: 0.85
The recall value is: 0.9754098360655737
The classification report is:
              precision    recall  f1-score   support

           0       0.96      0.78      0.86        94
           1       0.85      0.98      0.91       122

    accuracy                           0.89       216
   macro avg       0.91      0.88      0.88       216
weighted avg       0.90      0.89      0.89       216

The confusion matrix for this experiment is: [[ 73  21]
 [  3 119]]
```

The following confusion matrix shows that the model predicted 73 true positives, 12 false positives, 3 false negatives, and 119 true negatives, resulting in an accuracy of 0.888. The precision score was 0.85, and the recall value was 0.975.



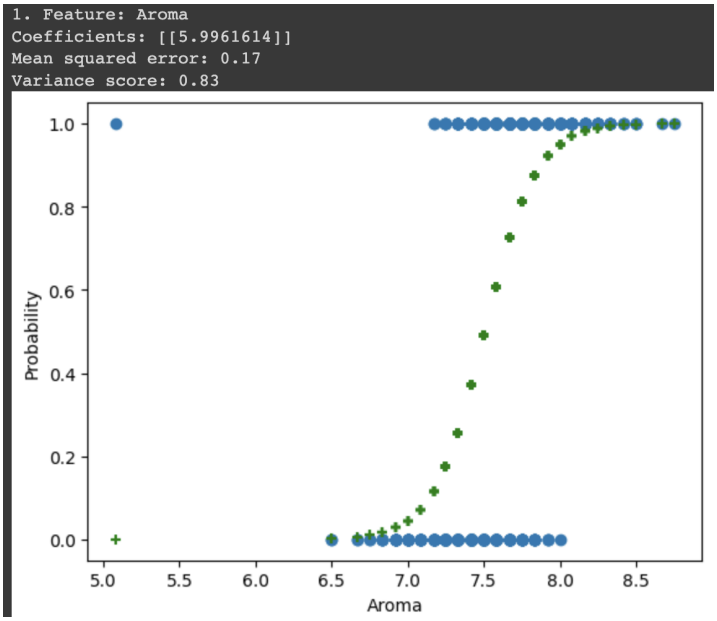
The following ROC Curve shows AUC of the logistic regression is 0.86.



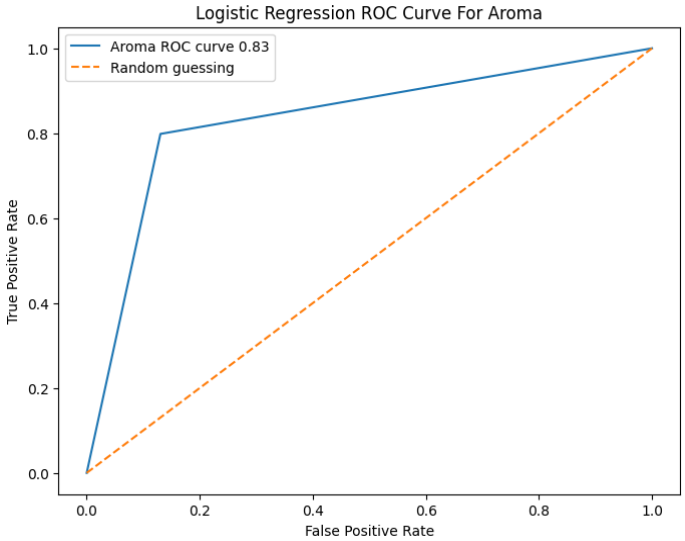
We also fitted a logistic regression model to each feature alone on the training set and reported the coefficients, mean squared error and variance score for each model on the test set. We also generated 15 plots of the logistic regression models generated on each feature. Each plot distinctly

shows the logistic regression probability curve and the ROC curve for the logistic regression of each feature.

The results show that feature Flavor had the highest coefficient and the lowest mean squared error, indicating that it is the most important feature in predicting the quality of coffee beans. The following logistic regression probability curve shows the probability of a good quality coffee bean versus the flavor of this coffee bean.



And we also provide the ROC curve for each future to see the AUC result. The following graph shows the AUC of the flavor feature is 0.87.



Feature	Coefficients	MSE
Aroma	6.599	0.19
Flavor	8.061	0.12
Aftertaste	7.600	0.18
Acidity	6.681	0.18
Body	5.754	0.21
Balance	6.374	0.18
Uniformity	1.624	0.35
Clean.Cup	1.542	0.31
Sweetness	0.964	0.42
Cupper.Points	6.865	0.15
Moisture	-2.047	0.40
Quakers	0.088	0.41
Category.One.Defects	-0.100	0.46
Category.Two.Defects	-0.066	0.36
altitude_mean_meters	0.001	0.44

Overall, our logistic regression model provides a reliable and accurate approach to predicting the quality of Arabica coffee beans and can be used to guide decision-making in the coffee industry.

D. Random Forest Classification

Introduction

Random Forest, which contains a large number of decision trees as the name implies. Each individual decision tree inside our random forest comes out with their own output for the class prediction. And the final result will be the class with the most votes from the decision trees. Note that the decision trees inside the random forest are uncorrelated to generate the unbiased prediction of class. The low correlation here is the key to good performance. Uncorrelated models can produce predictions that are more accurate than any of the individual ones. In addition, the decision trees are protecting each other from their own errors leads to this wonderful effect.

In this section, we use random forest classifiers to perform the classification task of the type of Arabic coffee beans based on the valid features we filtered from our dataset. At first, we test output of a default random forest classifier with 100 estimators/decision trees and no max depth of the trees. Then I pruned the default classifier by finding the number of estimators and maximum depth with the best performance. After finding the optimal number of estimators and the optima maximum depth, I'm able to evaluate the output of the tuned classifier. Spoil alert, the tuned classifier produces a much better classification.

Model Training and Evaluation

Default Random Forest Classifier with n_estimator=100 and max_depth=None

With the default random forest classifier, we have the ROC curve with AUC = 0.91, mean squared error=0.08 and variance score=0.92. Check the code for model initiation and evaluation for the prediction result.

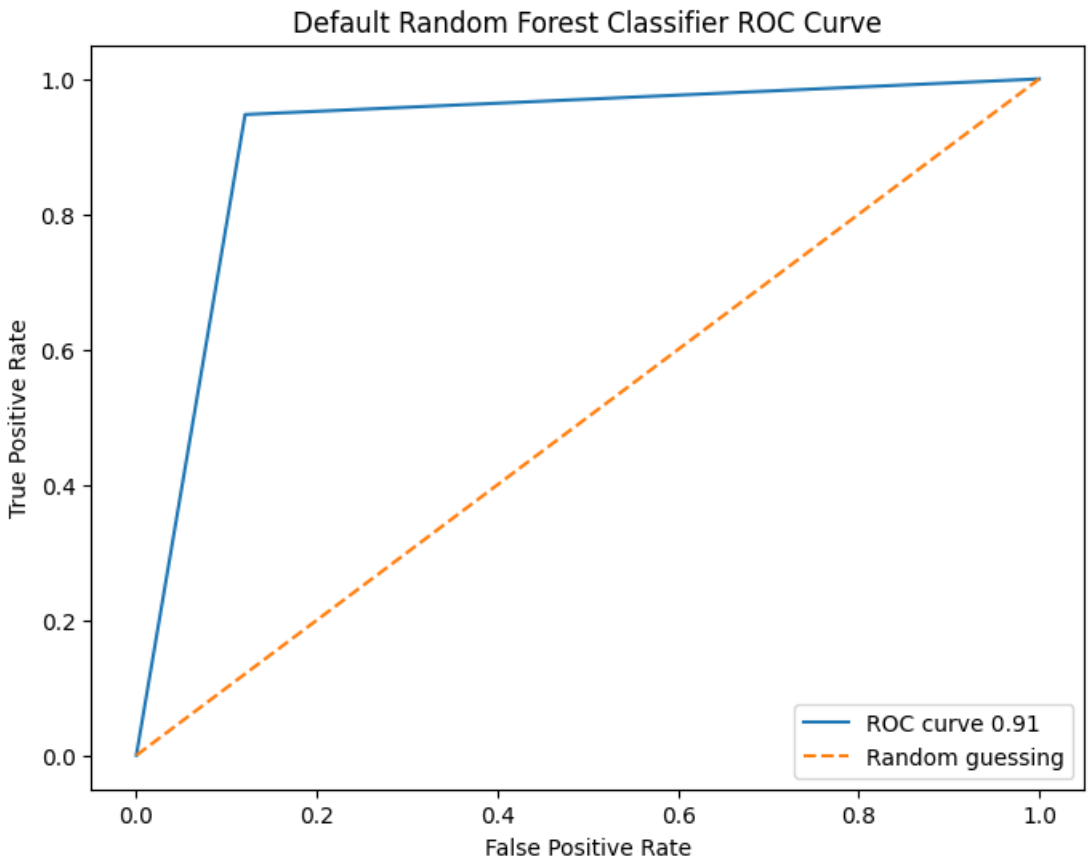
```
from sklearn.ensemble import RandomForestClassifier

