Introduction:

This study focuses on hyperparameter optimization to improve machine learning models using the white wine quality dataset. Our goal is to identify the best combination of algorithms and hyperparameters for predicting wine quality accurately. We aim to optimize predictive models for assessing wine quality by exploring machine learning algorithms (like Random Forest and Gradient Boosting) and tuning their hyperparameters. Through systematic experimentation, we seek to enhance model accuracy and improve wine quality predictions.

Our experimental setup involves defining a comprehensive search space for hyperparameters and utilizing advanced optimization techniques such as random search and Bayesian optimization. By rigorously evaluating model performance and using statistical tests, we aim to ensure unbiased and robust results. Through this study, we aim to identify optimal algorithm-hyperparameter configurations that yield the highest predictive accuracy for wine quality assessment.

Data Description: The dataset consists of the data of the white variant of the Portuguese "Vinho Verde" wine. The dataset provides valuable insights into the chemical properties of white wine variant and its quality ratings. It comprises of 12 different parameters named as fixed acidity, volatile acidity, citric acid, residual sugar, chlorides, free sulfur dioxide, total sulfur dioxide, density, pH, sulphates, alcohol and quality. We checked for the shape of the data to find out the number of rows and columns present in the data by using shape() function. The dataset consists of 4898 rows and 12 columns. We checked for the null values using isnull().sum() function. The dataset didn't have any null values. In this exercise, I will compare the performance of Random Forest and Gradient boosting machine learning algorithms and will also find out the most optimum value for the hyperparameters on this dataset and will eventually report the best performing algorithm. The target variable or feature will be a binary class of good or bad wine quality where we would be classifying the quality value greater than equal to 7 to be of good quality and rest would be classified as bad quality.

Experimental setup: The programming language used for this experiment will be python. We begin with importing a few libraries named as pandas, numpy, matplotlib.pyplot and seaborn. Pandas library helps in data manipulation and data analysis, numpy helps in performing operations on arrays, matplotlib.pyplot and seaborn are used for plotting. We begin the experiment by uploading our dataset using the pd.read_csv function. As the data was uploaded, we found out that all the columns and values were present in a merged form in a single column. Hence to separate the data in different columns we used a separator, which was semi-colon (";") in our case, along with the pd.read_csv function. After this we used head() function to view the values of the first few rows of the uploaded data. Once we have confirmed that this is the right dataset we begin with our initial investigation of the data. For this firstly we used shape() function to know the dimensionality of the data. By dimensionality we mean the number of rows and columns present in the dataset. Following this we used the info() function to get the information of all data types present in the dataset. After this we used the isnull function to find out the missing values in the dataset. Once we have figured out that the dataset doesn't have any missing values now, we will find out the correlation between different features present in our dataset. To find the correlation

we have used corr() function and we have chosen ‘pearson’ method for that. The correlation has been plotted using the sns.heatmap() function. After this the parameters which were highly correlated (i.e. having a correlation value greater than 0.5) were removed. We have used drop() function to remove the parameters that were highly correlated. We then plotted the non-correlated parameters using sns.heatmap() function. Now we have used describe() function to find out the general statistical characteristics of our dataset. After this we have now divided our data into two parts. First is the input data which consists of all the input features and the second is target data which consists of the target feature. For the target feature we have performed label binarization i.e. we have classified the label which is ‘quality’ in our case into a binary class where quality value greater than equal to 7 has been labeled as ‘Good’ and lower values have been labeled as ‘Bad’. For label binarization we have used apply(lambda y_value:'Good' if y_value>=7 else 'Bad') function. We are now using MinMax scaler to normalize the data. For this we have first imported the MinMax scaler from sklearn.preprocessing library. After this we used fit_transform() function to fit the transformation on our input data. After the preprocessing steps now, we will apply machine learning algorithms on our preprocessed dataset along with different hyperparameter optimization methods and the procedure for the same has been described below:

a) **Random forest without hyperparameter tuning:** We considered our problem as a Binary-class classification and began importing the required modules like Random Forest classifier, Cross validation score and accuracy score from the sklearn library. Now we have fitted the random forest model on the input and target variables using the fit() function. We have now performed K-fold cross-validation where K=5 for our dataset. The general working of K-fold cross validation can be explained as a single parameter called k that refers to the number of groups that a given data sample is to be split into. When a specific value for k is chosen, it may be used in place of k in reference to the model, such as K=5 becoming 5-fold cross-validation. If K=5 the dataset will be divided into 5 equal parts and the process mentioned next will run 5 times, each time with a different holdout set. Firstly, take the group as a holdout or test data set. Secondly, take the remaining groups as a training data set. Thirdly, fit a model on the training set and evaluate it on test set and finally retain the evaluation score and discard the model. At the end of the process summarize the skill of the model using the sample of model evaluation scores which is “accuracy score” in our case. As can be inferred from the process of cross validation we get 5 accuracy score values for each fold or each iteration and finally by taking a mean of these accuracy scores we get a mean accuracy score. We have used get_params() function to get the default hyperparameters for our Random Forest model and the same has been mentioned in the results section.

b) **Random forest with Random search for hyperparameter optimization:** As already mentioned above our problem is a binary classification problem. We began with importing necessary modules like Random Forest classifier from sklearn.ensemble library, RandomizedSearchCV from sklearn.model_selection library, accuracy_score from

sklearn.metrics library and randint,uniform from scipy.stats library. After this we start with defining our hyperparameter ranges. We have only considered the hyperparameters that have numerical values in our experiment. We begin with 'n_estimators' which has a range (10, 100). We have used 'randint' function here for the range which means that for calculation the model will select a random integer value each time within the described range i.e. a random integer is selected between 10 and 100 each time the iteration is repeated. The range for 'max_depth' was (3,10) and here also we used 'randint' function. The range for 'min_sample_split' was (2,20) and 'randint' was used here as well. 'min_sample_leaf" varied between (1,20) and 'randint' was used here as well for selecting the value. The range for 'max_leaf_nodes' was (10,100) along with 'randint' as a selection method. The range for 'n_jobs' was (0,100) and 'randint' was used for selection. The range for 'max_samples' was (0.1,0.9). Here we have used the uniform function for selection, this will ensure that there is an equal probability of selecting any value between the defined range i.e. 0.1 to 0.9 for 'max_samples' for each of the iterations performed. After this we initialize the random forest classifier model. We now perform the random search for the hyperparameters using RandomizedsearchCV() function. We have now performed K-fold cross-validation where K=5 for our dataset. Now we have fitted the random forest model on the input and target variables using the fit() function. We have then identified the best hyperparameters for our dataset and have used mean accuracy score as the evaluation matric and values for hyperparameters and mean accuracy is mentioned in the result section.

c) **Random forest with Bayesian optimization for hyperparameter optimization:** As the Bayesian optimization package was not installed initially with google colab notebook we began with installing the package for the same by using pip install bayesian-optimization. We began with importing necessary modules like Random Forest classifier from sklearn.ensemble library, Bayesianoptimization from bayes_opt library, cross_val_score from sklearn.model_selection and accuracy_score from sklearn.metrics library. After this we begin with defining an objective function 'rf_cv' that we want to optimize using bayesian optimization. The function takes hyperparameters (n_estimators, max_depth, min_samples_split, min_samples_leaf, max_leaf_nodes, max_samples) as input. The hyperparameters are converted to integers where necessary. Inside the function, a Random Forest Classifier model (rf_model) is initialized with the provided hyperparameters. K=5 Cross-validation is performed using the initialized model. The mean accuracy score from cross-validation is returned as the evaluation metric. Once the function is defined, we will now define the search space for different hyperparameters. The search spaces are 'n_estimators' (10, 100), 'max_depth' (3, 10), 'min_samples_split' (2, 20), 'min_samples_leaf' (1, 20), 'max_leaf_nodes' (10, 100), 'max_samples' (0.1, 1.0). Here also we have considered only the hyperparameters whose value vary numerically. After this we initialize Bayesian Optimization with the objective function 'rf_cv' and the search space for the hyperparameters. After this er have used maximize() function for bayesian optimization to maximize the objective function 'rf_cv'. We have then identified the best

hyperparameters for our dataset and have used mean accuracy score as the evaluation matric and values for hyperparameters and mean accuracy is mentioned in the result section.

d) **Gradient boosting without hyperparameter optimization:** We began by importing the required modules like Gradient Boosting Classifier, Cross validation score and accuracy score from the sklearn library. Now we have fitted the gradient boosting classifier model on the input and target variables using the fit() function. We have now performed K-fold cross-validation where K=5 for our dataset. The general working of k-fold cross validation is same as described in random forest model section. Here also we get mean accuracy score and default hyperparameter using the same procedure as used for random forest model and same has been reported in the results section.

