Final Term Project (FTP) Prepared by: Taeesh Azal Assadi Date: 12/08/2023

Project details

Project Name	Due Date	Course number	Course Title
WINE VARIETY PREDICTION	**12/08/2023**	**CS5805**	**Machine Learning- 1**

Contents

S.N	Title
1.	Abstract
2.	Introduction
3.	Description of the Dataset
4.	Phases a. Phase I - Exploratory Data Analysis (EDA) b. Phase II - Regression Analysis c. Phase III - Classification Analysis d. Phase IV - Classification Analysis
5.	Recommendation
6.	Appendix
7.	Reference

Table of figures

Figure 1: Displaying the attributes in the dataset.
Figure 2: Boxplot of price and point dropping the missing values
Figure 3:Vectorization performed on ‘description’ feature.
Figure 4:Results after PCA Analysis and Elbow plot
Figure 5: T-test analysis
Figure 6: OLS Regression results
Figure 7: Confidence Interval and stepwise regression and adjusted R square Analysis
Figure 8: Performance of Decision Tree Classifier
Figure 9: Performance of Logistic Regression
Figure 10: Performance of KNN model
Figure 11: Performance of SVM model
Figure 12: Performance of Naive Bayes model
Figure 13: Performance of Random Forest model
Figure 14: Performance of Neural Network model
Figure 15: Independent Research

1. Abstract

This study utilizes the Wine Reviews dataset, focusing on wines from the United States. It aims to explore feature engineering, classification, and regression analysis techniques. The dataset is analyzed for patterns, correlations, and trends, with specific attention to wine varieties as the primary target variable. Various machine learning models, including Logistic Regression, Decision Tree, KNN, SVM, and Neural Networks, are applied to classify wine varieties based on their characteristics. The project also delves into regression analysis to predict continuous numerical features and concludes with recommendations based on the classifiers' performance.

2. Introduction

In this project, I am embarking on a fascinating journey to blend the art of wine tasting with the science of machine learning. The essence of this endeavor is to create a predictive model that mimics the expertise of a master sommelier, identifying wines not by taste, but through their descriptions.

The cornerstone of this venture is the detailed examination of the 'Wine Reviews' dataset from Kaggle. This rich dataset, originally curated by Wine Enthusiast, features over 130,000 reviews of wines from around the world, with a notable emphasis on wines from the United States. It provides an extensive range of data points, including the country of origin, variety of the wine, the winery it originated from, along with tasting notes and ratings.

The primary objective here is to leverage deep learning techniques to predict wine varieties based on the narrative in the reviews. This model, while not capable of physically tasting the wine, aspires to accurately identify the variety of a wine based on a description, similar to what a sommelier might provide. This method represents a novel intersection of sensory expertise and algorithmic analysis, expanding the potential applications of artificial intelligence in the culinary domain.

The methodology of this project will encompass several critical stages:

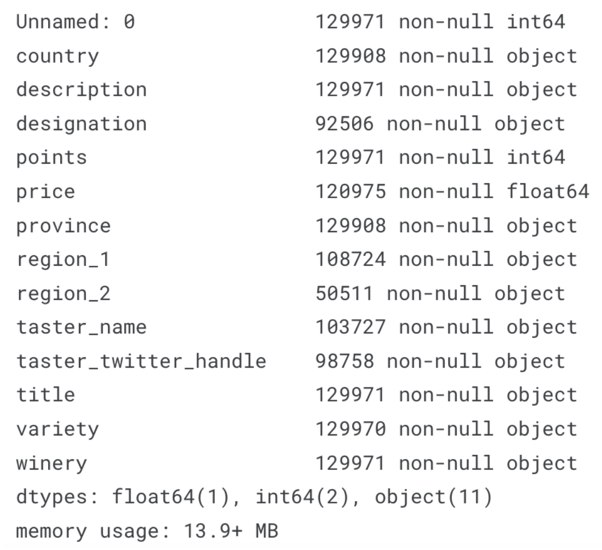
1. Conducting feature engineering and exploratory data analysis (EDA) to thoroughly understand the nuances of the dataset.
2. Performing regression analysis to make predictions about continuous numerical features, like a wine's rating and price, based on various factors.
3. Applying a range of classification models such as Logistic Regression, Decision Trees, KNN, SVM, Random Forest, and Neural Networks to classify wines based on their unique characteristics.
4. Delving into clustering and association rule mining for deeper insights.

This project is meticulously designed to utilize advanced data preprocessing techniques with the goal of identifying the most effective classifier for categorizing wines. My ambition is to not only deepen the comprehension of the wine dataset but also to provide valuable insights that could aid wine connoisseurs and enthusiasts in recognizing wines with the accuracy of a master sommelier.

3. Description of the Dataset

The dataset in focus is an extensive compilation of wine reviews, numbering approximately 130,000 entries. This dataset stands out due to its detailed coverage of wines predominantly from the United States, providing a rich source for analysis and application in the wine industry. It comprises critical attributes such as the country of origin, wine variety, winery details, identifiers of the tasters such as names and Twitter handles, province, specific regions within the producing country, and quantitative metrics like price and review points.

For the purpose of this analysis, the target variable is the 'variety' of the wine, which will be the outcome of our classification efforts. The independent variables—'price,' 'year,' 'points,' and 'description'—are the predictors that will be used to model and predict the wine variety. The 'year' variable indicates the vintage, which can significantly influence a wine's characteristics. 'Price' and 'points' provide a quantitative measure of a wine's value and quality as perceived in the market and by experts. The 'description' provides a textual insight into the sensory profile of the wine, which is a vital aspect of sommelier reviews.



```
Unnamed: 0               129971 non-null int64
country                  129908 non-null object
description              129971 non-null object
designation              92506 non-null object
points                   129971 non-null int64
price                    120975 non-null float64
province                 129908 non-null object
region_1                 108724 non-null object
region_2                 50511 non-null object
taster_name              103727 non-null object
taster_twitter_handle    98758 non-null object
title                    129971 non-null object
variety                  129970 non-null object
winery                   129971 non-null object
dtypes: float64(1), int64(2), object(11)
memory usage: 13.9+ MB
```

Figure 1: Displaying the attributes in the dataset.

The significance of this dataset in the industry is multifaceted. First, it offers a granular view of consumer and expert opinions, which can inform wineries about market trends and preferences. Second, the analysis of this dataset can assist retailers and distributors in inventory selection, pricing strategies, and marketing campaigns. Third, from a sommelier's perspective, it helps in understanding the correlation between descriptive language and wine varieties, enhancing the recommendation systems for end consumers. Lastly, it serves as a valuable educational tool for those studying oenology and viticulture, providing a database for research and study.

4. Phases

The project has four phases where in the first phase Exploratory Data Analysis is done. In this phase the data in the dataset is pre-processed before feeding it to our models. After preprocessing the data, we move into phase two which is regression analysis and then to phase three and finally to clustering and association. The details for every phase is explained in detail below under each subheading.

4.1 Phase I - Exploratory Data Analysis (EDA)

- Exploratory Data Analysis (EDA) is a crucial step in the project that involves comprehensively understanding the intricacies of the dataset. The EDA performed on the dataset of wine reviews includes several key steps to ensure data quality and prepare for further analysis.
- For Data preprocessing we removed NaN values which existed in features such as price, point, description, variety, etc.
- Insignificant and redundant feature such as- *'country','designation','taster_name','taster_twitter_handle', 'title','region_1','region_2'*
- *Observations were checked for duplicacy and duplicate entries were dropped.*
- Aggregation or down-sampling was not performed, as the dataset contains discrete data detailing individual wines, where maintaining the uniqueness of each entry is crucial.
- Outlier Analysis was performed and outliers in price , point and were removed

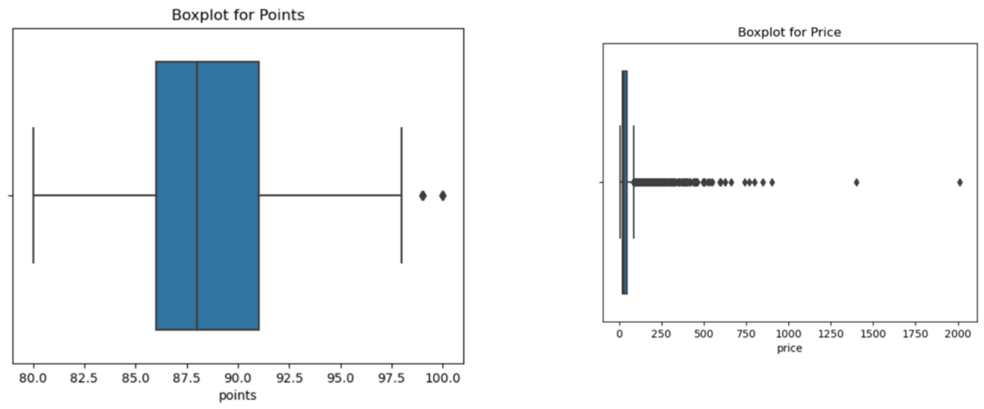
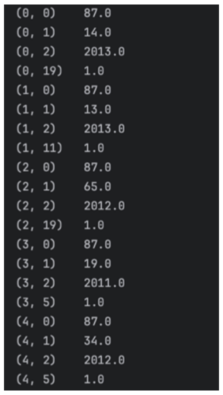


