ML Project Documentation

Iuliu Andrei Steau

January 17, 2025

1 Problem Statement

1.1 Domain Description

This project falls within the domain of machine learning (ML), a branch of artificial intelligence (AI) that focuses on developing algorithms and statistical models that enable computers to learn from and make predictions or decisions based on data without being explicitly programmed. The primary goal is to build a model that generalizes well to unseen data, effectively solving the classification on two datasets.

1.2 Requirements and Objectives

The project should describe the results of a well-chosen machine learning model on two datasets of different sizes, one with approximately 1000 samples and less than 100 features and the other with over 100,000 samples and 1000+ features. In addition, detailed analyzes of the data and performance evaluations must be presented to demonstrate the correctness of the results obtained.

The project was structured around the following objectives:

- Demonstrate the ability to handle datasets of varying complexity and scale using machine learning techniques.
- Apply machine learning methods to solve binary classification problems and optimize algorithm performance through parameter tuning and evaluation.
- Analyze the impact of data characteristics on model performance and decision-making processes.

2 Dataset Description

2.1 Small Dataset

The small data set contains data on wine quality, having 13 attributes and 1143 samples. These parameters describe the physical and chemical properties of the wine.

Numerical attributes

- Fixed and Volatile acidity
- Citric acid
- residual sugar
- chlorides
- Free and Total sulfur dioxide
- Density
- pH

- sulphates
- alcohol

Target

- Quality

2.2 Large Dataset

The large dataset contains tree observations from four areas of the Roosevelt National Forest in Colorado. All observations are cartographic variables (no remote sensing) from 30 meter x 30 meter sections of forest. There are over half a million measurements total. This includes information on tree type, shadow coverage, distance to nearby landmarks (roads etcetera), soil type, and local topography.

Numerical attributes

- Elevation
- Aspect
- Slope
- Horizontal and Vertical Distances (4 distances)
- Hillshade (3 hillshades)

Categorical attributes

- Wilderness (4 areas)
- Soil Types (38 types)

Target

- Cover Type

3 Theoretical Foundations

3.1 Random Forest

Random forests or random decision forests is an ensemble learning method for classification, regression and other tasks that works by creating a multitude of decision trees during training. For classification tasks, the output of the random forest is the class selected by most trees.

The Random Forest algorithm introduces extra randomness when growing trees, instead of searching for the very best feature when splitting a node, it searches for the best feature among a random subset of features. This results in a greater tree diversity, which (once again) trades a higher bias for a lower variance, generally yielding an overall better model.

Bagging, short for **Bootstrap Aggregating**, is a foundational technique used in Random Forest to enhance model stability and accuracy. It works by generating multiple subsets of the original dataset through random sampling with replacement, and then training a separate decision tree on each subset. The final prediction of the Random Forest model is obtained by aggregating the predictions of all individual trees.

Decision trees in Random Forest are simple models that learn from random subsets of the data and features, and their collective predictions are combined to enhance performance and reduce the risk of overfitting

3.2 Mathematical Background of Random Forest

3.2.1 Decision Tree

A decision tree is a non-linear model that recursively splits the feature space into smaller regions based on feature values, allowing for prediction of output variables. The splits are made based on criteria that minimize impurity at each node.

Each decision tree learns a function $f(\mathbf{X}) = \hat{y}$, where $\mathbf{X} = (X_1, X_2, \ldots, X_p)$ is the vector of features, and $\hat{y}$ is the predicted target variable.

A decision tree consists of:

- **Root node**: The first node that contains the entire dataset.
- **Internal nodes**: These nodes represent tests on features, each splitting the dataset into two child nodes.
- **Leaves (terminal nodes)**: These nodes contain the predictions. For classification, they hold the predicted class, and for regression, they contain the predicted value (mean of the target variable in that region).

At each internal node, a decision is made by testing a feature and threshold value:

$$X_j \leq \theta, \quad \text{where } X_j \text{ is the feature, and } \theta \text{ is the threshold.}$$

To decide the best feature and threshold at each node, the algorithm uses impurity measures to evaluate how well a split separates the data. Common impurity measures include:

- **Gini Impurity**
- **Entropy**

These measures quantify how mixed the target classes are in a node. The goal is to minimize impurity and create pure nodes that have data belonging mostly to a single class (for classification).

The **Gini Impurity** is a measure used for classification tasks. It quantifies the degree of disorder or impurity in a node. For a node t, it is defined as:

$$\text{Gini}(t) = 1 - \sum_{k=1}^{K} p_k(t)^2,$$

where:

- K is the number of classes.
- $p_k(t)$ is the proportion of samples belonging to class k in node t.

The Gini Impurity ranges from 0 (pure node, all samples belong to one class) to 0.5 (completely impure, classes are evenly distributed).

Entropy is another measure of impurity used in decision trees, based on information theory. For a node t, entropy is defined as:

$$\text{Entropy}(t) = -\sum_{k=1}^{K} p_k(t) \log_2 p_k(t),$$

where:

- $p_k(t)$ is the probability of class k in node t.
- The sum is over all classes.

Entropy ranges from 0 (perfectly pure node, all data points belong to one class) to $\log_2 K$ (maximum entropy, classes are equally distributed).

The objective when splitting a node is to reduce entropy as much as possible, selecting the feature and threshold that results in the highest information gain.

3.2.2 Bootstrap Aggregating (Bagging)

Random Forest applies **Bootstrap Aggregating** (bagging) to construct multiple decision trees. In bagging, each decision tree is trained on a random subset of the original training data, selected with replacement. This random sampling is known as a *bootstrap sample*, and leads to high diversity among the trees, reducing variance and overfitting.

3.2.3 Random Feature Selection

To further increase the diversity between the trees and reduce correlation, Random Forest selects a random subset of features for each node split. If there are p total features, only a subset of m features is considered for each split. Typically, $m = \sqrt{p}$ for classification tasks. For each tree t_i, a bootstrap sample S_i is drawn from the original dataset D. This sample is used to train tree t_i. Since the sample is chosen with replacement, some data points in D will appear multiple times in S_i, while others may be excluded.

$$S_i = \{(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)\} \quad \text{(with replacement)}.$$

3.2.4 Aggregation of Predictions in Classification

In Random Forest, the final prediction is made by aggregating the predictions from all the individual decision trees. For classification tasks, the aggregation is done using **majority voting**.