# Default model with no restrains on number of estimators and depth
rf_clf = RandomForestClassifier()
rf_clf.fit(X_train, y_train)
y_pred_rf = rf_clf.predict(X_test)
```

```
print("Test Score:", rf_clf.score(X_test, y_test))
print("Mean squared error: %.2f" % mean_squared_error(y_test, y_pred_rf))
print("Variance score: %.2f" % rf_clf.score(X_test , y_test))
```

```
Test Score: 0.9212962962962963
Mean squared error: 0.08
Variance score: 0.92
```

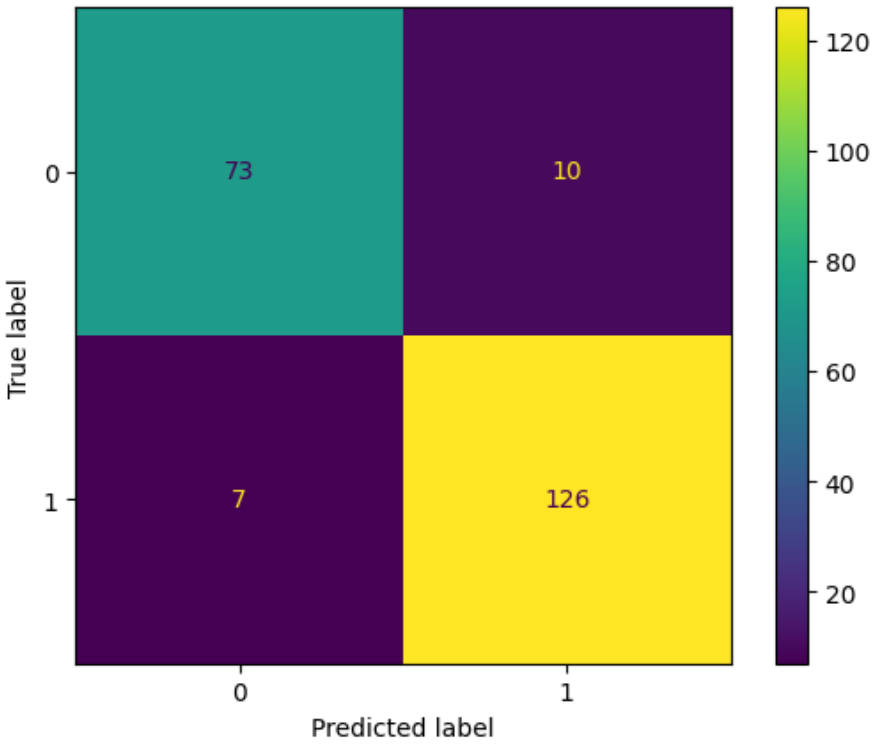
With the code for generating the ROC curve and calculating the AUC.

```
fpr_rf, tpr_rf, _ = roc_curve(y_test, y_pred_rf)
roc_auc_rf = metrics.roc_auc_score(y_test, y_pred_rf)
plt.figure(figsize=(8, 6))
plt.plot(fpr_rf, tpr_rf, label="ROC curve %0.2f"% roc_auc_rf)
plt.plot([0, 1], [0, 1], linestyle="--", label="Random guessing")
plt.title("Default Random Forest Classifier ROC Curve")
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.legend()
plt.show()
```



Plus the code to calculate the confusion matrix and then display it.

```
cfm_rf = metrics.ConfusionMatrixDisplay(confusion_matrix_rf, display_labels=None)
cfm_rf.plot()
plt.show()
```



Tuning parameters of random forest classifier:

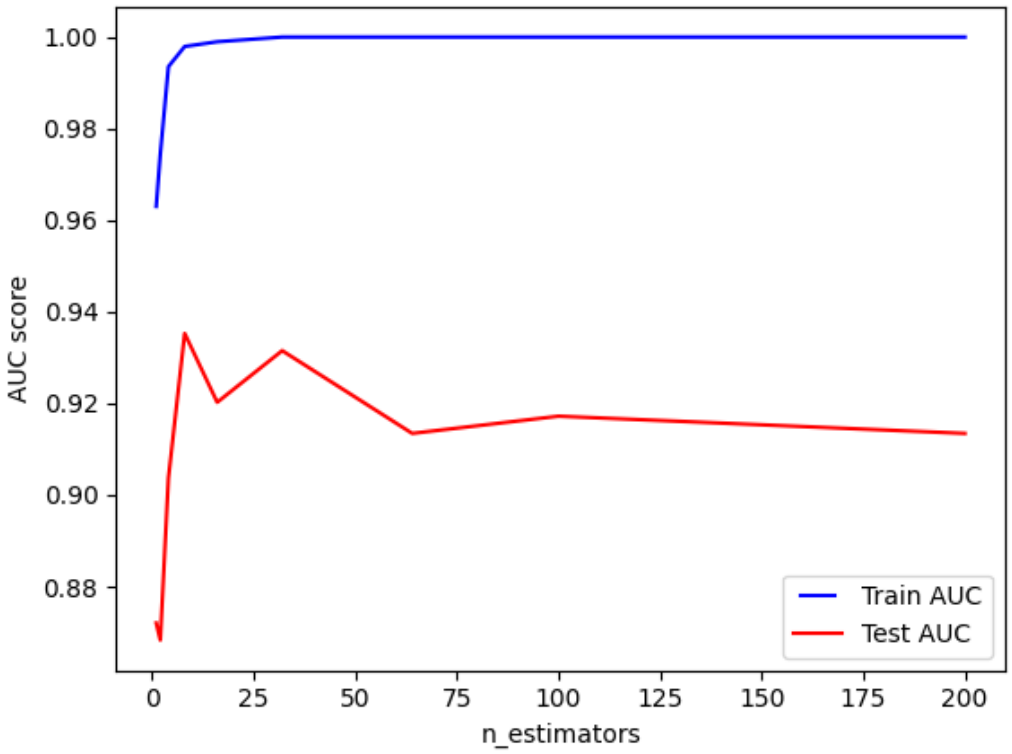
1. Number of estimators

The possible general acknowledgment of the number of estimators may be “the more the better”, but does the truth really agree with this? From the plot below we compared the train and test AUC with different numbers of estimators with 1, 2, 4, 8, 16, 32, 64, 100, and 200. It is easy to observe that once the number of estimators exceeds 32, there is a clear overfitting based on the test AUC. And the train AUC reaches its high even earlier at about 16.

Therefore, we need to stop at 32 decision trees, if we keep increasing the tree number it will only cause a decrease in the model’s performance.

Here we have the code for n_estimator pruning.