Figure 2: BOXPLOT of price and point dropping the missing values

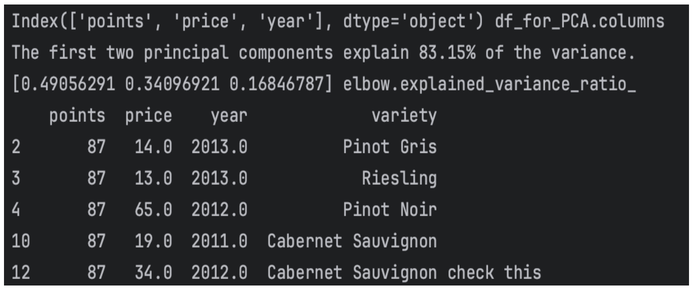
- Text vectorization on the wine descriptions was performed. This process converts the textual content into numerical data, enabling quantitative analysis and model training. We used the CountVectorizer method from the scikit-learn library, which tokenizes the text and counts the frequency of each token. In the dataset, the 'description' column contains a wealth of textual information that could potentially predict the wine variety. Through vectorization, we transformed this unstructured text into a structured, fixed-length set of features in the form of a matrix. Each row in the matrix corresponds to a wine description, and each column represents a unique word in the dataset's combined text.This matrix was then used as input for machine learning models. It is important to note that we filtered out English stop words to remove common but uninformative words that do not contribute to the predictive power of the model. Moreover, we also explored the possibility of using TF-IDF (Term Frequency-Inverse Document Frequency) vectorization as an alternative to CountVectorizer to account for the importance of each term in the corpus relative to its frequency across all documents.



```
(0, 0)    87.0
(0, 1)    14.0
(0, 2)    2013.0
(0, 19)   1.0
(1, 0)    87.0
(1, 1)    13.0
(1, 2)    2013.0
(1, 11)   1.0
(2, 0)    87.0
(2, 1)    65.0
(2, 2)    2012.0
(2, 19)   1.0
(3, 0)    87.0
(3, 1)    19.0
(3, 2)    2011.0
(3, 5)    1.0
(4, 0)    87.0
(4, 1)    34.0
(4, 2)    2012.0
(4, 5)    1.0
```

Figure 3: Vectorization performed on 'description' feature

- In our dataset, we applied one-hot encoding to the 'province' feature, a categorical variable representing the wine's region of production. This encoding process converts each unique province value into a separate binary column, allowing for the incorporation of this categorical data into machine learning models that require numerical input.
- After this PCA was done. The results from the analysis suggest that 83.15% variance is explained the 10 components. So none of the attributes were further dropped.



```
Index(['points', 'price', 'year'], dtype='object') df_for_PCA.columns
The first two principal components explain 83.15% of the variance.
[0.49056291 0.34096921 0.16846787] elbow.explained_variance_ratio_
    points  price    year              variety
2       87   14.0  2013.0           Pinot Gris
3       87   13.0  2013.0             Riesling
4       87   65.0  2012.0           Pinot Noir
10      87   19.0  2011.0   Cabernet Sauvignon
12      87   34.0  2012.0   Cabernet Sauvignon check this
```

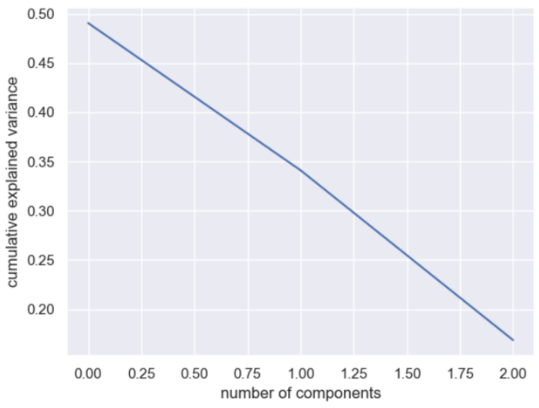
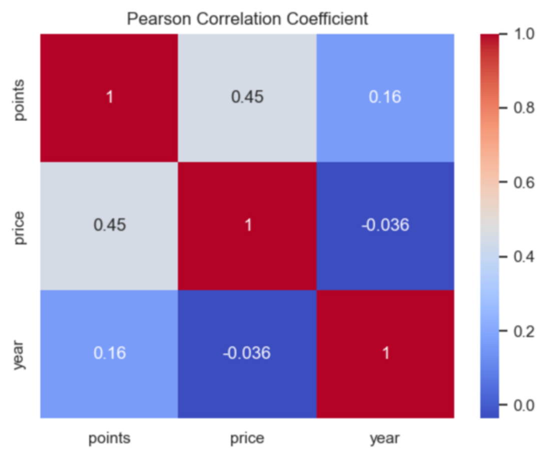
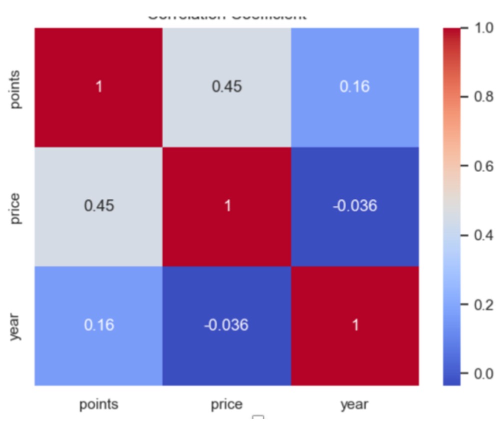


Figure 4: Results after PCA Analysis and Elbow plot





- A high covariance between two variables indicates that the variables are strongly correlated, while a low covariance indicates that the variables are weakly correlated or uncorrelated. The diagonal elements of the matrix represent the variances of the variables, and the off-diagonal elements represent the covariances between each pair of variables. A high variance for a variable indicates that the values of that variable are spread out, while a low variance indicates that the values are clustered around the mean

  Now we head to Regression Analysis after Feature Engineering and EDA

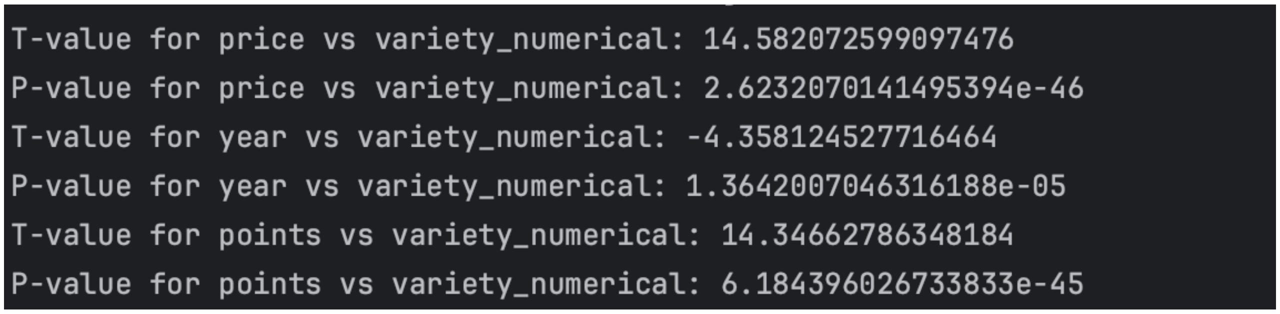
4.2 Phase II - Regression Analysis

In this phase Regression analysis is done and the results are shared in this report in the snapshot of the code.

a. T-test analysis

In the second phase of the project, regression analysis was conducted to identify trends within the data through various analytical methods, including T-test and Stepwise regression analyses.

The T-test is commonly utilized to assess the difference in means between two groups, which aids in establishing whether the observed disparity is statistically significant. A notable t-statistic value suggests a substantial divergence between group means, implying that the variance is likely to be significant rather than a result of random variability. Below is a visual representation of the T-test analysis outcomes.