e) **Gradient Boosting with Random Search for hyperparameter Optimization:** We began with importing necessary modules like Gradient Boosting Classifier from sklearn.ensemble library, RandomizedSearchCV from sklearn.model_selection library, accuracy_score from sklearn.metrics library and randint,uniform from scipy.stats library. After this we start with defining our hyperparameter ranges. We have only considered the hyperparameters that have numerical values in our experiment. We begin with 'n_estimators' which has a range (50, 200). We have used 'randint' function here for the range which means that for calculation the model will select a random integer value each time within the described range i.e. a random integer is selected between 50 and 200 each time the iteration is repeated. The range for 'max_depth' was (3,10) and here also we used 'randint' function. The range for 'min_sample_split' was (2,20) and 'randint' was used here as well. 'min_sample_leaf" varied between (1,20) and 'randint' was used here as well for selecting the value. The range for 'learning_rate' was (0.01,0.3). Here we have used the uniform function for selection, this will ensure that there is an equal probability of selecting any value between the defined range i.e. 0.01 to 0.3 for 'learning_rate' for each of the iterations performed. After this we initialize the gradient boosting classifier model. We now perform the random search for the hyperparameters using RandomizedsearchCV() function. We have now performed K-fold cross-validation where K=5 for our dataset. Now we have fitted the gradient boosting model on the input and target variables using the fit() function. We have then identified the best hyperparameters for our dataset and have used mean accuracy score as the evaluation matric and values for hyperparameters and mean accuracy is mentioned in the result section.

f) **Gradient boosting with Bayesian optimization for hyperparameter optimization:** We began with importing necessary modules like Gradient Boosting Classifier from sklearn.ensemble library, Bayesianoptimization from bayes_opt library, cross_val_score from sklearn.model_selection and accuracy_score from sklearn.metrics library. After this we begin with defining an objective function 'gb_cv' that we want to optimize using bayesian optimization. The function takes hyperparameters (n_estimators, max_depth, min_samples_split, min_samples_leaf, learning_rate) as input. The hyperparameters are converted to integers where necessary. Inside the function, a Gradient Boosting Classifier model (gb_model) is initialized with the provided hyperparameters. K=5 Cross-validation is performed using the initialized model. The mean accuracy score from cross-validation

is returned as the evaluation metric. Once the function is defined, we will now define the search space for different hyperparameters. The search spaces are 'n_estimators' (50, 200), 'max_depth' (3, 10), 'min_samples_split' (2, 20), 'min_samples_leaf' (1, 20), and 'learning_rate' (0.01, 0.3). Here also we have considered only the hyperparameters whose value vary numerically. After this we initialize Bayesian Optimization with the objective function 'gb_cv' and the search space for the hyperparameters. After this we used maximize() function for bayesian optimization to maximize the objective function 'gb_cv'. We have then identified the best hyperparameters for our dataset and have used mean accuracy score as the evaluation matric and values for hyperparameters and mean accuracy is mentioned in the result section.

Results: The different results obtained in our experiment are mentioned in this section. The very first result came from using the shape() function which shows us that the data consists of 4898 rows and 12 columns. The info() function shows that all the 12 columns have 4898 data points, and we have 11 float data types and 1 integer data type columns. The isnull() function shows that there were no missing values in our dataset. Now we have found out the correlation between different input features and the same has been plotted in figure 1 present in appendix. As can be seen from figure 1, density and residual sugar are having high correlation, hence we are removing density from our study. Similarly, total sulfur dioxide and free sulfur dioxide have a high correlation, so total sulfur dioxide is removed from the study. The updated parameters with non-correlative parameters are again plotted and can be seen in figure 2 present in appendix. The describe() function now gives us the statistical characteristics of the data like counts, mean, standard deviation etc. Now we split the data into input and target features. The input features are fixed acidity, volatile acidity, citric acid, residual sugar, chlorides, free sulfur dioxide, pH, sulphates, and alcohol, while the target feature is wine quality. After label binarization the target variable was modified and now labels the data as 'Good' or 'Bad'. After this we normalized the data using MinMax scaler so now all the input features are normalized and have values ranging between 0 and 1. After the normalization of input data we used Random Forest and Gradient Goosting algorithms to make our prediction. We have used Random search and Bayesian optimization for hyperparameter optimization, and the mean accuracy scores obtained from different models have been mentioned in the table below.