```
# n_estimators
n_estimators = [1, 2, 4, 8, 16, 32, 64, 100, 200]
train_results = []
test_results = []
for num in n_estimators:
  rf = RandomForestClassifier(n_estimators=num, n_jobs=-1)
  rf.fit(X_train, y_train)
  train_pred = rf.predict(X_train)
  roc_auc_rf_tuning = roc_auc_score(y_train, train_pred)
  train_results.append(roc_auc_rf_tuning)

  y_pred_rf_tuning = rf.predict(X_test)
  roc_auc_rf_tuning = roc_auc_score(y_test, y_pred_rf_tuning)
  test_results.append(roc_auc_rf_tuning)

line1, = plt.plot(n_estimators, train_results, 'b', label="Train AUC")
line2, = plt.plot(n_estimators, test_results, 'r', label="Test AUC")
plt.legend(handler_map={line1: HandlerLine2D(numpoints=2)})
plt.ylabel('AUC score')
plt.xlabel('n_estimators')
plt.show()
```

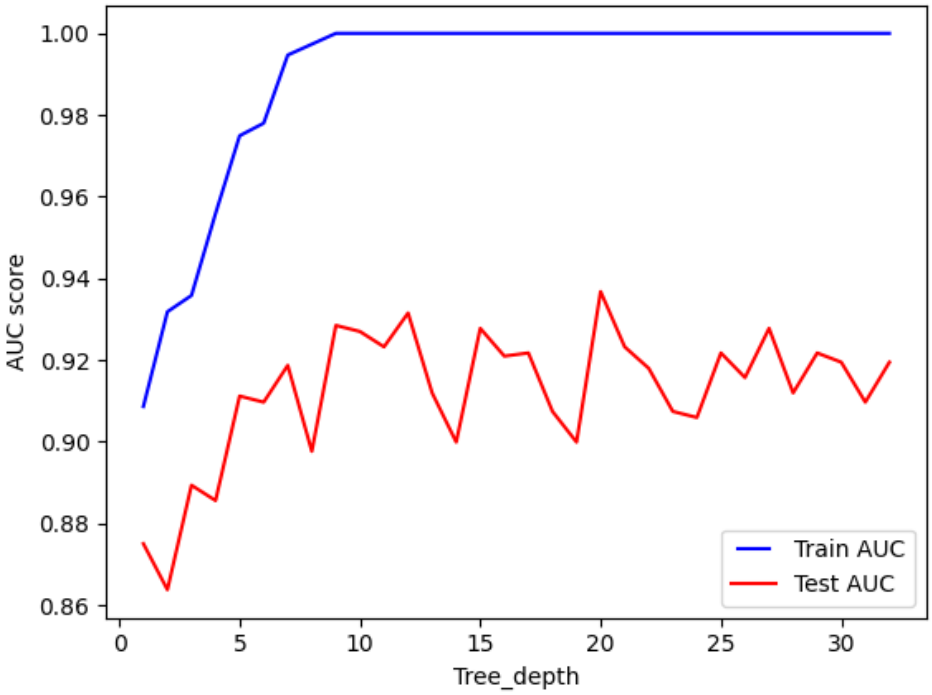


2. Maximum depth of decision trees

The same thing happens to the maximum depth of the random forest. It's not the deeper the better. Actually, after a certain boundary, increasing the depth does not have any stable effect on the test AUC. The model overfits data as the maximum depth grows and the train AUC reaches its high at about maximum depth = 10. While the test AUC failed to show such consistency as the train AUC does.

Here we have the code of maximum depth tuning.

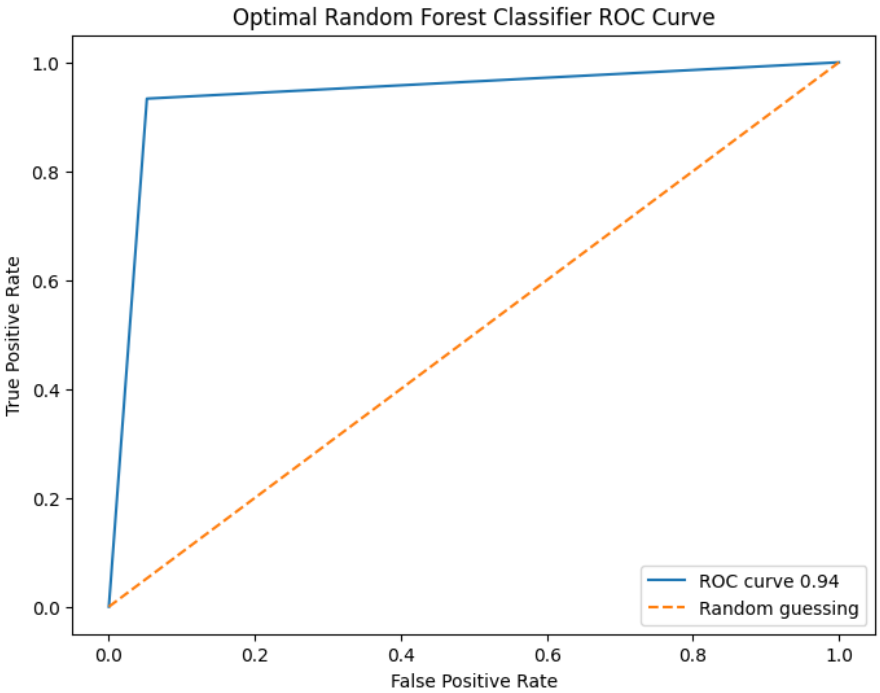
```
# max_depth
max_depths = np.linspace(1, 32, 32, endpoint=True)
train_results = []
test_results = []

for depth in max_depths:
  rf = RandomForestClassifier(n_estimators=optimal_n_estimator, max_depth=int(depth), n_jobs=-1)
  rf.fit(X_train, y_train)
  train_pred = rf.predict(X_train)
  roc_auc_rf_tuning = roc_auc_score(y_train, train_pred)
  train_results.append(roc_auc_rf_tuning)

  y_pred_rf_tuning = rf.predict(X_test)
  roc_auc_rf_tuning = roc_auc_score(y_test, y_pred_rf_tuning)
  test_results.append(roc_auc_rf_tuning)

line1, = plt.plot(max_depths, train_results, 'b', label="Train AUC")
line2, = plt.plot(max_depths, test_results, 'r', label="Test AUC")
plt.legend(handler_map={line1: HandlerLine2D(numpoints=2)})
plt.ylabel('AUC score')
plt.xlabel('Tree_depth')
plt.show()
```



Fine-tuned optimal Random Forest Classifier

After pruning the decision trees in our random forest classifier. Multiple evaluation scores are better than the default classifier. With the AUC of 0.94, MSE of 0.06 and variance score of 0.94.