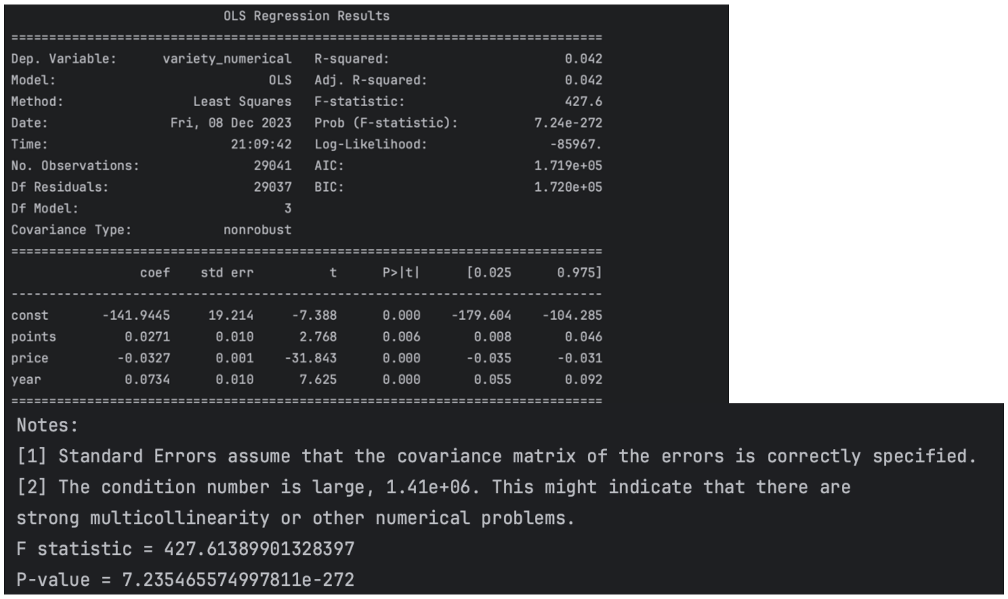
```
T-value for price vs variety_numerical: 14.582072599097476
P-value for price vs variety_numerical: 2.6232070141495394e-46
T-value for year vs variety_numerical: -4.358124527716464
P-value for year vs variety_numerical: 1.3642007046316188e-05
T-value for points vs variety_numerical: 14.34662786348184
P-value for points vs variety_numerical: 6.184396026733833e-45
```

Figure 5: T Test Analysis

From the analysis we know that all the features are real and meaningful.

b. OLS Regression Analysis

After this Stepwise regression analysis was performed. There are no attributes whose value is greater than 0.05, hence we consider all the features. The low value of R-squared is due to the classification data.

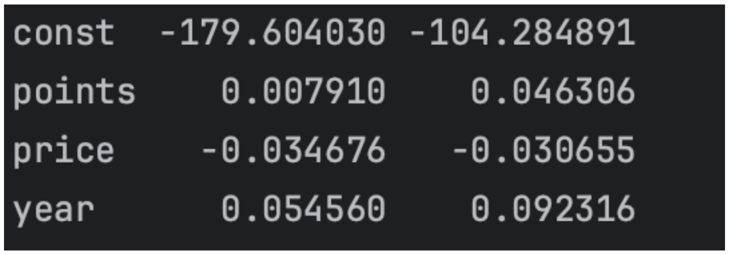


```
                            OLS Regression Results
==============================================================================
Dep. Variable:      variety_numerical   R-squared:                       0.042
Model:                            OLS   Adj. R-squared:                  0.042
Method:                 Least Squares   F-statistic:                     427.6
Date:                Fri, 08 Dec 2023   Prob (F-statistic):          7.24e-272
Time:                        21:09:42   Log-Likelihood:                -85967.
No. Observations:               29041   AIC:                         1.719e+05
Df Residuals:                   29037   BIC:                         1.720e+05
Df Model:                           3
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const       -141.9445     19.214     -7.388      0.000    -179.604    -104.285
points         0.0271      0.010      2.768      0.006       0.008       0.046
price         -0.0327      0.001    -31.843      0.000      -0.035      -0.031
year           0.0734      0.010      7.625      0.000       0.055       0.092
==============================================================================
Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 1.41e+06. This might indicate that there are
strong multicollinearity or other numerical problems.
F statistic = 427.61389901328397
P-value = 7.235465574997811e-272
```

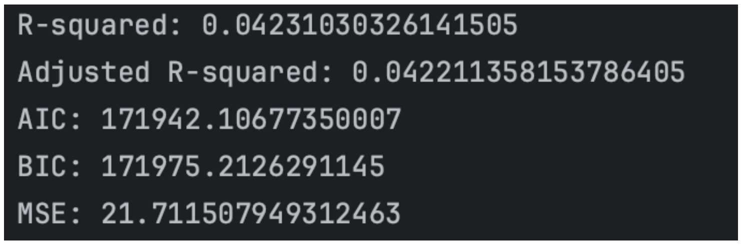
Figure 6: OLS Regression results

C. Confidence Interval Analysis, stepwise regression and Adjusted R square Analysis

- The confidence intervals for the OLS regression coefficients are fairly tight, suggesting precision in the estimates. The constant term, points, and year show positive influence, with the year having a relatively wider interval, indicating more uncertainty in its exact effect. Price, on the other hand, has a negative impact on the dependent variable, with a narrow interval indicating high precision in this estimate. Overall, the confidence intervals suggest statistically significant results, but the practical significance would depend on the context and the magnitude of these effects relative to the domain of wine variety classification.



```
const   -179.604030 -104.284891
points     0.007910    0.046306
price     -0.034676   -0.030655
year       0.054560    0.092316
```



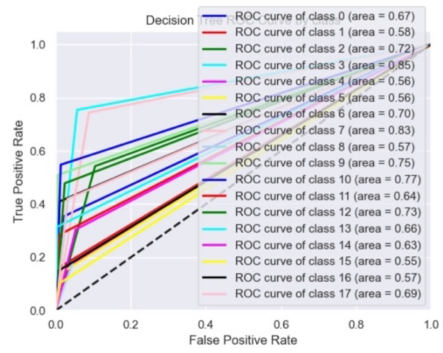
```
R-squared: 0.04231030326141505
Adjusted R-squared: 0.042211358153786405
AIC: 171942.10677350007
BIC: 171975.2126291145
MSE: 21.711507949312463
```

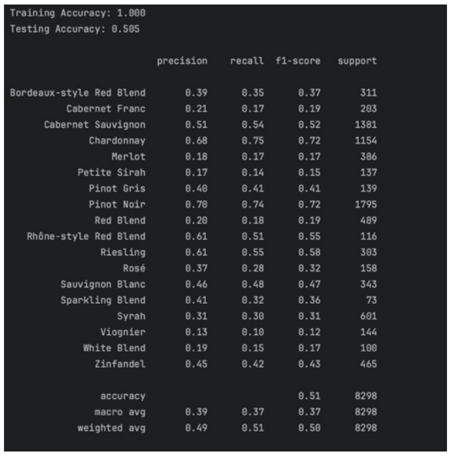
Figure 7: Confidence Interval and stepwise regression and adjusted R square Analysis

4.3 Phase III- Classification Analysis

During this stage of the project, a suite of machine learning models was employed to classify the varieties of wine within the dataset. The ensemble of classifiers tested includes Decision Trees, Logistic Regression, K-Nearest Neighbors (KNN), Support Vector Machines (SVM), Naïve Bayes, Random Forests, and Neural Networks. The rationale for using a diverse range of classifiers is to determine the most effective model based on performance metrics. For a comprehensive evaluation, metrics such as the Confusion Matrix, Precision, Sensitivity, Specificity, F-score, and ROC curve were computed for each classifier. These metrics were then showcased for each model, culminating in a comparative analysis to discern the most superior classifier. The report starts by detailing individual classifier performance followed by a juxtaposition of all the models at the conclusion.