Each tree in the forest makes a prediction by classifying the input data to one of the K classes. The final predicted class is the class that receives the most votes from the individual trees. Mathematically, the prediction is:

$$\hat{y} = \text{mode}(\hat{y}_1, \hat{y}_2, \ldots, \hat{y}_T),$$

where:

- $\hat{y}_t$ is the predicted class label from tree t.
- T is the total number of trees in the Random Forest.

The class with the majority of votes across all trees becomes the final prediction. In case of a tie, a random tie-breaking mechanism or a secondary criterion can be applied.

3.3 Overfitting and Generalization in Random Forest

Random Forest effectively mitigates overfitting due to its ensemble nature. While individual decision trees are prone to overfitting (especially with deep trees), the averaging process of Random Forest reduces this risk. Bagging reduces variance, and random feature selection helps decrease correlation between trees, making the model more robust to noise in the data.

By combining multiple decision trees, each trained on different subsets of the data and using different features, Random Forest is able to generalize well even for complex multiclass classification problems.

3.3.1 Advantages of Random Forest for Multiclass Classification

- **High Accuracy**: Random Forest tends to perform very well on multiclass classification problems due to its ability to aggregate multiple trees' predictions.
- **Robust to Overfitting**: Despite individual decision trees potentially overfitting, the aggregation process in Random Forest reduces the overall variance.
- **Handles Missing Data**: Random Forest can handle missing data by considering different ways to impute or ignore missing values during training.
- **Feature Importance**: Random Forest can provide insights into the importance of each feature in predicting the target classes, which is useful for feature selection.

This random feature selection ensures that each tree learns different patterns from the data, contributing to the model's diversity.

3.4 Shapley Values

Shapley values, originating from cooperative game theory, provide a principled approach for attributing the contributions of individual features in a predictive model. They fairly distribute the total model output among the features based on their marginal contributions.

3.4.1 Definition

Given a model f that predicts an output based on input features $\mathbf{x} = \{x_1, x_2, \ldots, x_p\}$, the Shapley value ϕ_i for feature x_i is defined as:

$$\phi_i = \sum_{S \subseteq N \setminus \{i\}} \frac{|S|!(|N| - |S| - 1)!}{|N|!} \left[f(S \cup \{i\}) - f(S) \right],$$

where:

- $N = \{1, 2, \ldots, p\}$ is the set of all features.
- $S \subseteq N \setminus \{i\}$ represents a subset of features excluding x_i.
- $|S|$ is the size of subset S.
- $f(S)$ denotes the model's prediction using only the features in subset S.
- $f(S \cup \{i\})$ represents the prediction with feature x_i added to S.

3.4.2 Marginal Contribution

The term $f(S \cup \{i\}) - f(S)$ represents the **marginal contribution** of feature x_i when added to the subset S. It quantifies how much the output changes due to the inclusion of x_i.

3.4.3 Weighting Factor

The factor $\frac{|S|!(|N|-|S|-1)!}{|N|!}$ assigns a weight to each subset S to ensure all permutations of features are considered equally.

3.4.4 Computational Complexity

Since the formula requires evaluating all 2^p subsets of features, computing exact Shapley values is infeasible for large p. In practice, Monte Carlo approximations or other heuristic methods are used to estimate Shapley values efficiently.

3.4.5 Key Properties

Shapley values have desirable properties for feature attribution:

- **Efficiency**: The sum of all Shapley values equals the difference between the total model output and a baseline prediction.
- **Symmetry**: Features with identical effects receive the same Shapley value.
- **Additivity**: For combined models, the Shapley values are the sum of the values for individual models.