```
rf_optimal = RandomForestClassifier(n_estimators=optimal_n_estimator, max_depth=optimal_max_depth, n_jobs=-1)
rf_optimal.fit(X_train, y_train)
y_pred_rf_optimal = rf_optimal.predict(X_test)
fpr_rf, tpr_rf, _ = roc_curve(y_test, y_pred_rf_optimal)
roc_auc_rf = metrics.roc_auc_score(y_test, y_pred_rf_optimal)
plt.figure(figsize=(8, 6))
plt.plot(fpr_rf, tpr_rf, label="ROC curve %0.2f"% roc_auc_rf)
plt.plot([0, 1], [0, 1], linestyle="--", label="Random guessing")
plt.title("Optimal Random Forest Classifier ROC Curve")
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.legend()
plt.show()
```

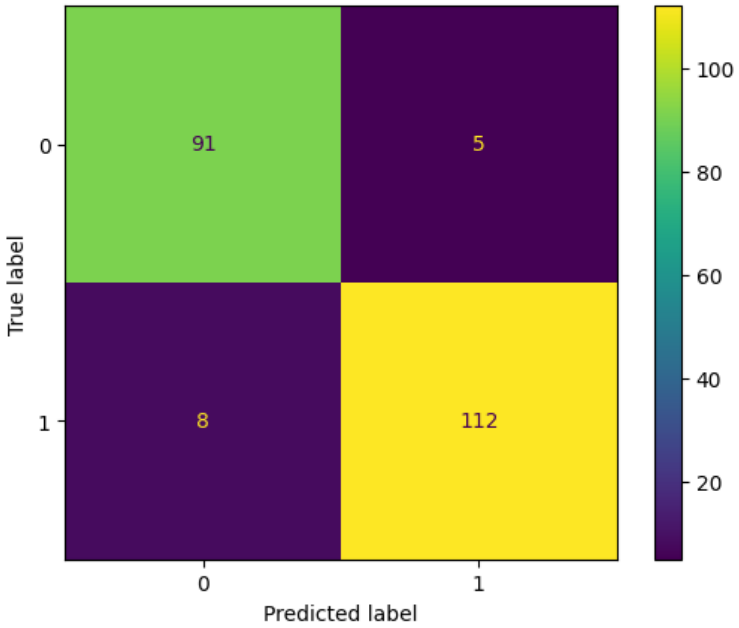


The evaluation code and output.

```
confusion_matrix_rf_optimal = confusion_matrix(y_test, y_pred_rf_optimal)
print("Accuracy of optimal random forest classifier: ", accuracy_score(y_test, y_pred_rf_optimal))
print("Mean squared error (MSE) of optimal random forest classifier: ", mean_squared_error(y_test, y_pred_rf_optimal))
print("Variance score of optimal random forest classifier: ", rf_optimal.score(X_test, y_test))
print("Precision score of optimal random forest classifier: ", precision_score(y_test, y_pred_rf_optimal))
print("Recall value of optimal random forest classifier: ", recall_score(y_test, y_pred_rf_optimal))
print("AUC of optimal random forest classifier: ", roc_auc_rf)
print("Confusion matrix of the optimal random forest classifier: \n", confusion_matrix_rf_optimal)
```

```
Accuracy of optimal random forest classifier:  0.9398148148148148
Mean squared error (MSE) of optimal random forest classifier:  0.06018518518518518
Variance score of optimal random forest classifier:  0.9398148148148148
Precision score of optimal random forest classifier:  0.9572649572649573
Recall value of optimal random forest classifier:  0.9333333333333333
AUC of optimal random forest classifier:  0.940625
Confusion matrix of the optimal random forest classifier:
 [[ 91   5]
 [  8 112]]
```

Visualization of the confusion matrix.

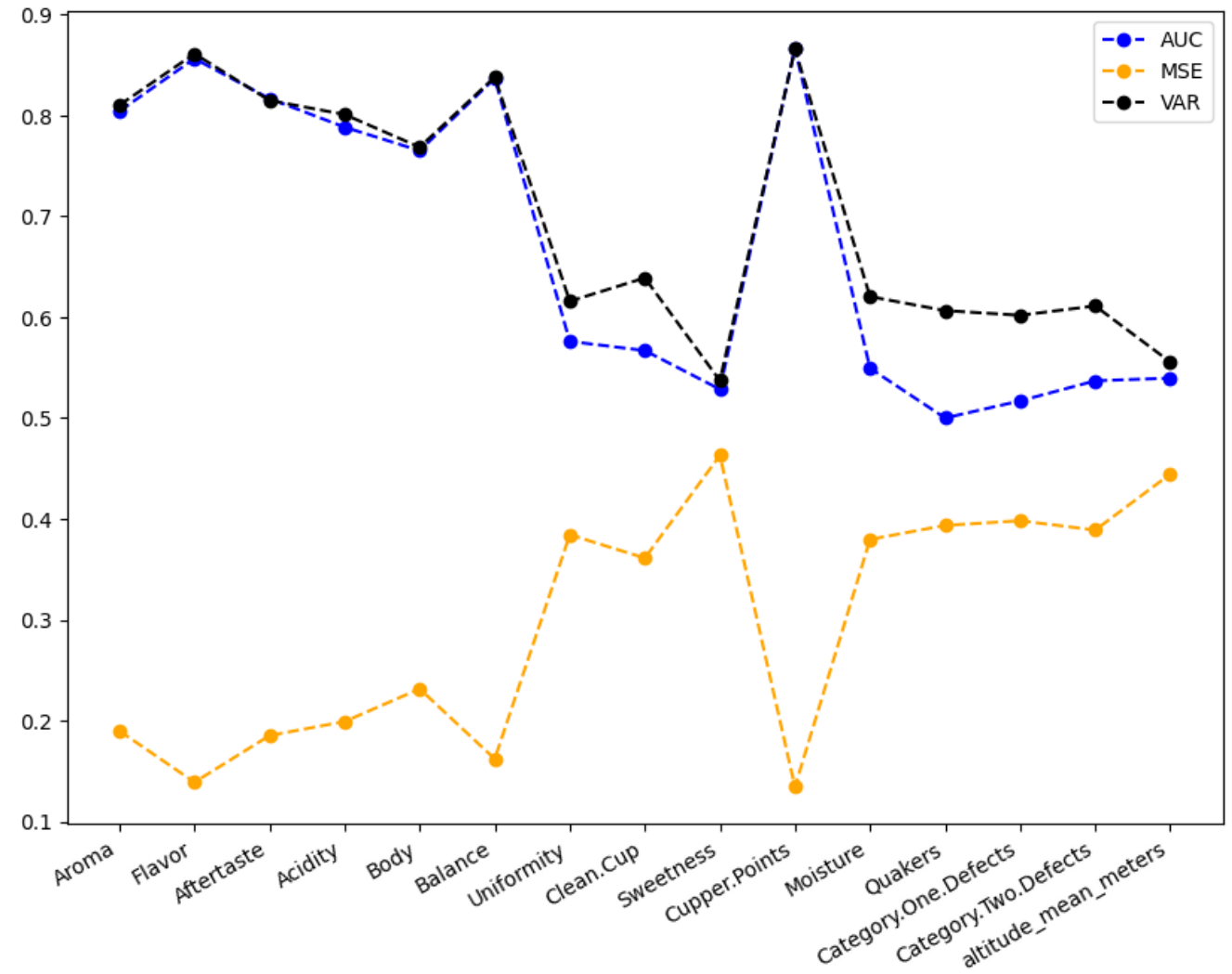


Classification Based on Each Valid Feature

Utilizing the fine-tuned classifier, I perform the classification task on each valid feature and get the AUC of each feature on the following plot. Along with the code to calculate the AUC for each feature.