4.3.1 Decision tree



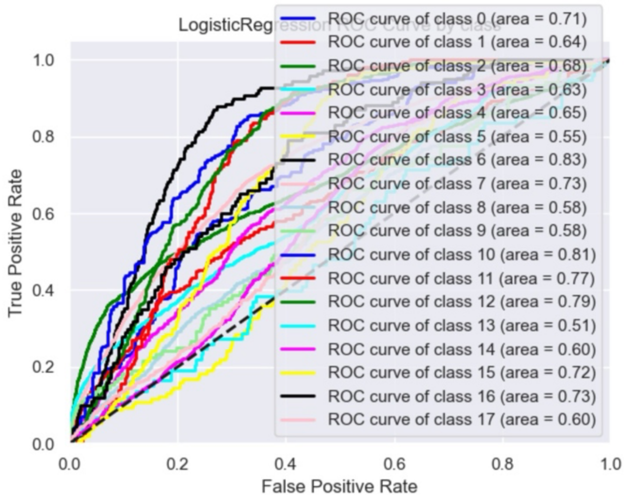


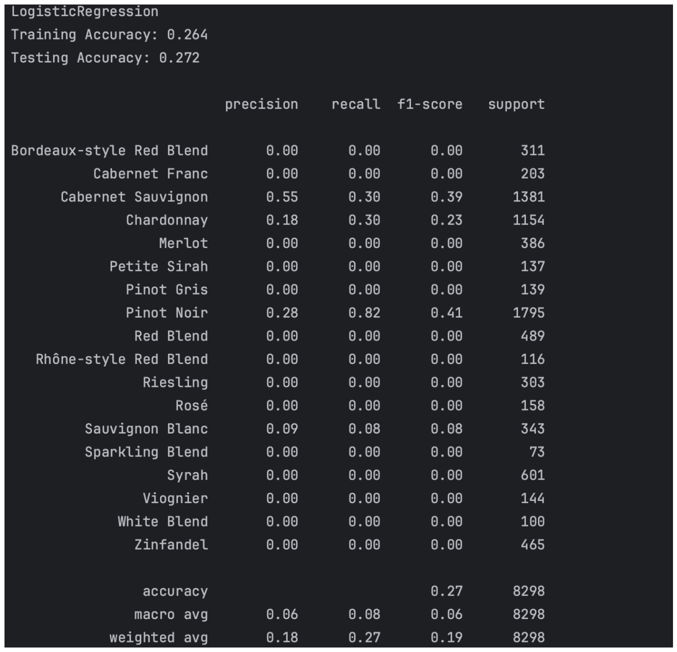
```
Training Accuracy: 1.000
Testing Accuracy: 0.505

                          precision    recall  f1-score   support

Bordeaux-style Red Blend       0.39      0.35      0.37       311
         Cabernet Franc       0.21      0.17      0.19       203
     Cabernet Sauvignon       0.51      0.54      0.52      1381
             Chardonnay       0.68      0.75      0.72      1154
                 Merlot       0.18      0.17      0.17       386
           Petite Sirah       0.17      0.14      0.15       137
             Pinot Gris       0.40      0.41      0.41       139
             Pinot Noir       0.70      0.74      0.72      1795
              Red Blend       0.20      0.18      0.19       489
  Rhône-style Red Blend       0.61      0.51      0.55       116
               Riesling       0.61      0.55      0.58       303
                   Rosé       0.37      0.28      0.32       158
        Sauvignon Blanc       0.46      0.48      0.47       343
         Sparkling Blend       0.41      0.32      0.36        73
                  Syrah       0.31      0.30      0.31       601
              Viognier       0.13      0.10      0.12       144
            White Blend       0.19      0.15      0.17       100
              Zinfandel       0.45      0.42      0.43       465

               accuracy                           0.51      8298
              macro avg       0.39      0.37      0.37      8298
           weighted avg       0.49      0.51      0.50      8298
```

Figure 8: Performance of Decision Tree Classifier

The testing accuracy is found to be 50.5% while the training accuracy is 100% which shows overfitting of data. The area under the curve is 0.5 which suggests that the model is accurately able to distinguish between two classes 50% of time. This model is promising however, there is overfitting of data which can be resolved by tuning of hyperparameter. After hyperparameter tuning it was found to be 60.78%

4.3.2 Logistic Regression



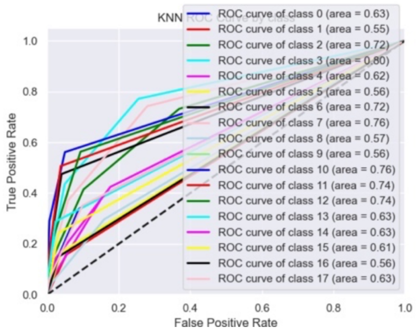


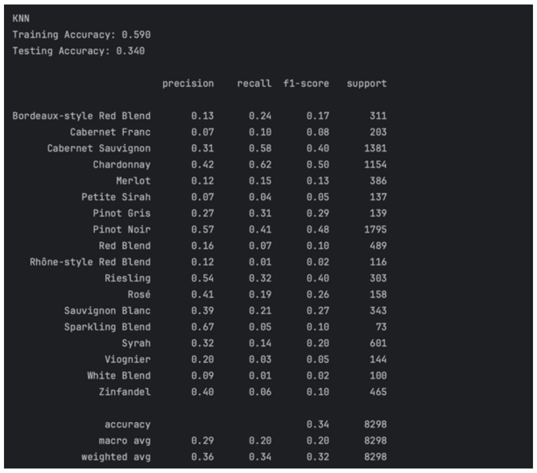
```
LogisticRegression
Training Accuracy: 0.264
Testing Accuracy: 0.272

                          precision    recall  f1-score   support

Bordeaux-style Red Blend       0.00      0.00      0.00       311
         Cabernet Franc       0.00      0.00      0.00       203
     Cabernet Sauvignon       0.55      0.30      0.39      1381
             Chardonnay       0.18      0.30      0.23      1154
                 Merlot       0.00      0.00      0.00       386
           Petite Sirah       0.00      0.00      0.00       137
             Pinot Gris       0.00      0.00      0.00       139
             Pinot Noir       0.28      0.82      0.41      1795
              Red Blend       0.00      0.00      0.00       489
  Rhône-style Red Blend       0.00      0.00      0.00       116
               Riesling       0.00      0.00      0.00       303
                   Rosé       0.00      0.00      0.00       158
        Sauvignon Blanc       0.09      0.08      0.08       343
         Sparkling Blend       0.00      0.00      0.00        73
                  Syrah       0.00      0.00      0.00       601
              Viognier       0.00      0.00      0.00       144
            White Blend       0.00      0.00      0.00       100
              Zinfandel       0.00      0.00      0.00       465

               accuracy                           0.27      8298
              macro avg       0.06      0.08      0.06      8298
           weighted avg       0.18      0.27      0.19      8298
```

Figure 9: Performance of Logistic Regression

The testing accuracy is 27%. This model shows poor performance because of low accuracy and AUC in the ROC curve.

4.3.3 K-nearest neighbor (KNN)



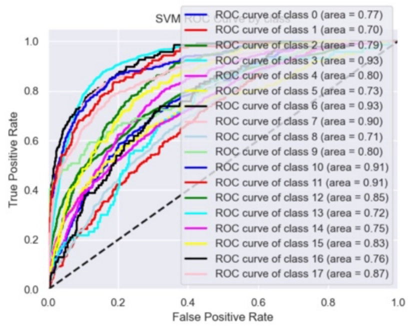


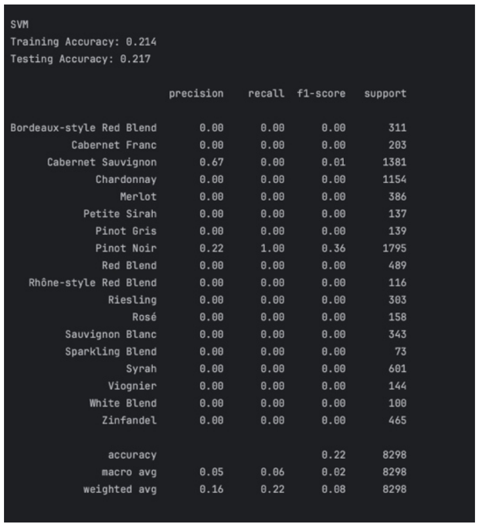
```
KNN
Training Accuracy: 0.590
Testing Accuracy: 0.340

                          precision    recall  f1-score   support

Bordeaux-style Red Blend       0.13      0.24      0.17       311
         Cabernet Franc       0.07      0.10      0.08       203
     Cabernet Sauvignon       0.31      0.58      0.40      1381
             Chardonnay       0.42      0.62      0.50      1154
                 Merlot       0.12      0.15      0.13       386
           Petite Sirah       0.07      0.04      0.05       137
             Pinot Gris       0.27      0.31      0.29       139
             Pinot Noir       0.57      0.41      0.48      1795
              Red Blend       0.16      0.07      0.10       489
   Rhône-style Red Blend       0.12      0.01      0.02       116
               Riesling       0.54      0.32      0.40       303
                   Rosé       0.41      0.19      0.26       158
        Sauvignon Blanc       0.39      0.21      0.27       343
        Sparkling Blend       0.67      0.05      0.10        73
                  Syrah       0.32      0.14      0.20       601
               Viognier       0.20      0.03      0.05       144
            White Blend       0.09      0.01      0.02       100
              Zinfandel       0.40      0.06      0.10       465

               accuracy                           0.34      8298
              macro avg       0.29      0.20      0.20      8298
           weighted avg       0.36      0.34      0.32      8298
```

Figure 10: Performance of KNN model

The testing accuracy is 34%. There is overfitting of the data and the model performance is not great however after hypertuning it increased to 42%. The number of neighbors used in KNN is n=3.