4 Design and Implementation

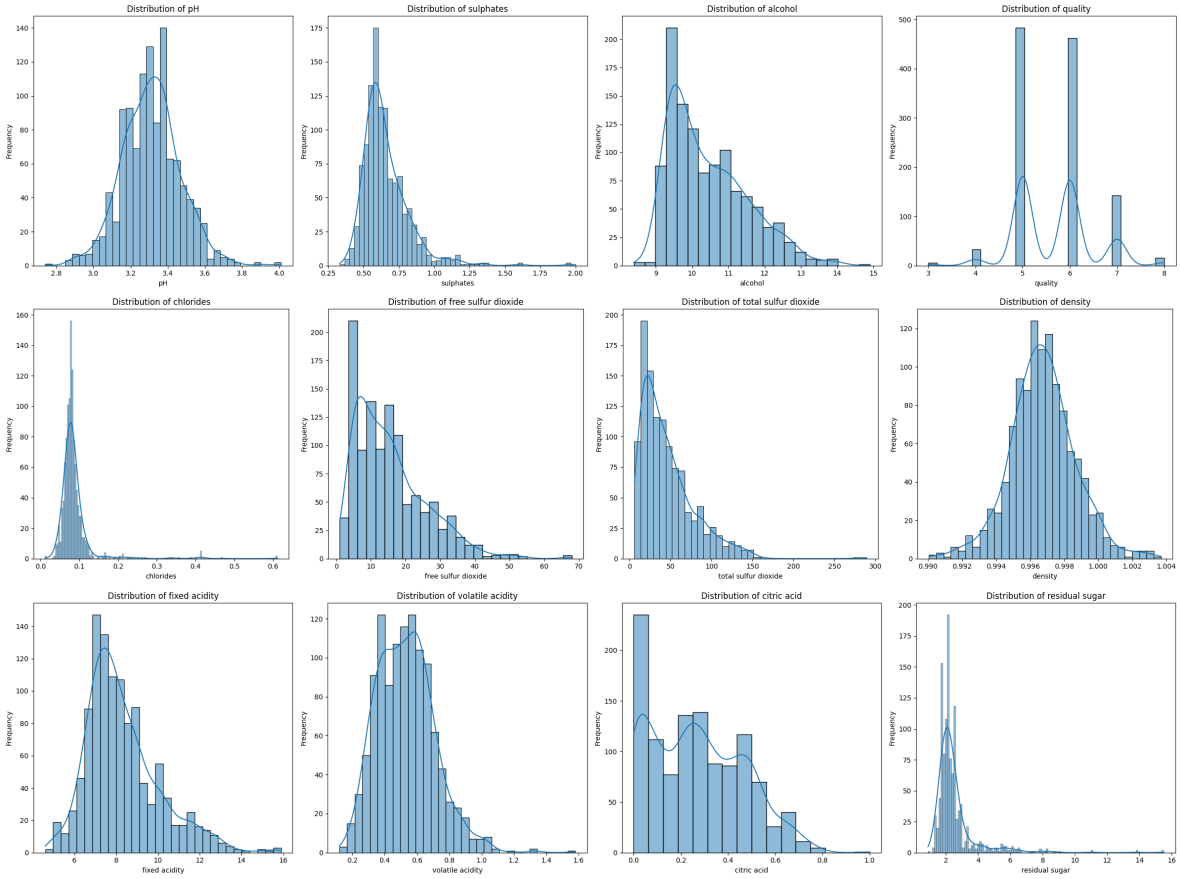
4.1 Wine Dataset

To begin with, I loaded the file and analyzed the data at first glance, more precisely I listed a series of statistics for each attribute of the dataset.

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	quality
count	1143.000000	1143.000000	1143.000000	1143.000000	1143.000000	1143.000000	1143.000000	1143.000000	1143.000000	1143.000000	1143.000000	1143.000000
mean	8.311111	0.531339	0.268364	2.532152	0.086933	15.615486	45.914698	0.996730	3.311015	0.657708	10.442111	5.657043
std	1.747595	0.179633	0.196686	1.355917	0.047267	10.250486	32.782130	0.001925	0.156664	0.170399	1.082196	0.805824
min	4.600000	0.120000	0.000000	0.900000	0.012000	1.000000	6.000000	0.990070	2.740000	0.330000	8.400000	3.000000
25%	7.100000	0.392500	0.090000	1.900000	0.070000	7.000000	21.000000	0.995570	3.205000	0.550000	9.500000	5.000000
50%	7.900000	0.520000	0.250000	2.200000	0.079000	13.000000	37.000000	0.996680	3.310000	0.620000	10.200000	6.000000
75%	9.100000	0.640000	0.420000	2.600000	0.090000	21.000000	61.000000	0.997845	3.400000	0.730000	11.100000	6.000000
max	15.900000	1.580000	1.000000	15.500000	0.611000	68.000000	289.000000	1.003690	4.010000	2.000000	14.900000	8.000000

```
df_wine = pd.read_csv('WineQT.csv')
df_wine.describe()
```

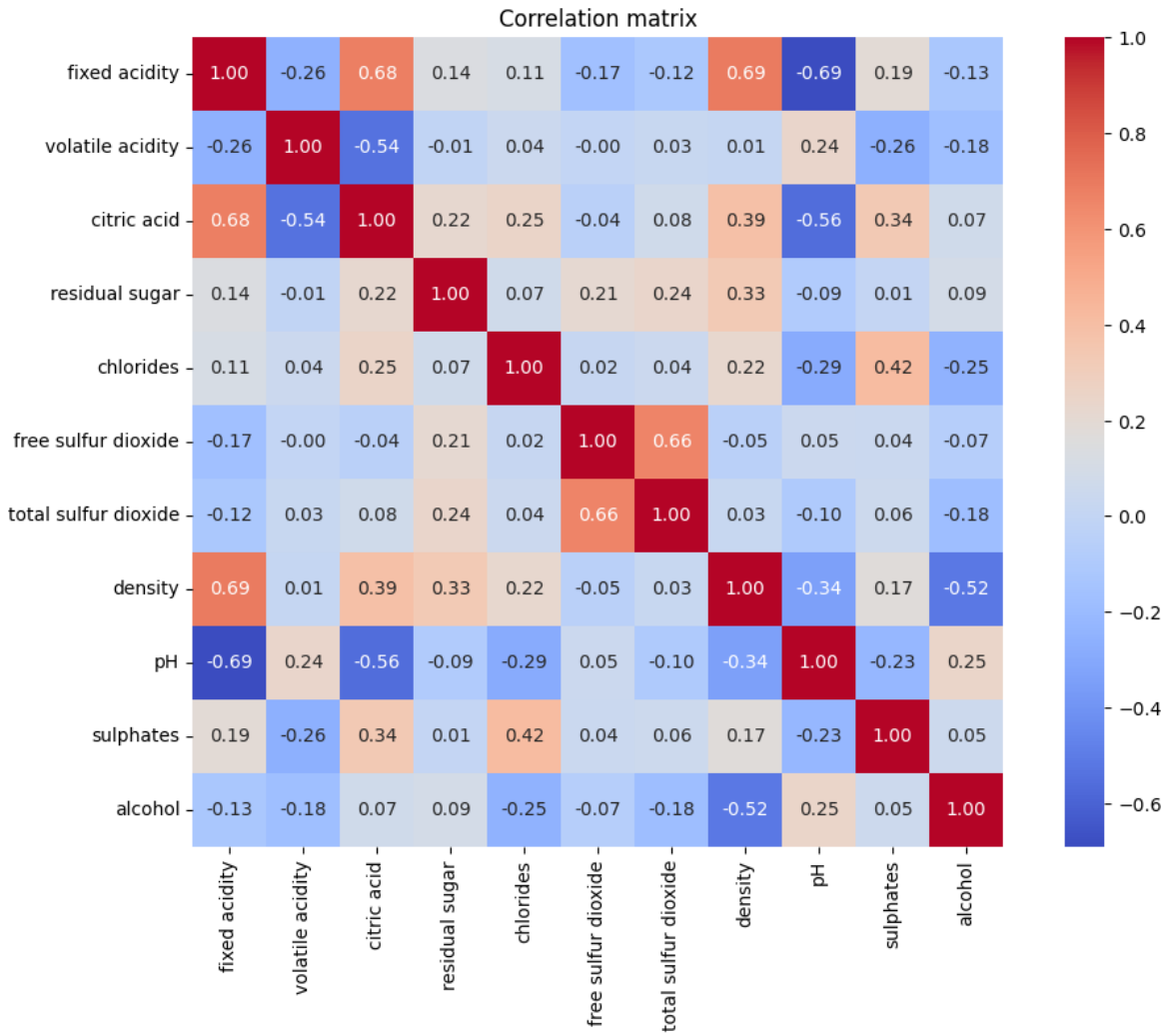
I decided to see how the data is distributed for each feature, to observe if the data is skewed or unbalanced (even if the random forest is not necessarily significantly impacted by the data distribution)