```
AUC_features_rf = []
for i in range(15):
    X_1 = pd.DataFrame(df[col_names[i]])
    X_train_1, X_test_1, y_train_1, y_test_1 = train_test_split(X_1, binary_y, test_size=0.2)
    rf_optimal.fit(X_train_1, y_train_1)
    y_pred_rf_1 =rf_optimal.predict(X_test_1)

    print(str(i+1) + '. Feature: ' + col_names[i])
    print("Mean squared error: %.2f" % mean_squared_error(y_test_1, y_pred_rf_1))
    print('Variance score: %.2f' % rf_optimal.score(X_test_1,y_test_1))

    plt.scatter(X_train_1, y_train_1)
    plt.scatter(X_train_1, rf_optimal.predict_proba(X_train_1)[:,1], color="green", marker="+")
    plt.xlabel(col_names[i])
    plt.ylabel("Probability")
    plt.show()

    fpr_1, tpr_1, _ = roc_curve(y_test_1, y_pred_rf_1, pos_label=None, sample_weight=None, drop_intermediate=True)
    roc_auc_1 = metrics.roc_auc_score(y_test_1, y_pred_rf_1)
    AUC_features_rf.append(roc_auc_1)
    plt.figure(figsize=(8, 6))
    plt.plot(fpr_1, tpr_1, label=col_names[i] + " ROC curve %0.2f"% roc_auc_1)
    plt.plot([0, 1], [0, 1], linestyle="--", label="Random guessing")
    plt.title("Optimal Random Forest Classification ROC Curve For " + col_names[i])
    plt.xlabel("False Positive Rate")
    plt.ylabel("True Positive Rate")
    plt.legend()
    plt.show()
```

```
# Plot AUC based on each feature
fig, ax = plt.subplots(figsize=(10, 8))
plt.plot(feature_names, AUC_features_rf, color='blue', marker='o', linestyle='--', label="AUC")
plt.plot(feature_names, MSE_features_rf, color='orange', marker='o', linestyle='--', label="MSE")
plt.plot(feature_names, VAR_features_rf, color='black', marker='o', linestyle='--', label="VAR")

plt.legend()
fig.autofmt_xdate()
plt.show()
```



From the plot of AUC, MSE, and variance score of each feature. We could get to a conclusion about which features affect the most on our classification result. From the plot, “Aroma”, “Flavor”, “Balance” and “Cupper.Points” have the highest AUC and VAR and the least MSE. Thus, these features are the most contributors to our classification output. On the other hand, “Uniformity”, “Sweetness”, and “altitude_mean_meters” have the least AUC and VAR and the highest MSE of the features. They are not really relevant to our classification result. We could observe a similarity from other models like the Linear Regression model, Logistic Regression model, and KNN Classification model as well.

E. K Nearest Neighbour Classification

Introduction

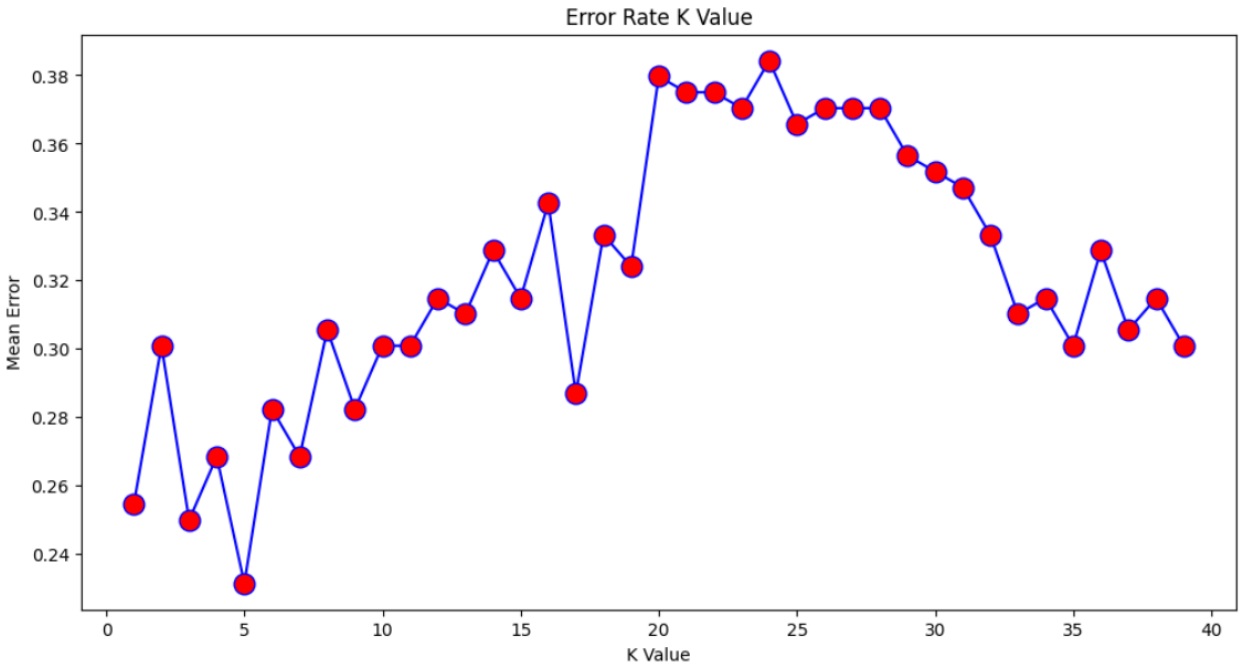
The quality of coffee is a crucial factor that impacts its taste, sweetness, flavor, and aroma, given its popularity as a globally-consumed beverage. One way to assess coffee quality is by utilizing a K Nearest Neighbour Classification model - a supervised machine learning algorithm that is commonly employed for classification tasks.

By training the KNN algorithm on a dataset of coffee samples that are rated based on quality, it can then be used to forecast the quality of new, unseen coffee samples based on their attributes, such as aroma, flavor, acidity, and body. This approach is beneficial for coffee growers, roasters, and tasters alike as it enables them to gain a better understanding of the factors that contribute to coffee quality and make more informed decisions about how to enhance it. This article will provide a step-by-step guide on how to use KNN to evaluate coffee quality.

Model Training and Evaluation

To apply KNN classification, first we need to split the data into two sets: a training set and a testing set. The training set is used to train the KNN classifier, while the testing set is used to evaluate the performance of the classifier. As we have already separated the data into two sets in the previous step. Then, we need to determine the optimal K value, which represents the number of nearest neighbors that will be considered when classifying new instances. In our case, we fit the KNeighborsClassifier into models with different K values (from 1 to 40), in terms of calculating the error rate and finding the K value with the minimum error rate.