4.3.4 Support vector Machine (SVM)



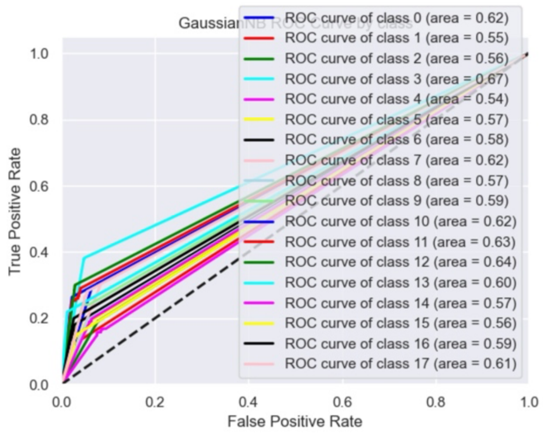


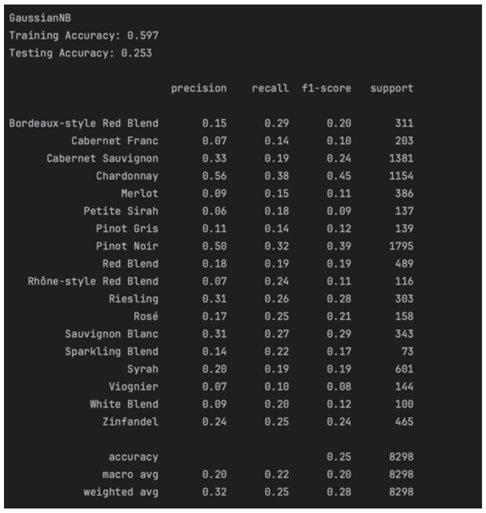
```
SVM
Training Accuracy: 0.214
Testing Accuracy: 0.217

                          precision    recall  f1-score   support

Bordeaux-style Red Blend       0.00      0.00      0.00       311
         Cabernet Franc       0.00      0.00      0.00       203
     Cabernet Sauvignon       0.67      0.00      0.01      1381
             Chardonnay       0.00      0.00      0.00      1154
                 Merlot       0.00      0.00      0.00       386
           Petite Sirah       0.00      0.00      0.00       137
             Pinot Gris       0.00      0.00      0.00       139
             Pinot Noir       0.22      1.00      0.36      1795
              Red Blend       0.00      0.00      0.00       489
   Rhône-style Red Blend       0.00      0.00      0.00       116
               Riesling       0.00      0.00      0.00       303
                   Rosé       0.00      0.00      0.00       158
        Sauvignon Blanc       0.00      0.00      0.00       343
        Sparkling Blend       0.00      0.00      0.00        73
                  Syrah       0.00      0.00      0.00       601
               Viognier       0.00      0.00      0.00       144
            White Blend       0.00      0.00      0.00       100
              Zinfandel       0.00      0.00      0.00       465

               accuracy                           0.22      8298
              macro avg       0.05      0.06      0.02      8298
           weighted avg       0.16      0.22      0.08      8298
```

Figure 11: Performance of SVM model

The SVM model has testing accuracy of 21.7% and the area under the curve is extremely low. SVM model has performed poorly in our dataset.

4.3.5 Naïve Bayes



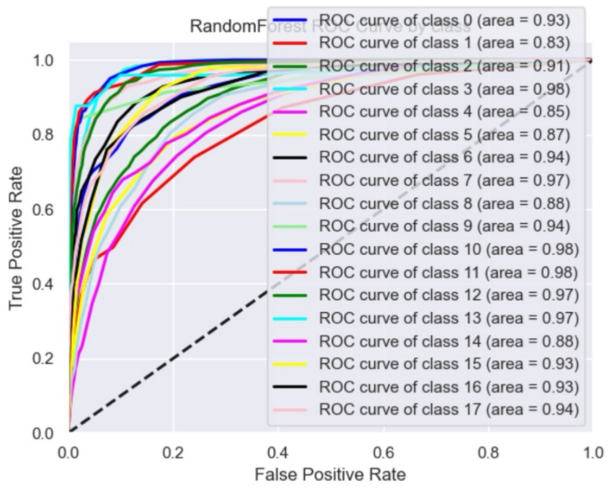


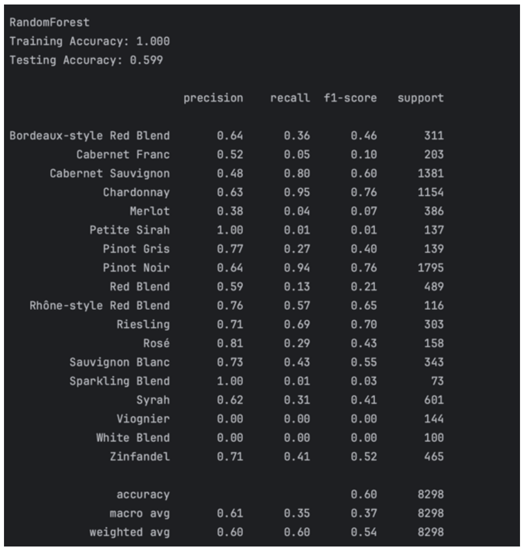
```
GaussianNB
Training Accuracy: 0.597
Testing Accuracy: 0.253

                          precision    recall  f1-score   support

Bordeaux-style Red Blend       0.15      0.29      0.20       311
          Cabernet Franc       0.07      0.14      0.10       203
      Cabernet Sauvignon       0.33      0.19      0.24      1381
              Chardonnay       0.56      0.38      0.45      1154
                  Merlot       0.09      0.15      0.11       386
            Petite Sirah       0.06      0.18      0.09       137
              Pinot Gris       0.11      0.14      0.12       139
              Pinot Noir       0.50      0.32      0.39      1795
               Red Blend       0.18      0.19      0.19       489
   Rhône-style Red Blend       0.07      0.24      0.11       116
                Riesling       0.31      0.26      0.28       303
                    Rosé       0.17      0.25      0.21       158
        Sauvignon Blanc       0.31      0.27      0.29       343
         Sparkling Blend       0.14      0.22      0.17        73
                   Syrah       0.20      0.19      0.19       601
                Viognier       0.07      0.10      0.08       144
             White Blend       0.09      0.20      0.12       100
               Zinfandel       0.24      0.25      0.24       465

                accuracy                           0.25      8298
               macro avg       0.20      0.22      0.20      8298
            weighted avg       0.32      0.25      0.28      8298
```

Figure 12: Performance of Naive Bayes model

The accuracy is 25% . The naive bayes model performs poorly for our dataset.

4.3.6 Random Forest



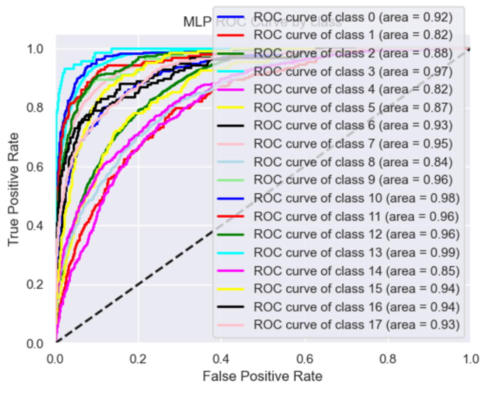


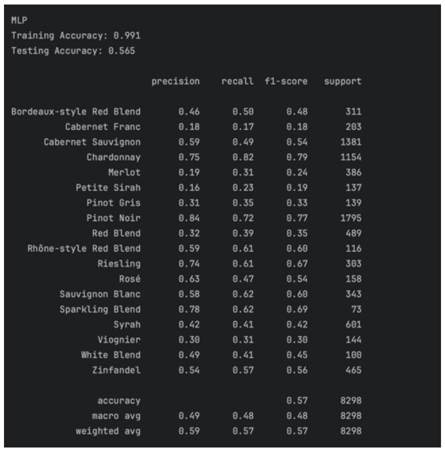
```
RandomForest
Training Accuracy: 1.000
Testing Accuracy: 0.599

                          precision    recall  f1-score   support

Bordeaux-style Red Blend       0.64      0.36      0.46       311
          Cabernet Franc       0.52      0.05      0.10       203
      Cabernet Sauvignon       0.48      0.80      0.60      1381
              Chardonnay       0.63      0.95      0.76      1154
                  Merlot       0.38      0.04      0.07       386
            Petite Sirah       1.00      0.01      0.01       137
              Pinot Gris       0.77      0.27      0.40       139
              Pinot Noir       0.64      0.94      0.76      1795
               Red Blend       0.59      0.13      0.21       489
   Rhône-style Red Blend       0.76      0.57      0.65       116
                Riesling       0.71      0.69      0.70       303
                    Rosé       0.81      0.29      0.43       158
         Sauvignon Blanc       0.73      0.43      0.55       343
         Sparkling Blend       1.00      0.01      0.03        73
                   Syrah       0.62      0.31      0.41       601
                Viognier       0.00      0.00      0.00       144
             White Blend       0.00      0.00      0.00       100
               Zinfandel       0.71      0.41      0.52       465

                accuracy                           0.60      8298
               macro avg       0.61      0.35      0.37      8298
            weighted avg       0.60      0.60      0.54      8298
```

Figure 13: Performance of Random Forest model

The accuracy is found to be 60%. The precision of the model is much higher than any other model. However, there is overfitting as the training accuracy is found to be 100%. We can however remove the overfitting easily by tuning. After hyperparameter tuning it is found to be 75.87% accuracy