After dividing the dataset into train and test (0.3 test and 0.7 train), I decided to analyze the correlations between the data, thus I noticed that the "fixed acidity" feature is correlated directly proportionally with "citric acid" and "density" and inversely proportionally with "pH"

```
X = df_wine.drop(['quality','Id'], axis=1)
y = df_wine['quality']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42

correlation_matrix = X_train.corr()
plt.figure(figsize=(12, 8))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt=".2f", square=True)
plt.title('Correlation matrix')
```

```
plt.show()
```



Researching the distributions, I noticed that the classes are extremely unbalanced, so I decided to use an oversampling method on the training set for the classes with few samples and to eliminate the class with the fewest samples (class 3 with only 6 samples) . Also at this step I encoded the target values.

```
label_encoder = LabelEncoder()
y_train = label_encoder.fit_transform(y_train)
y_test = label_encoder.transform(y_test)
```

```
smote = SMOTE(sampling_strategy={0:46,3:200,4:20}, random_state=42)
X_resampled, y_resampled = smote.fit_resample(X_train, y_train)
```

Since we can see that the data follows a certain normal distribution, I decided to scale the data according to the standard deviation (Standardization), even if the impact for Random Forest is not significant.

```
scaler = StandardScaler()
column_names = X.columns
X_train = pd.DataFrame(scaler.fit_transform(X_train), columns=column_names)
X_resampled = pd.DataFrame(scaler.fit_transform(X_resampled), columns=column_names)
X_test = pd.DataFrame(scaler.transform(X_test), columns=column_names)
```

After preparing the data, we performed a grid search to find out the best combination of hyperparameters to maximize accuracy and minimize overfeeting.

```
param_grid = {
    'n_estimators': [75, 100, 125],
    'max_depth': [30, 40, 50, None],
    'max_features': ['sqrt', 'log2', None],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4],
    'criterion': ['gini', 'entropy'],
    'class_weight': ['balanced', 'balanced_subsample', {1:1,2:1,3:2,0:3,4:5},None],
}

base_model = RandomForestClassifier(random_state=42, oob_score=True)


grid_search = GridSearchCV(base_model, param_grid=param_grid,
                           cv=3,
                           scoring='accuracy',
                           verbose=2,
                           n_jobs=-1)

grid_search.fit(X_resampled, y_resampled)
best_rf_model = grid_search.best_estimator_
rf_predictions = best_rf_model.predict(X_test)

accuracy_score_rf = accuracy_score(y_test, rf_predictions)
print(f"Accuracy: {accuracy_score_rf}, OOB SCORE: {best_rf_model.oob_score_}")

conf_matrix = confusion_matrix(y_test, rf_predictions)
print("Confusion Matrix:")
print(conf_matrix)

print("Best Hyperparameters Found:")
print(grid_search.best_params_)
```

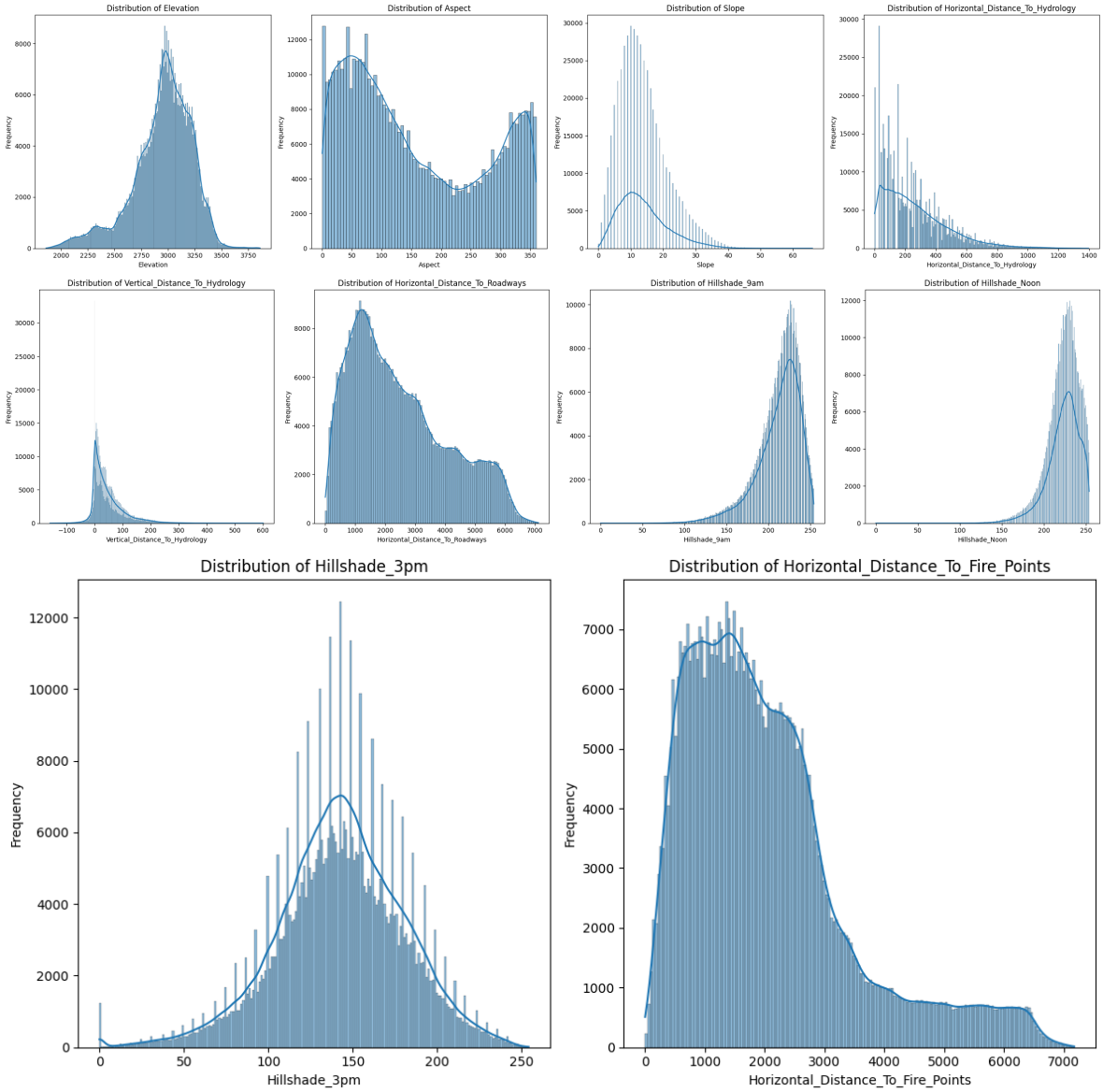
As a result, we obtained an accuracy of 0.64 and an OOB score of 0.67, which shows a slight overfitting. I will discuss the evaluation, improvement and testing of various approaches in more detail in the evaluation part.