```
# Calculating error for K values between 1 and 40

errorRate = []

for i in range(1, 40):
    knn = KNeighborsClassifier(n_neighbors=i)
    knn.fit(X_train, y_train)
    pred_i = knn.predict(X_test)
    errorRate.append(np.mean(pred_i != y_test))


plt.figure(figsize=(12, 6))
plt.plot(range(1, 40), errorRate, color='blue', marker='o',
         markerfacecolor='red', markersize = 12)
plt.title('Error Rate K Value')
plt.xlabel('K Value')
plt.ylabel('Mean Error')
```



From the line chart above, we can tell that the optimal K value is 5, since it has the lowest error rate compared to other K values. After finding the optimal K value, we fit the KNeighborsClassifier with 5 as K value into our model using all 15 features to predict the target variable, we reported the model test score, model training score, coefficients, mean squared error, and variance score for the model on the test set. The results show that the model had a mean squared error of 0.23 and a variance score of 0.77.

```
knn = KNeighborsClassifier(n_neighbors = 5)
knn.fit(X_train, y_train)
y_pred_knn = knn.predict(X_test)


print('Model test Score: %.3f, ' %knn.score(X_test, y_test),
      'Model training Score: %.3f' %knn.score(X_train, y_train))
print("Mean squared error: %.2f" % mean_squared_error(y_test, y_pred_knn))
print("Variance score: %.2f" % knn.score(X_test , y_test))
```

```
Model test Score: 0.769,  Model training Score: 0.830
Mean squared error: 0.23
Variance score: 0.77
```

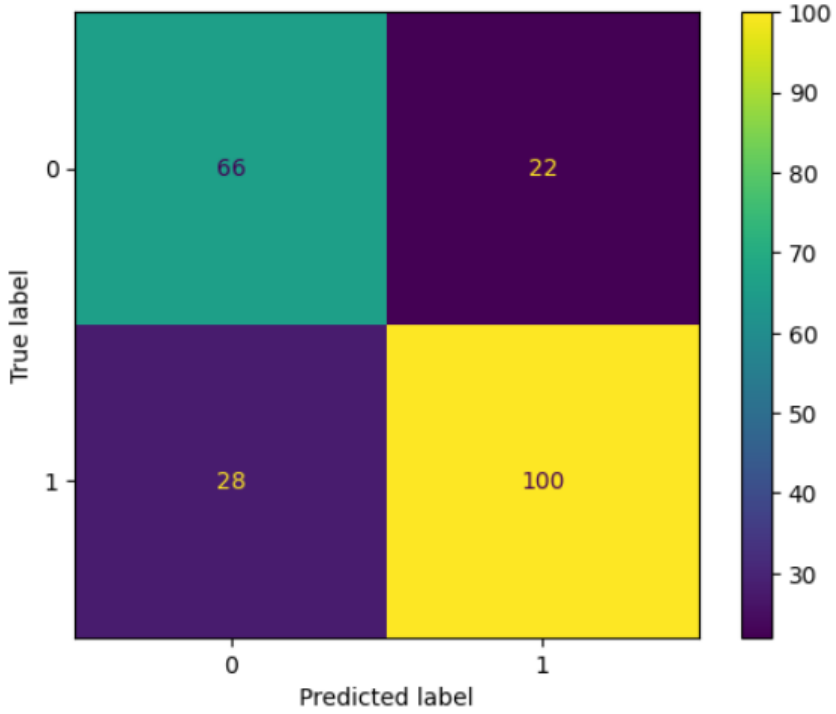
Then we calculate the accuracy of the classifier, which is 0.7685. It means that it correctly classified 76.85% of the instances in the testing set. This is a reasonable accuracy, but it can be improved with the further tuning of the algorithm. And we receive a good precision score of 0.8197, which means that when it does prediction, it is correct 81.97% of the time, as it indicates that the classifier has a low false positive rate. Moreover, the recall of the classifier is 0.7813, which means that it correctly identifies 78.13% of the good quality beans, as it indicates that the classifier has a low false negative rate.

We also provide a classification report and a confusion matrix. The report has a detailed evaluation of the performance of the classifier for each class. The precision, recall, and F1-score are reported for each class, as well as the support (number of instances) for each class, while the confusion matrix provides additional insights into the model's performance. It shows that the model correctly classified 66 instances as true negatives and 100 instances as true positives. However, it also misclassified 22 instances of class 0 as false negatives and 28 instances of class 1 as false positives.

```
print('Accuracy : ', accuracy_score(y_test, y_pred_knn))
print('Precision : ', precision_score(y_test, y_pred_knn))
print('Recall:  ', recall_score(y_test, y_pred_knn))
print('Classification Report :\n', classification_report(y_test, y_pred_knn))
print('Confusion Matrix :\n', confusion_matrix(y_test, y_pred_knn))
```

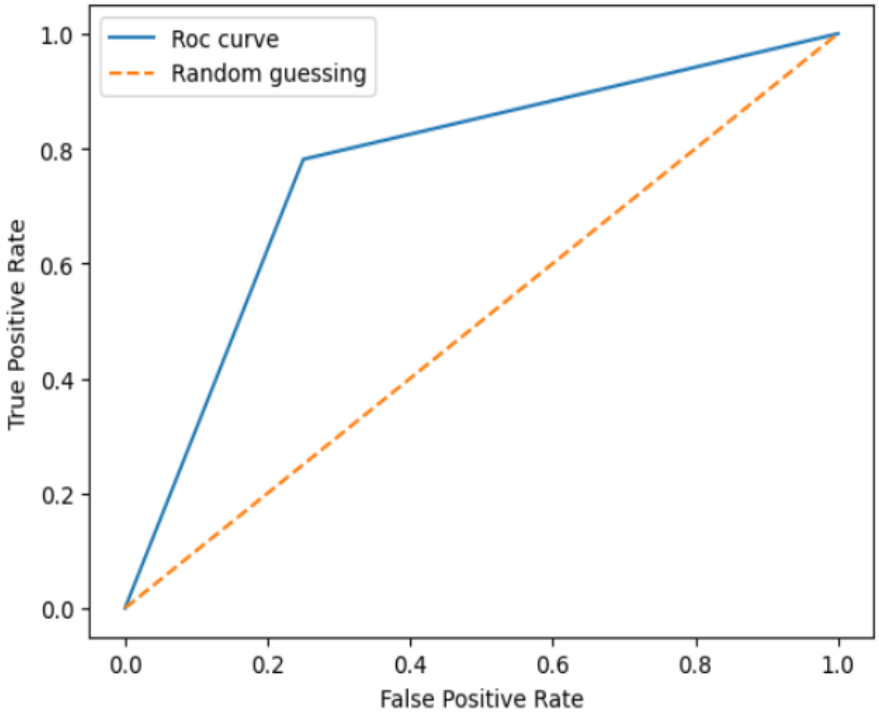
```
Classification Report :
              precision    recall  f1-score   support