4.3.6 Neural Network





```
MLP
Training Accuracy: 0.991
Testing Accuracy: 0.565

                          precision    recall  f1-score   support

Bordeaux-style Red Blend       0.46      0.50      0.48       311
         Cabernet Franc       0.18      0.17      0.18       203
     Cabernet Sauvignon       0.59      0.49      0.54      1381
             Chardonnay       0.75      0.82      0.79      1154
                 Merlot       0.19      0.31      0.24       386
           Petite Sirah       0.16      0.23      0.19       137
             Pinot Gris       0.31      0.35      0.33       139
             Pinot Noir       0.84      0.72      0.77      1795
              Red Blend       0.32      0.39      0.35       489
   Rhône-style Red Blend       0.59      0.61      0.60       116
               Riesling       0.74      0.61      0.67       303
                   Rosé       0.63      0.47      0.54       158
        Sauvignon Blanc       0.58      0.62      0.60       343
        Sparkling Blend       0.78      0.62      0.69        73
                  Syrah       0.42      0.41      0.42       601
               Viognier       0.30      0.31      0.30       144
            White Blend       0.49      0.41      0.45       100
              Zinfandel       0.54      0.57      0.56       465

               accuracy                           0.57      8298
              macro avg       0.49      0.48      0.48      8298
           weighted avg       0.59      0.57      0.57      8298
```

Figure 14: Performance of Neural Network model

The accuracy is found to be 56.5%. We can however remove the overfitting easily by tuning. After hyperparameter tuning it is found to be 58.25% accuracy

4.3.8 Final model selection:

Here we can see the random forest has achieved the highest testing accuracy 60% and has a high AUC. The precision of the model is much higher than any other model. However, there is overfitting as the training accuracy is found to be 100%. We can however remove the overfitting easily by tuning. After hyperparameter tuning it is found to be 75.87% accuracy

4.4 Phase IV- Classification Analysis

Upon completion of Phase 4 of the project, which entailed rigorous independent research, the following observations were made regarding the application of clustering and association rule mining algorithms to the wine variety dataset:

K-Means Algorithm:

- The K-Means algorithm, with the aid of silhouette analysis, revealed distinct groupings within the wine varieties. The silhouette scores provided insights into the cohesion and separation of the clusters, guiding the selection of an optimal number of clusters.

- The within-cluster variation plot highlighted a marked decrease in variance with an increasing number of clusters up to a point, beyond which the marginal gains diminished, indicating a suitable k-value.

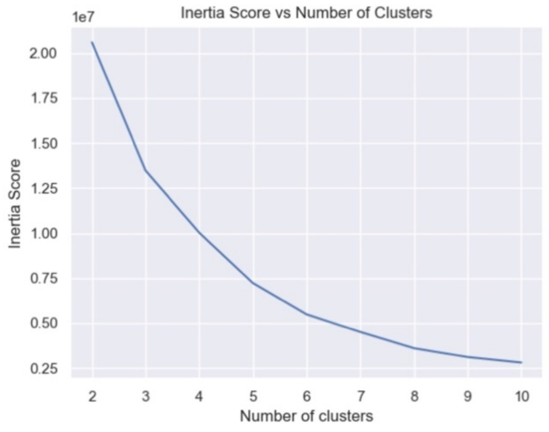
DBSCAN Algorithm:

- DBSCAN's density-based clustering effectively identified outliers and noise, distinguishing between high-density regions (core points) and sparse regions (noise).
- Unlike K-Means, DBSCAN did not force every point into a cluster, allowing for a more nuanced understanding of the dataset's structure.

Apriori Algorithm:

- The Apriori algorithm was instrumental in uncovering association rules among the wine characteristics. For instance, certain grape varieties were frequently associated with specific tasting notes, and certain regions often correlated with higher wine scores.
- The support and confidence measures from the Apriori algorithm indicated strong rules, such as specific wine varieties being highly associated with certain aromas and flavors, which could potentially be used to make predictions about wine characteristics or to inform marketing strategies.

Overall, these clustering and association rule mining techniques provided valuable insights into the relationships within the wine variety dataset, offering a potential for predictive modeling and a deeper understanding of the factors that influence wine variety classification.



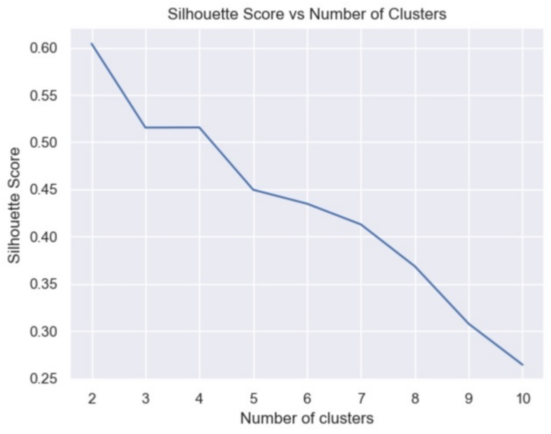


Figure 15: Independent Research

5. Recommendations

In this project, I gained a comprehensive understanding of both data preprocessing and machine learning modeling. Since the dataset needed extensive cleaning, the addition of new variables, and encoding of categorical attributes, I delved deeply into data preprocessing, acquiring valuable skills throughout this phase. Additionally, I honed my expertise in applying various machine learning techniques to the processed dataset. Subsequently, I employed these machine learning models to make predictions and conducted essential evaluations, including generating the confusion matrix, ROC curve, and other critical metrics.

a. **What did you learn from this project?**

Throughout this project, we gained several key insights:

- The dataset provides a comprehensive view of wine reviews, with a primary focus on wines from the United States.
- Feature engineering and exploratory data analysis (EDA) are essential steps in understanding and preparing the data for modeling.
- Regression analysis provided valuable information about trends and relationships within the dataset.
- Classification analysis using various machine learning models revealed the Random Forest as the top-performing model, with room for improvement through hyperparameter tuning.
- Clustering and association rule mining techniques, such as K-Means, DBSCAN, and Apriori, offered deeper insights into the dataset's structure and relationships.

b. **Which classifiers perform the best for the selected dataset?**

Among the classifiers tested, the Random Forest classifier demonstrated the highest testing accuracy (60%) and had a high AUC score. It also showed superior precision compared to other models. However, it's important to address overfitting by tuning hyperparameters to achieve optimal performance.

c. **How do you think you can improve the performance of the classification?**

To enhance classification performance, we recommend the following actions:

- Hyperparameter Tuning: Further tune the hyperparameters of the Random Forest model to reduce overfitting and improve generalization.
- Feature Engineering: Explore additional feature engineering techniques to capture more meaningful information from the dataset.

- Ensembling: Consider ensembling multiple classifiers or models to combine their strengths and enhance overall performance.

- Address Class Imbalance: If class imbalance exists in the dataset, employ techniques like oversampling or undersampling to balance the class distribution.

d. **What features are associated with the target variable?**

Feature importance analysis should be conducted to identify which attributes or features have the most significant influence on predicting wine variety. This analysis can provide valuable insights into the key factors that differentiate wine varieties.

e. **Number of clusters in this feature space.**

Clustering analysis using techniques like K-Means and DBSCAN has indicated the presence of distinct groupings within the dataset. Further exploration of these clusters and their characteristics can yield valuable information for wine variety classification and segmentation.

In summary, this project has uncovered valuable insights into wine variety prediction, with a focus on feature engineering, regression analysis, and classification using machine learning models. The Random Forest classifier has shown promise but requires hyperparameter tuning to optimize its performance. Additionally, the clustering and association rule mining techniques have the potential to provide deeper insights into the dataset's structure and relationships, which can be further explored in future research.

6. Appendix

In this section of the report softcopy of the code is included.

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns; sns.set()

data = pd.read_csv('winemag-data-130k-v2.csv')
data.head(20)
data.info()
data.describe()
data[['country','variety','winery','taster_name','taster_twitter_handle','provi
nce','region_1','region_2']].describe()
data.country.value_counts().head(15).plot.barh(width=0.9,figsize=(10,6),color='
darkred');
(data.country.value_counts(normalize=True)*100).head(6)
data.variety.value_counts().head(70)
US = data[data['country'] == 'US']
US.head()
years = US.title.str.extract('([1-2][0-9]{3})').astype(float)
years[years < 1990] = None
```