4.2 Forest Cover Dataset

In the same way as with the small dataset, I loaded the dataset into the DataFrame, and analyzed the data a little in the same way, less the categorical data.

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

df_cov = pd.read_csv('covtype.csv')
df_cov.iloc[:,:10].describe() #Numerical
df_cov.iloc[:,10:].describe() #Categorical

import seaborn as sns

max_plots_per_row = 4
numeric_columns = [col for col in df_cov.iloc[:,:10] if df_cov.iloc[:,:10][col].dtype in
```

```
for i in range(0, len(numeric_columns), max_plots_per_row):
    cols_to_plot = numeric_columns[i:i + max_plots_per_row]
    fig, axes = plt.subplots(1, len(cols_to_plot), figsize=(6 * len(cols_to_plot), 6))
    if len(cols_to_plot) == 1:
        axes = [axes]
    for ax, col in zip(axes, cols_to_plot):
        sns.histplot(df_cov.iloc[:,:10][col], kde=True, ax=ax)
        ax.set_title(f'Distribution of {col}')
        ax.set_xlabel(col)
        ax.set_ylabel('Frequency')
    plt.tight_layout()
    plt.show()
```

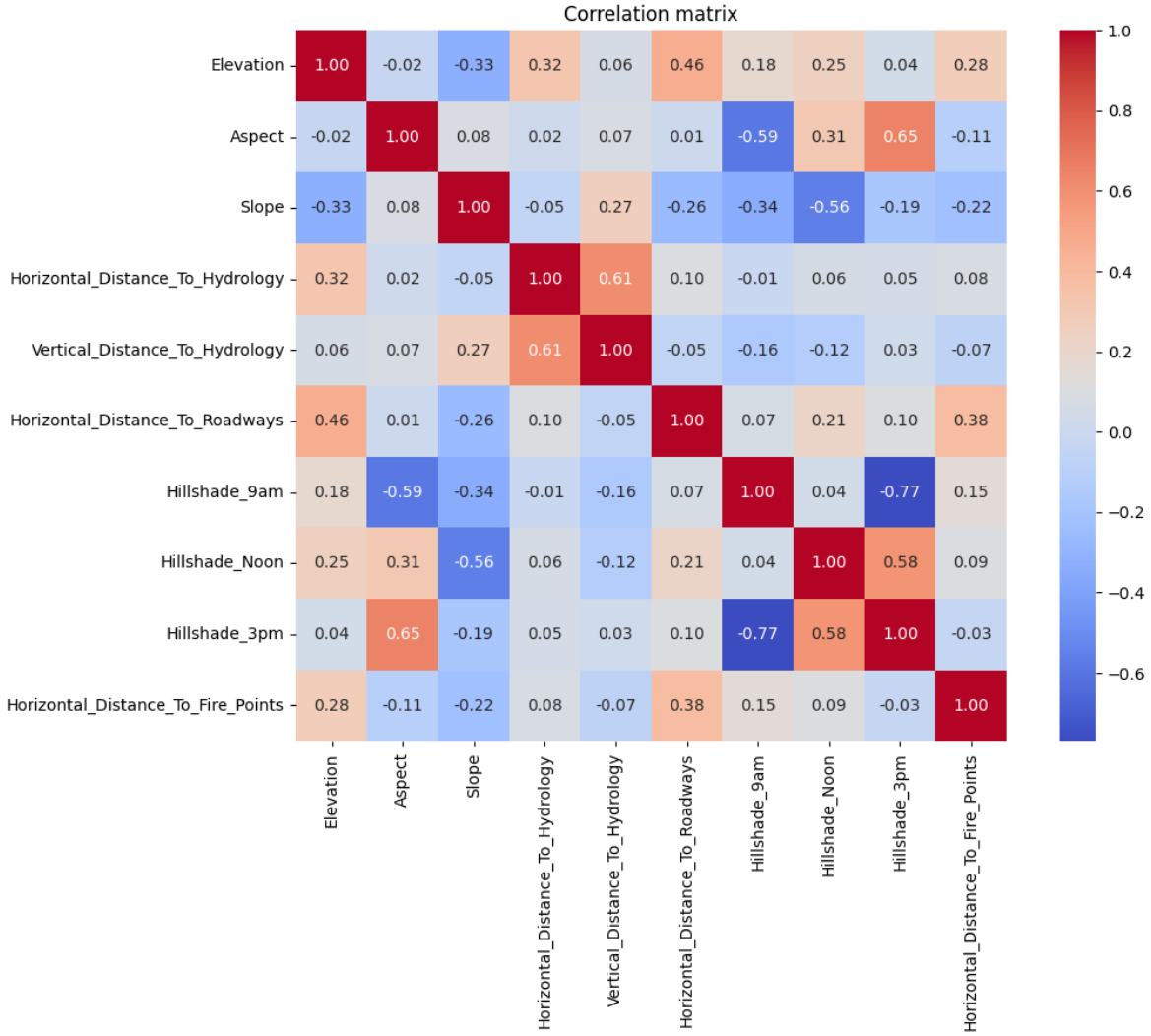


Later, I also analyzed the target which was unbalanced in the same way, so after dividing the data in train and test (0.7 and 0.3), and encoding the target, I decided to apply the same resampling method, SMOTE, to balance the dataset.