           0       0.70      0.75      0.73        88
           1       0.82      0.78      0.80       128

    accuracy                           0.77       216
   macro avg       0.76      0.77      0.76       216
weighted avg       0.77      0.77      0.77       216

Confusion Matrix :
 [[ 66  22]
 [ 28 100]]
```

```
[13] cfm = metrics.ConfusionMatrixDisplay(confusion_matrix(y_test, y_pred_knn), display_labels=None)
     cfm.plot()
     plt.show()
```



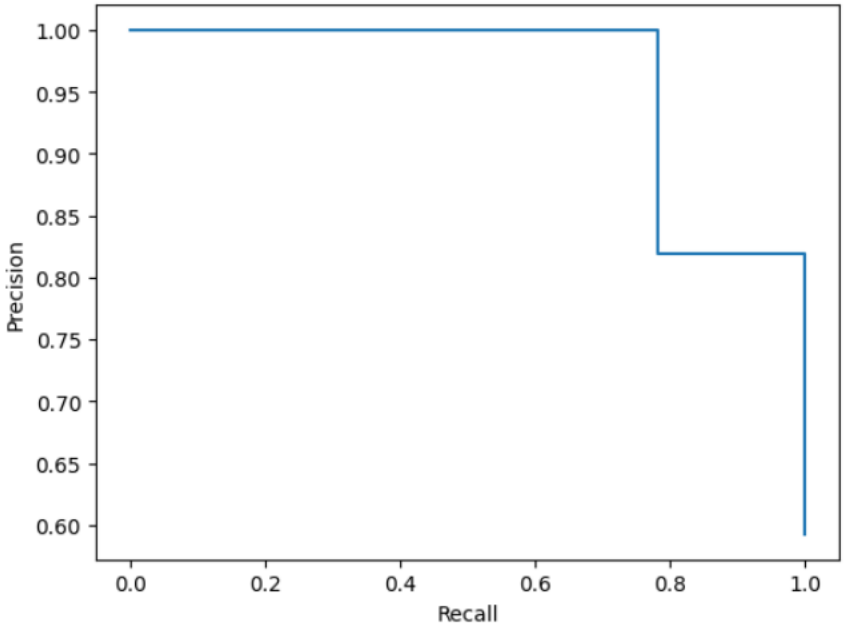
Furthermore, we plot the ROC curve graph to measure the trade-off between TPR and FPR, and the Precision/Recall curve graph to measure the trade-off between precision and recall. From ROC curve, we receive an AUC (Area Under the Curve) value of 0.7656, which indicates that the model is performing better than random guessing.

Roc curve:



AUC: 0.765625

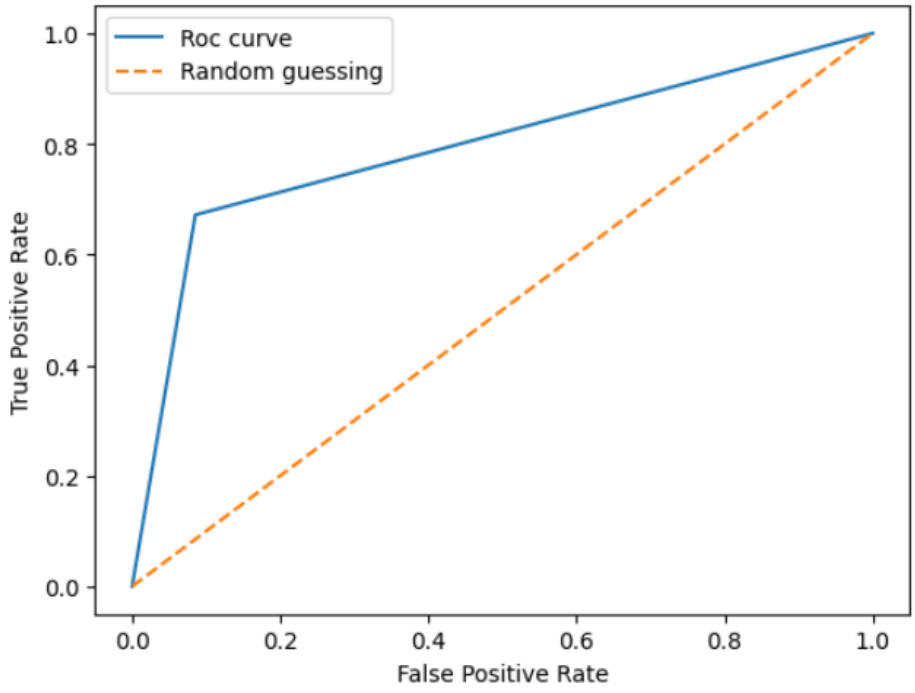
Precision/Recall curve:



In order to explore in depth the factors that affect the quality of coffee, we then plot the ROC curve for 15 features separately, and collect AUC value for all 15 features, in terms to find out which feature has the biggest impact on the prediction.

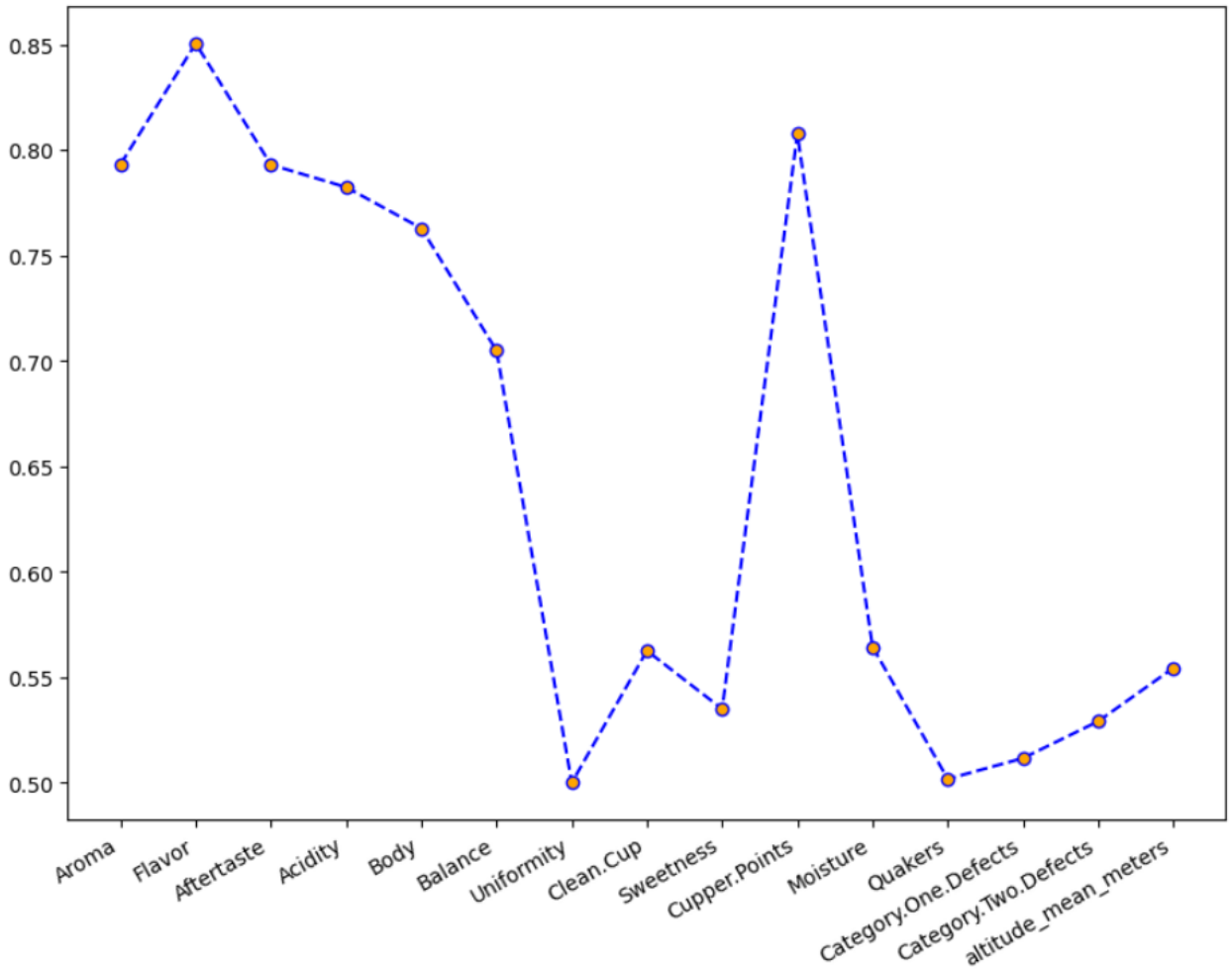
```
AUC = []