```
US = US.assign(year = years)
US=US.dropna(subset=['price'])
plt.scatter(x=US[US['variety'] == 'Pinot Noir']['points'],y=US[US['variety'] ==
'Pinot Noir']['price'],c=US[US['variety'] == 'Pinot Noir']['year']);
US[US['variety'] == 'Pinot Noir']
sns.boxplot(x='variety', y='year', data = US[US['variety'] == 'Pinot Noir'])
sns.jointplot(x='year',y='price',data=US[US['variety'] == 'White Blend']);
US = US.drop_duplicates('description')
US.shape
US.variety.value_counts()
US = US.groupby('variety').filter(lambda x: len(x) >500)
wine_us =US.variety.unique().tolist()
wine_us.sort()
from sklearn.model_selection import train_test_split
X = US.drop(['Unnamed:
0','country','designation','points','province','taster_name',
       'taster_twitter_handle',
'title','region_1','region_2','variety','winery'], axis = 1)
print(X.columns, "X columns")
y = US.variety
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=42)
print(X_train.shape, X_test.shape, y_train.shape, y_test.shape)
output = set()
for x in US.variety:
    x = x.lower()
    x = x.split()
    for y in x:
        output.add(y)
variety_list =sorted(output)
extras = ['.', ',', '"', "'", '?', '!', ':', ';','-' ,'(', ')', '[', ']', '{',
'}', 'cab',"%"]

from nltk.corpus import stopwords
import nltk
stop = set(stopwords.words('english'))
stop.update(variety_list)
stop.update(extras)
US.variety.value_counts()
US.head()
sum(US['year']>2002)
data_=US[US['year']>2002]
plt.figure(figsize=(10,4))
data_.variety.value_counts().plot.bar()
rep_v = data_.groupby('variety')['year'].agg('median')
for i in rep_v.index :
    data_.loc[(data_['year'].isna()) & (data_['variety'] == i),'year']=rep_v[i]
data_.variety.describe()
X_ = data_.drop(['Unnamed: 0','designation','country','taster_name',
       'taster_twitter_handle',
'title','region_1','region_2','variety','winery'], axis = 1)
y = data_.variety
print("Length of y:", len(y))
print("Length of X:", len(X_))
X = pd.get_dummies(X_,columns=["province"])
X_apriori = X.copy()
# drop all features except one hot encoded province
columns_to_drop = [
    'description', 'price', 'year', 'points'
]
for col in columns_to_drop:
    if col in X_apriori.columns:
        X_apriori.drop(col, axis=1, inplace=True)
province_columns = [col for col in X.columns if 'province_' in col]
X[province_columns] = X[province_columns].astype(int)
# X = pd.get_dummies(X_, columns=["province"], drop_first=True)
# X = X.astype(int)
# y = data_.variety
print(y)
X.info()
X.head()


#################PCA#####################
# PCA
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler
from sklearn.feature_extraction.text import CountVectorizer
from scipy.sparse import  hstack
# Copy the dataset
df_for_PCA = X.copy()
# stop_list = list(stop)
# vect = CountVectorizer(stop_words=stop_list)
# X_dtmm = vect.fit_transform(df_for_PCA.description)
# Xss = df_for_PCA.drop(['description'], axis=1).to_numpy()
# X_combined = hstack((Xss, X_dtmm)).todense()
# X_combined_dense = np.asarray(X_combined)

# Standardize the data
scaler = StandardScaler()
# show columns in df_for_PCA
print(df_for_PCA.columns, "df_for_PCA.columns")
# drop description column for now and all province_ columns
df_for_PCA = df_for_PCA.drop(['description'], axis=1)
columns_to_drop = [
    'province_America', 'province_Arizona', 'province_California',
    'province_Colorado', 'province_Connecticut', 'province_Idaho',
    'province_Kentucky', 'province_Massachusetts', 'province_Michigan',
    'province_Missouri', 'province_Nevada', 'province_New Jersey',
    'province_New Mexico', 'province_New York', 'province_North Carolina',
    'province_Ohio', 'province_Oregon', 'province_Pennsylvania',
    'province_Texas', 'province_Virginia', 'province_Washington',
    'province_Washington-Oregon'
]

# Dropping columns from df_for_PCA
for col in columns_to_drop:
    if col in df_for_PCA.columns:
        df_for_PCA.drop(col, axis=1, inplace=True)
# print df_for_PCA.columns
print(df_for_PCA.columns, "df_for_PCA.columns")
# make a for loop to drop all province_ columns

X_std = scaler.fit_transform(df_for_PCA)
```

```
# Drop the specified columns
# display df_for_PCA

#one-hot encode the day of week column
# df_for_PCA = pd.get_dummies(df_for_PCA, columns=["day_of_week"])

# Standardize the data to have a mean of ~0 and a variance of 1

# Create a PCA instance: pca
pca = PCA(n_components=2)
principalComponents = pca.fit_transform(X_std)

# Plot an elbow curve to find the optimal number of components

elbow = PCA().fit(X_std)
plt.plot(np.cumsum(elbow.explained_variance_ratio_))
plt.xlabel('number of components')
plt.ylabel('cumulative explained variance')
plt.show()

print("The first two principal components explain {}% of the variance.".format(
    round(sum(elbow.explained_variance_ratio_[:2])*100, 2)))

print(elbow.explained_variance_ratio_,"elbow.explained_variance_ratio_")
# plot elbow plot for above elbow variable
plt.plot(elbow.explained_variance_ratio_)
plt.xlabel('number of components')
plt.ylabel('cumulative explained variance')
plt.show()


df_for_PCA.head()

#################PCA#####################


#######heat map################
# show the heatmap for df_for_PCA
sns.heatmap(df_for_PCA.corr(), annot=True, cmap='coolwarm')
plt.title('Correlation Coefficient')
plt.show()

# show the heatmap for df_for_PCA using pearson correlation
sns.heatmap(df_for_PCA.corr(method='pearson'), annot=True, cmap='coolwarm')
plt.title('Pearson Correlation Coefficient')
plt.show()


# do t test for df_for_PCA
# add variety to df_for_PCA from data
from scipy.stats import ttest_ind
df_for_PCA['variety'] = data_.variety
numerical_columns = ["price", "year", "points"]
print(df_for_PCA.head(), "check this")
df_for_PCA['variety_numerical'] = pd.Categorical(df_for_PCA['variety']).codes
```

```
df_for_PCA = df_for_PCA.drop('variety', axis=1)
df_for_PCA_for_f_test = df_for_PCA.copy()
def t_test(data, target, columns):
    for col in columns:
        t_value, p_value = ttest_ind(data[data[target] == 0][col],
                                     data[data[target] == 1][col],
                                     nan_policy='omit')  # Omit NaN values
        print(f'T-value for {col} vs {target}: {t_value}')
        print(f'P-value for {col} vs {target}: {p_value}')

# Perform the t-test
t_test(df_for_PCA, 'variety_numerical', numerical_columns)

#perform the f test
from scipy.stats import f_oneway
def f_test(data, target, columns):
    unique_varieties = data[target].unique()
    groups = [data[data[target] == variety][col] for variety in
unique_varieties for col in columns]
    f_value, p_value = f_oneway(*groups)
    print(f'F-value for {col} vs {target}: {f_value}')
    print(f'P-value for {col} vs {target}: {p_value}')

# Perform the F-test
f_test(df_for_PCA, 'variety_numerical', numerical_columns)

#######After t test###############

new_df = df_for_PCA.copy()

X_train, X_test, y_train, y_test = train_test_split(
    new_df.drop('variety_numerical', axis=1),
    new_df['variety_numerical'],
    test_size=0.3,
    random_state=5805  # Set a random seed for reproducibility
)

import statsmodels.api as sm

model = sm.OLS(y_train, sm.add_constant(X_train)).fit()  # Add a constant term
print(model.summary())

f_statistic = model.fvalue
p_value = model.f_pvalue

print('F statistic = ' + str(f_statistic))
print('P-value = ' + str(p_value))

# plot train, test, and predicted values in one plot
plt.scatter(X_train['points'], y_train, color='blue', label='train')
plt.scatter(X_test['points'], y_test, color='red', label='test')
plt.scatter(X_test['points'], model.predict(sm.add_constant(X_test)),
color='green', label='predicted')
plt.xlabel('points')
plt.ylabel('variety')
plt.legend()
plt.show()
```

```python
# Confidence interval Analysis
import statsmodels.stats.api as sms

confidence_interval = model.conf_int()
print(f"Confidence interval:\n{confidence_interval}")


final_confidence_interval = model
y_pred = model.predict(sm.add_constant(X_test))

#Calculate  and print R squared, adjusted R - square, AIC, BIC and MSE

from sklearn.metrics import mean_squared_error
from sklearn.metrics import r2_score
from sklearn.metrics import mean_squared_error
from sklearn.metrics import mean_absolute_error
from sklearn.metrics import mean_squared_log_error
from math import sqrt

r_squared = model.rsquared
adj_r_squared = model.rsquared_adj
aic = model.aic
bic = model.bic
mse = mean_squared_error(y_test, y_pred)

print(f"R-squared: {r_squared}")
print(f"Adjusted R-squared: {adj_r_squared}")
print(f"AIC: {aic}")
print(f"BIC: {bic}")
print(f"MSE: {mse}")








from sklearn.feature_extraction.text import CountVectorizer
#generando variable de textmining

from scipy.sparse import csr_matrix, hstack
stop_list = list(stop)
vect = CountVectorizer(stop_words = stop_list)
X_dtm = vect.fit_transform(X.description)
Xs = X.drop(['description'],axis=1).to_numpy()
print(Xs)
X_dtm = hstack((Xs,X_dtm))
X_dtm_dense = X_dtm.todense()
df_X_dtm = pd.DataFrame(X_dtm_dense)

# View the DataFrame
print(df_X_dtm.head())
print(X_dtm, "X_dtm")
len(y)
print("Length of X_dtm:", X_dtm.shape[0])
print("Length of y:", len(y))
```

```
Xtrain,Xtest,ytrain,ytest = train_test_split(X_dtm,y,random_state=1)