```
from sklearn.model_selection import train_test_split
```

```
from sklearn.metrics import classification_report
from sklearn.ensemble import RandomForestClassifier
from imblearn.over_sampling import SMOTE
from sklearn.metrics import accuracy_score, confusion_matrix
from sklearn.ensemble import RandomForestClassifier
from sklearn.preprocessing import LabelEncoder

X = df_cov.drop(['Cover_Type'], axis=1)
y = df_cov['Cover_Type']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42

label_encoder = LabelEncoder()
y_train = label_encoder.fit_transform(y_train)
y_test = label_encoder.transform(y_test)

smote = SMOTE(sampling_strategy={3:6000,4:20000,5:36000,6:42000,2:75000,0:198310}, rando
X_resampled, y_resampled = smote.fit_resample(X_train, y_train)
```

The correlation matrix for the numerical data shows a moderate correlation between the hillshade data and also a correlation between "Aspect" and "Hillshade 3pm" Regarding the random forest model, I



could not apply the same strategy of running GridSearch because the size of the dataset is superior in

all aspects, so I tried to constantly adjust the parameters after several runs.

```
base_model = RandomForestClassifier(random_state=42,class_weight="balanced",oob_score=Tr
                                    n_estimators=140,criterion="entropy",max_depth=40,mi
                                    min_samples_split=5)
base_model.fit(X_train, y_train)
rf_predictions = base_model.predict(X_test)
accuracy_score_rf = accuracy_score(y_test, rf_predictions)
print(f"Accuracy:{accuracy_score_rf}",f"OOB_Score:{base_model.oob_score_}")
conf_matrix = confusion_matrix(y_test, rf_predictions)

correct_predictions_per_class = conf_matrix.diagonal()
total_predictions_per_class = conf_matrix.sum(axis=1)

print("\nNumber of Correct Predictions per Class:")
for class_idx, correct in enumerate(correct_predictions_per_class):
    print(f"Class {class_idx}: {correct} out of {total_predictions_per_class[class_idx]}

print("\nAccuracy per Class:")
for class_idx, (correct, total) in enumerate(zip(correct_predictions_per_class, total_pr
    accuracy = correct / total if total > 0 else 0
    print(f"Class {class_idx}: {accuracy:.2%}")
```

5 Evaluation and improvement

5.1 Wine Dataset

Regarding the random forest model for this data set, I tried multiple methods to help the hyperparameters because the accuracy around the minority classes was always close to 0.

```
%%time
base_model = RandomForestClassifier(random_state=42,class_weight={1:1,2:1,3:2,0:3,4:5},o
                                    n_estimators=125,criterion="gini",max_depth=30,max_f
                                    min_samples_split=2)
base_model.fit(X_resampled, y_resampled)
rf_predictions = base_model.predict(X_test)
accuracy_score_rf = accuracy_score(y_test, rf_predictions)
print(f"Accuracy:{accuracy_score_rf}",f"OOB_Score:{base_model.oob_score_}")
conf_matrix = confusion_matrix(y_test, rf_predictions)

print("Confusion Matrix:")
print(conf_matrix)

num_classes = len(set(y_test))
tp = np.zeros(num_classes)
fp = np.zeros(num_classes)
fn = np.zeros(num_classes)
tn = np.zeros(num_classes)

for i in range(num_classes):
    print(f"Class {i}:")
    print(f"    True Positives (TP): {tp[i]}")
    print(f"    False Negatives (FN): {fn[i]}")
    print(f"    False Positives (FP): {fp[i]}")
    print(f"    True Negatives (TN): {tn[i]}")

correct_predictions_per_class = conf_matrix.diagonal()
```

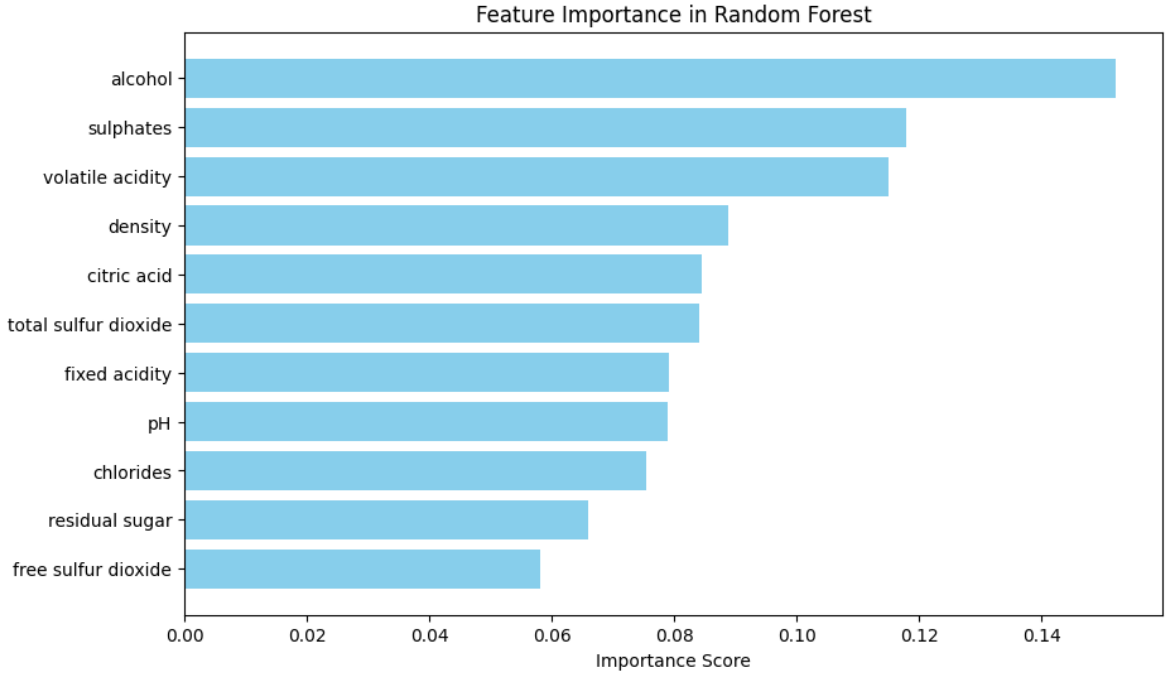
```
total_predictions_per_class = conf_matrix.sum(axis=1)