for i in range(15):
  Xi = pd.DataFrame(df[col_names[i]])
  Xi_train, Xi_test, yi_train, yi_test = train_test_split(Xi, binary_y, test_size=0.2)
  knn = KNeighborsClassifier(n_neighbors = 4)
  knn.fit(Xi_train, yi_train)
  yi_pred_knn = knn.predict(Xi_test)
  yi_train_pred = knn.predict(Xi_train)
  print("Feature: " + col_names[i])
  print('Model test Score: %.3f, ' %knn.score(Xi_test, yi_test),
      'Model training Score: %.3f' %knn.score(Xi_train, yi_train))
  print("Mean squared error: %.2f" % mean_squared_error(yi_test, yi_pred_knn))
  print("Variance score: %.2f" % knn.score(Xi_test , yi_test))
  print('Accuracy : ', accuracy_score(yi_test, yi_pred_knn))
  print("AUC: ", roc_auc)
  fpr, tpr, thr1 = roc_curve(yi_test, yi_pred_knn)
  pre, rec, thr2 = precision_recall_curve(yi_test, yi_pred_knn)
  roc_auc = metrics.roc_auc_score(yi_test, yi_pred_knn)
  AUC.append(roc_auc)
  print("Roc curve: ")
  plt.plot(fpr, tpr, label="Roc curve")
  plt.plot([0, 1], [0, 1], linestyle="--", label="Random guessing")
  plt.legend()
  plt.xlabel('False Positive Rate')
  plt.ylabel('True Positive Rate')
  plt.show()
```

```
Feature: Aroma
Model test Score: 0.764,  Model training Score: 0.769
Mean squared error: 0.24
Variance score: 0.76
Accuracy :  0.7638888888888888
AUC:  0.765625
Roc curve:
```



At the end, we plot the AUC graph for 15 features to visualize the data.

```
# plot AUC

fig, ax = plt.subplots(figsize=(10, 8))
ax.plot_date(feature_names, AUC, color='blue', marker='o', linestyle='--', markerfacecolor='orange')

fig.autofmt_xdate()
plt.show()
```



As we can see, the Flavor feature has the most impact on prediction, and the Uniformity has the lowest impact.

IV. Models Comparison Results

In this part, we present a detailed analysis of the performance of the four prediction models we implemented in this research: Linear Regression, Logistic Regression, KNN, and Random Forest. We ended up collecting and summarizing the performance data of all models, filling up the following table.

	Linear Regression	Logistic Regression	KNN	Random Forest
Accuracy		0.888	0.769	0.940
MSE	0.0005	0.11	0.23	0.06
Variance Score	0.9999	0.89	0.77	0.940
Precision		0.85	0.820	0.957
Recall		0.975	0.781	0.933
AUC		0.88	0.766	0.941

From the results collected above, we can see that the Linear Regression model has the highest variance Score 0.999 and the lowest MSE 0.0005. The error of the Linear Regression model is relatively lower than the others models. Besides, the Random Forest Model has a 0.940 accuracy, 0.957 precision score, 0.933 recall value, and 0.941 AUC (Area under ROC curve), which also performed excellently among the four models. However, Logistic Regression Model and K Nearest Neighbors Model didn't do great performance in our research. The Logistic Regression model has a 0.888 accuracy, 0.85 precision score, 0.975 recall value, and 0.88 AUC. KNN model only has 0.769 accuracies, with a 0.82 precision score, 0.781 recall value, and 0.766 AUC.

At the end, we could come with a conclusion that the relation between feature and class in our dataset is more linearly correlated. And thus it leads to the best performance from the linear regression model.

V. Future Work

To enhance the performance of the prediction and improve its effectiveness and applicability, we still have a lot to improve on in the future. First, we can incorporate additional features from other data sets or organizations, such as different types of coffee beans, and climate conditions. It could help improve the predictive power and performance of the models. What's more, we also plan to use more models for our research in the future. We could apply some more complicated models such as Support Vector Machines (SVM), and we could also implement some deep learning techniques such as Convolutional Neural Networks to see if they would produce better results. Besides, we could also add geographic analysis to our research. In our current research, we only used altitude as a basis for geographical location research. In future studies, we can add more features, such as country, terrain, and so on, to better analyze the influence of geographical location on coffee quality.

VI. Conclusion

In our study, we explored the potential of machine learning algorithms to predict Arabica coffee quality using data sets from the Coffee Quality Institute. We developed and compared four models: linear regression, logistic regression, K-nearest neighbor classification, and the random forest model. Based on the MSE, variance score, accuracy, precision score, recall value, and ROC AUC, our evaluation shows that linear regression and random forest models outperform other models.

In conclusion, our study highlights the effectiveness of linear regression and random forest models in predicting Arabica coffee quality. By improving these models, we can further improve the predictive power of machine learning in the coffee industry, so that consumers can more accurately select their favorite flavor of coffee.

VII. Reference

International Coffee Organization. (2021). Coffee Market Report - January 2021. Retrieved from https://www.ico.org/documents/cy2020-21/cmr-0121-e.pdf

VIII. Annex A

Jinyin Chai — Logistic Regression

Qiyang Xie — Random Forest

Ying Shen — Linear Regression

Zifei Song — K Nearest Neighbours Classification