##############Phase 3#####################
Xtrain,Xtest,ytrain,ytest = train_test_split(X_dtm,y,random_state=1)
print("new Length of y:", len(y))
wine=y.unique()
from sklearn.linear_model import LogisticRegression
models = {}
for z in wine:
    model = LogisticRegression()
    y_binary = ytrain == z
    model.fit(Xtrain, y_binary)
    models[z] = model
testing_probs = pd.DataFrame(columns = wine)
len(wine)
probs = pd.DataFrame(columns = wine)
for z in wine:
    probs[z] = models[z].predict_proba(Xtest)[:,1]
probs.head()
probs_=probs.fillna(0)
pred = probs.idxmax(axis=1)
comparison = pd.DataFrame({'actual':ytest.values, 'predicted':pred.values})
from sklearn.metrics import classification_report,
confusion_matrix,accuracy_score
print('Accuracy Score:',accuracy_score(comparison.actual,
comparison.predicted)*100,"%")\

from sklearn.tree import DecisionTreeClassifier



from sklearn.metrics import accuracy_score, r2_score, confusion_matrix,
classification_report, recall_score, \
    confusion_matrix, mean_squared_error, ConfusionMatrixDisplay,
RocCurveDisplay
from sklearn.linear_model import LogisticRegression, LinearRegression
from sklearn.model_selection import train_test_split, KFold, cross_val_score,
ShuffleSplit
from sklearn.metrics import classification_report, accuracy_score, roc_curve,
auc
from sklearn.neighbors import KNeighborsRegressor
import re
from sklearn.ensemble import RandomForestClassifier
from sklearn.datasets import make_classification
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.svm import SVC
from sklearn.naive_bayes import GaussianNB
from sklearn.preprocessing import label_binarize
from itertools import cycle
from sklearn.neural_network import MLPClassifier
x_train, x_test, y_train, y_test = train_test_split(X_dtm, y, test_size=0.2,
random_state=5805)
```

```
# CREATING ALL THE MODELS

models = []
models.append(('LogisticRegression', LogisticRegression()))
models.append(('RandomForest', RandomForestClassifier()))
models.append(('Decision Tree', DecisionTreeClassifier()))
models.append(('KNN', KNeighborsClassifier(n_neighbors=3)))
models.append(('SVM', SVC(probability=True)))
models.append(('MLP', MLPClassifier(hidden_layer_sizes=(100, ), max_iter=500,
random_state=42)))
models.append(('GaussianNB', GaussianNB()))

# RUNNING OF THE MODELS
# IN PY CHARM CHECK THE PLOT FOR CONFUSION MATRIX

for name, model in models:
    print()
    print()
    # Train model
    if name in ['GaussianNB', 'MLP']:
        model.fit(x_train.toarray(), y_train)
        predictions = model.predict(x_test.toarray())
    else:
        model.fit(x_train, y_train)
        predictions = model.predict(x_test)

    # Print accuracy
    print(name)
    print(f"Training Accuracy: {accuracy_score(y_train,
model.predict(x_train.toarray() if name == 'GaussianNB' else x_train)):.3f}")
    print(f"Testing Accuracy: {accuracy_score(y_test, predictions):.3f}\n")

    # Classification report
    print(classification_report(y_test, predictions, zero_division=0))

    # Plot ROC Curve for each class
    y_test_bin = label_binarize(y_test, classes=np.unique(y))
    n_classes = y_test_bin.shape[1]
    y_pred_proba = model.predict_proba(x_test.toarray() if name == 'GaussianNB'
else x_test)

    fpr = dict()
    tpr = dict()
    roc_auc = dict()
    for i in range(n_classes):
        fpr[i], tpr[i], _ = roc_curve(y_test_bin[:, i], y_pred_proba[:, i])
        roc_auc[i] = auc(fpr[i], tpr[i])

    # Plot ROC curve
    plt.figure()
    colors = cycle(['blue', 'red', 'green', 'cyan', 'magenta', 'yellow',
'black', 'pink', 'lightblue', 'lightgreen'])
    for i, color in zip(range(n_classes), colors):
        plt.plot(fpr[i], tpr[i], color=color, lw=2,
                 label=f'ROC curve of class {i} (area = {roc_auc[i]:.2f})')
    plt.plot([0, 1], [0, 1], 'k--', lw=2)
    plt.xlim([0.0, 1.0])
    plt.ylim([0.0, 1.05])
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title(f'{name} ROC Curve by class')
    plt.legend(loc="lower right")
    plt.show()


    # Plotting Confusion Matrix
    cm = confusion_matrix(y_test, predictions, labels=np.unique(y))
    # dont plot consfusion matrix but display it
    print(cm,"Confusion matrix")



################PHASE 4######################
# # K-mean clustering with silhouette score for the k-selection within cluster
variation plot
#
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score
import matplotlib.pyplot as plt
#
# # Create empty lists to store the scores for the evaluation metrics
silhouette_scores = []
inertia_scores = []

# Create a list of possible k values
k_values = list(range(2, 11))

# For each k value
for k in k_values:
    # Create a KMeans instance with k clusters
    kmeans = KMeans(n_clusters=k, random_state=5805)
    # Fit the model to the data
    kmeans.fit(X_dtm)
    # Append the average silhouette score and inertia score to the respective
lists
    silhouette_scores.append(silhouette_score(X_dtm, kmeans.labels_))
    inertia_scores.append(kmeans.inertia_)

# Plot the silhouette score vs k values
plt.plot(k_values, silhouette_scores)
plt.xlabel('Number of clusters')
plt.ylabel('Silhouette Score')
plt.title('Silhouette Score vs Number of Clusters')
plt.show()

# Plot the inertia score vs k values
plt.plot(k_values, inertia_scores)
plt.xlabel('Number of clusters')
plt.ylabel('Inertia Score')
plt.title('Inertia Score vs Number of Clusters')
plt.show()


# phase 4 part 2
# from sklearn.cluster import DBSCAN
# from sklearn.metrics import silhouette_score
```

```
#
# # Create an empty list to store the silhouette scores
# silhouette_scores = []
#
# # Define a range of possible eps values
# eps_values = [0.001, 0.01, 0.1, 1, 10, 100]
#
# # Iterate over different eps values
# for eps in eps_values:
#     # Create a DBSCAN instance with the current eps value
#     dbscan = DBSCAN(eps=eps)
#     # Fit the model to the data
#     dbscan.fit(X_dtm)
#
#     # Check if more than one cluster has been formed (excluding noise points)
#     if len(set(dbscan.labels_)) > 1:
#         # Calculate silhouette score and append it to the list
#         silhouette_scores.append(silhouette_score(X_dtm, dbscan.labels_))
#     else:
#         # Append a default value (e.g., -1) if only one cluster is formed
#         silhouette_scores.append(-1)
#
# # Plot the silhouette scores against the eps values
# plt.plot(eps_values, silhouette_scores)
# plt.xlabel('Eps values')
# plt.ylabel('Silhouette Score')
# plt.title('Silhouette Score vs Eps values')
# plt.show()
#

# from sklearn.cluster import DBSCAN
# from sklearn.decomposition import PCA
# from sklearn.preprocessing import StandardScaler
# from sklearn.pipeline import make_pipeline
# import numpy as np
# import pandas as pd
#
# # Assuming X_dtm is already defined and is your dataset
#
# # Create a pipeline that standardizes the data and then applies PCA
# pca_pipeline = make_pipeline(StandardScaler(), PCA(n_components=0.95))
#
# # Fit the pipeline and transform the data
# X_dtm_pca = pca_pipeline.fit_transform(X_dtm)
#
# # Sample a subset of your data if it's too large
# # For example, sample 10% of your data
# sample_idx = np.random.choice(X_dtm_pca.shape[0], int(X_dtm_pca.shape[0] *
0.1), replace=False)
# X_dtm_sample = X_dtm_pca[sample_idx, :]
#
# # Define DBSCAN with a larger eps and min_samples
# dbscan = DBSCAN(eps=5, min_samples=10)
#
# # Fit DBSCAN to the sample data
# dbscan.fit(X_dtm_sample)
#
```

```
from mlxtend.frequent_patterns import apriori, association_rules

# Assuming 'df_for_PCA' is preprocessed and contains only one-hot encoded
features
# Apply the Apriori algorithm to find frequent itemsets
# select
frequent_itemsets = apriori(X_apriori, min_support=0.01, use_colnames=True)

# Generate association rules from the frequent itemsets
rules = association_rules(frequent_itemsets, metric='confidence',
min_threshold=0.5)

# Display the top association rules sorted by confidence
rules = rules.sort_values(by='confidence', ascending=False)
print(rules.head(), "rules.head()")
```

7. References

- https://www.geeksforgeeks.org/detect-and-remove-the-outliers-using-python/
- https://www.geeksforgeeks.org/ml-label-encoding-of-datasets-in-python/
- https://scikit-learn.org/stable/modules/generated/sklearn.metrics.ConfusionMatrixDispl ay.html
- https://towardsdatascience.com/ways-to-detect-and-remove-the-outliers-404d16608db a
- https://www.kaggle.com/datasets/zynicide/wine-reviews