print("\nNumber of Correct Predictions per Class:")
for class_idx, correct in enumerate(correct_predictions_per_class):
    print(f"Class {class_idx}: {correct} out of {total_predictions_per_class[class_idx]}


print("\nAccuracy per Class:")
for class_idx, (correct, total) in enumerate(zip(correct_predictions_per_class, total_pr
    accuracy = correct / total if total > 0 else 0
    print(f"Class {class_idx}: {accuracy:.2%}")
Accuracy: 0.6461988304093568 OOB_Score: 0.6481271282633371
Confusion Matrix:
[[  0   7   3   0   0]
 [  0 129  11   5   0]
 [  0  62  73   4   0]
 [  0   4  20  19   0]
 [  0   1   2   2   0]]
Class 0:
    True Positives (TP): 0.0
    False Negatives (FN): 10.0
    False Positives (FP): 0.0
    True Negatives (TN): 332.0
Class 1:
    True Positives (TP): 129.0
    False Negatives (FN): 16.0
    False Positives (FP): 74.0
    True Negatives (TN): 123.0
Class 2:
    True Positives (TP): 73.0
    False Negatives (FN): 66.0
    False Positives (FP): 36.0
    True Negatives (TN): 167.0
Class 3:
    True Positives (TP): 19.0
    False Negatives (FN): 24.0
    False Positives (FP): 11.0
    True Negatives (TN): 288.0
Class 4:
    True Positives (TP): 0.0
    False Negatives (FN): 5.0
    False Positives (FP): 0.0
    True Negatives (TN): 337.0

Number of Correct Predictions per Class:
Class 0: 0 out of 10
Class 1: 129 out of 145
Class 2: 73 out of 139
Class 3: 19 out of 43
Class 4: 0 out of 5

Accuracy per Class:
Class 0: 0.00%
Class 1: 88.97%
Class 2: 52.52%
Class 3: 44.19%
```

```
Class 4: 0.00%
CPU times: user 407 ms, sys: 0 ns, total: 407 ms
Wall time: 405 ms
```

As we can see, after running the grid search from the first part and recreating the model with the parameters that make the accuracy more efficient, the results are reasonable, especially because the OOB score and the accuracy are at an almost equivalent value, which gives us the idea that the model generalizes quite well. However, the high value of the TN values suggests a low recall value, so I tried another similar variant in which I focused on increasing the recall value using "recall" as the scoring method for a possible improvement of the model , thus we also obtained a changed series of hyper parameters for the model.

```
Fitting 3 folds for each of 2592 candidates, totalling 7776 fits
/usr/local/lib/python3.11/dist-packages/sklearn/model_selection/_search.py:1107: UserWar
  warnings.warn(
Accuracy: 0.5964912280701754,OOB_SCORE:0.630188679245283
Confusion Matrix:
[[  0  10   0   0   0]
 [  0 132  12   1   0]
 [  0  72  66   1   0]
 [  0   6  31   6   0]
 [  0   2   2   1   0]]
Best Hyperparameters Found:
{'class_weight': 'balanced', 'criterion': 'gini', 'max_depth': 30, 'max_features': 'sqrt
CPU times: user 16.7 s, sys: 1.36 s, total: 18.1 s
Wall time: 53 s
```

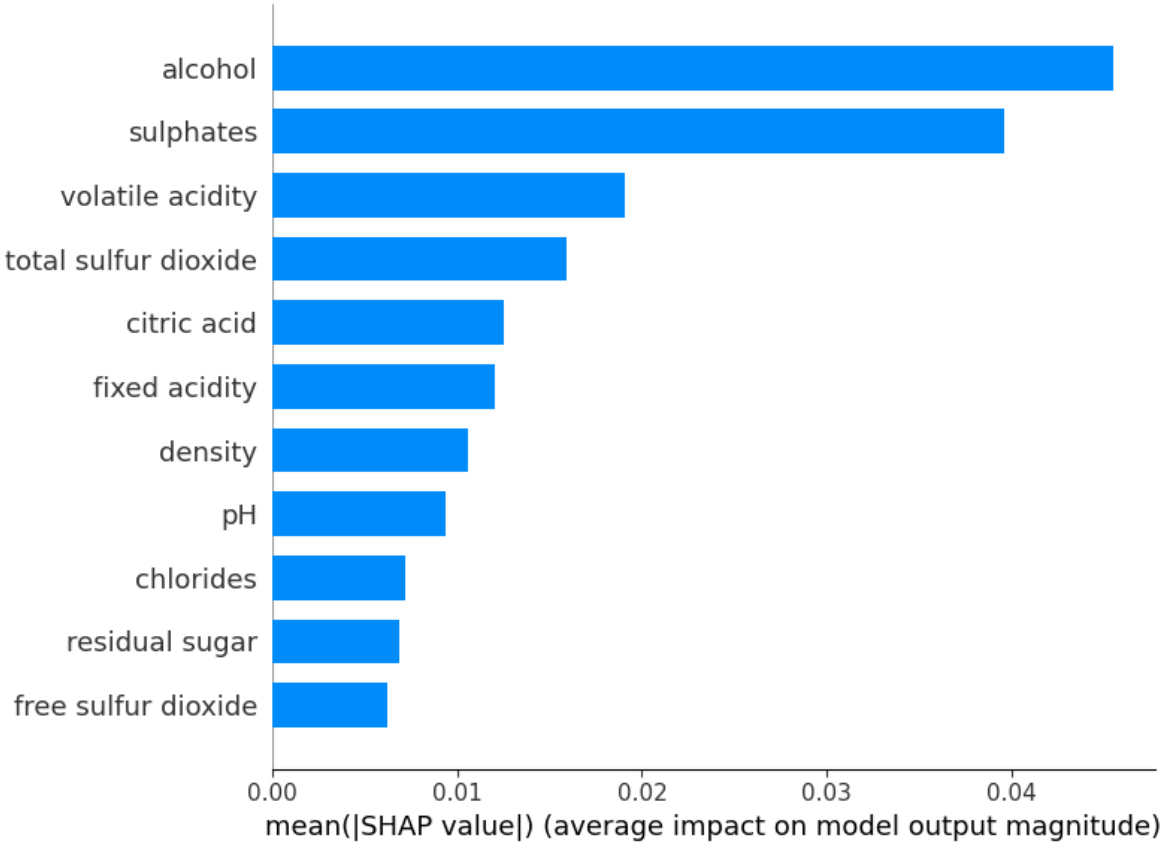
Unfortunately, the result was an undesirable one by increasing the false positive values, thus the accuracy being affected in the end. In order to better understand the initially chosen model, I decided to check which are the most important features for it using initial feature importance.



I considered that removing any feature could affect the quality of the model because it is trained on a fairly small data set, so I kept all features even if some were quite correlated.

In order to see the importance of the models in more detail, I decided to also calculate the Shapley values, which above the feature importance, give us a perspective of the importance of the features for

each class, also taking into account the correlations between the variables.



After calculating the Shapley values, we also considered the elimination of the most unimportant 6 features, but the results were not surprising, the accuracy was not affected, instead the OOB score increased, thus showing a small overfitting.

```
Accuracy: 0.6432748538011696 OOB_Score: 0.6549375709421112
```

5.2 Forest Cover Dataset

The large dataset, on the other hand, proved to give much better results even after several attempts.

```
base_model = RandomForestClassifier(random_state=42,class_weight="balanced",oob_score=Tr
                                    n_estimators=140,criterion="entropy",max_depth=40,mi
                                    min_samples_split=5)
base_model.fit(X_train, y_train)
rf_predictions = base_model.predict(X_test)
accuracy_score_rf = accuracy_score(y_test, rf_predictions)
print(f"Accuracy: {accuracy_score_rf}",f"OOB_Score: {base_model.oob_score_}")
conf_matrix = confusion_matrix(y_test, rf_predictions)

correct_predictions_per_class = conf_matrix.diagonal()
total_predictions_per_class = conf_matrix.sum(axis=1)

print("\nNumber of Correct Predictions per Class:")
for class_idx, correct in enumerate(correct_predictions_per_class):
    print(f"Class {class_idx}: {correct} out of {total_predictions_per_class[class_idx]}
```

```
print("\nAccuracy per Class:")
for class_idx, (correct, total) in enumerate(zip(correct_predictions_per_class, total_pr
    accuracy = correct / total if total > 0 else 0
    print(f"Class {class_idx}: {accuracy:.2%}")

Number of Correct Predictions per Class:
Class 0: 55874 out of 63552
Class 1: 76046 out of 84991
Class 2: 8244 out of 10726
Class 3: 389 out of 824
Class 4: 1262 out of 2848
Class 5: 2900 out of 5210
Class 6: 5439 out of 6153

Accuracy per Class:
Class 0: 87.92%
Class 1: 89.48%
Class 2: 76.86%
Class 3: 47.21%
Class 4: 44.31%
Class 5: 55.66%
Class 6: 88.40%
CPU times: user 2min 15s, sys: 599 ms, total: 2min 16s
Wall time: 2min 15s
```

After adjusting the number of trees, we have already noticed a significant increase in accuracy, more precisely the reduction to 125 estimators.

```
feature_importances = base_model_2.feature_importances_

features_df = pd.DataFrame({
    'Feature': X_train.columns,
    'Importance': feature_importances
}).sort_values(by='Importance', ascending=False)

features_df
```

After rendering the most important features and another small adjustment of the number of trees to 145, I decided to choose the first 10 most important features, thus finally obtaining an accuracy of 0.95.

```
Accuracy: 0.9551358545988617 OOB_Score: 0.9692627080365519
```

6 Conclusion

This project aimed to build a classification model using the Random Forest algorithm, applied to two datasets: Wine Quality and Forest Cover. The Random Forest model was chosen for its ability to handle high-dimensional data, model complex non-linear relationships, and minimize overfitting through bootstrap sampling and majority voting across multiple decision trees.

Comprehensive exploratory data analysis (EDA) provided insights into feature distributions, correlations, and class imbalances present in both datasets. Based on these findings, hyperparameter tuning was performed using GridSearchCV to optimize model performance, with a specific emphasis on improving the recall metric to better capture minority classes.

The model's effectiveness was evaluated using various performance metrics, including accuracy, precision, recall, F1-score, and confusion matrices. For the Wine Quality dataset, an accuracy of 0.64 was achieved, while the Forest Cover dataset resulted in an accuracy of 0.95. The OOB (Out-of-Bag) score, closely aligned with the accuracy on test data, confirmed the model's ability to generalize well without significant overfitting.

Analysis of prediction outcomes showed a high number of True Negatives, indicating opportunities for further refinement using class weighting or resampling methods to improve balance across predictions. This approach would enhance the detection of less frequent classes without compromising overall performance.

Looking forward, implementing more advanced techniques, such as boosting or stacking ensembles, could further improve classification accuracy and robustness. In conclusion, the Random Forest model delivered reliable and interpretable results, demonstrating its versatility and strength in tackling multi-class classification tasks across diverse datasets.