ML Algorithm	**Accuracy with default Hyperparameters**	**Accuracy by using Random Search in HPO**	**Accuracy by using Bayesian Optimization in HPO**
Random Forest	0.8078	0.8074	0.8074
Gradient Boosting	0.7984	**0.8105**	0.8050

The above table shows that the highest accuracy was obtained by using Random Search for hyperparameter optimization with gradient boosting algorithm. Hyperparameters obtained for each case have also been reported in the appendix section below.

References:

1) Cortez,Paulo, Cerdeira,A., Almeida,F., Matos,T., and Reis,J.. (2009). Wine Quality. UCI Machine Learning Repository. https://doi.org/10.24432/C56S3T.
2) "Detect and Remove the Outliers using Python", https://www.geeksforgeeks.org/detect-and-remove-the-outliers-using-python/
3) "How to use seaborn plotting", https://www.geeksforgeeks.org/python-seaborn-tutorial/
4) "Normalizing the data by use of scaler", https://www.geeksforgeeks.org/data-pre-processing-wit-sklearn-using-standard-and-minmax-scaler/
5) Z-score for normalization, https://medium.com/@TheDataGyan/day-8-data-transformation-skewness-normalization-and-much-more-4c144d370e55
6) Cross Validation Explained: Evaluating estimator performance. Improve your ML model using cross validation. https://towardsdatascience.com/cross-validation-explained-evaluating-estimator-performance-e51e5430ff85
7) Sklearn API Reference. https://scikit-learn.org/stable/modules/classes.html#module-sklearn.ensemble
8) A Beginner's Guide to Random Forest Hyperparameter Tuning https://www.analyticsvidhya.com/blog/2020/03/beginners-guide-random-forest-hyperparameter-tuning/
9) Bayesian optimization https://www.analyticsvidhya.com/blog/2021/05/bayesian-optimization-bayes_opt-or-hyperopt/

Appendix

ML Algorithm	Default hyperparameters	Best hyperparameters from Random Search	Best hyperparameters from Bayesian Optimization
Random Forest	{'bootstrap': True, 'ccp_alpha': 0.0, 'class_weight': None, 'criterion': 'gini', 'max_depth': None, 'max_features': 'sqrt',	'max_depth': 8, 'max_leaf_nodes': 96, 'max_samples': 0.5687508340232413, 'min_samples_leaf': 8, 'min_samples_split': 17, 'n_estimators': 91, 'n_jobs': 52	'max_depth': 6.023615130494811, 'max_leaf_nodes': 36.21062261782377, 'max_samples': 0.6506676052501416, 'min_samples_leaf': 3.6503833523887947, 'min_samples_split': 7.258603673633926, 'n_estimators': 42.97256589643225

	'max_leaf_nodes': None, 'max_samples': None, 'min_impurity_decrease': 0.0, 'min_samples_leaf': 1, 'min_samples_split': 2, 'min_weight_fraction_leaf': 0.0, 'n_estimators': 100, 'n_jobs': None, 'oob_score': False, 'random_state': None, 'verbose': 0, 'warm_start': False}		
Gradient Boosting	'ccp_alpha': 0.0, 'criterion': 'friedman_mse', 'init': None, 'learning_rate': 0.1, 'loss': 'log_loss', 'max_depth': 3, 'max_features': None, 'max_leaf_nodes': None, 'min_impurity_decrease': 0.0, 'min_samples_leaf': 1, 'min_samples_split': 2, 'min_weight_fraction_leaf': 0.0, 'n_estimators': 100, 'n_iter_no_change': None, 'random_state':	'learning_rate': 0.03930163420191516, 'max_depth': 6, 'min_samples_leaf': 15, 'min_samples_split': 8, 'n_estimators': 57	'learning_rate': 0.11861663446573512, 'max_depth': 9.655000144869414, 'min_samples_leaf': 14.907884894416696, 'min_samples_split': 12.775852715546659, 'n_estimators': 73.40279606636548

	None, 'subsample': 1.0, 'tol': 0.0001, 'validation_fraction': 0.1, 'verbose': 0, 'warm_start': False}		

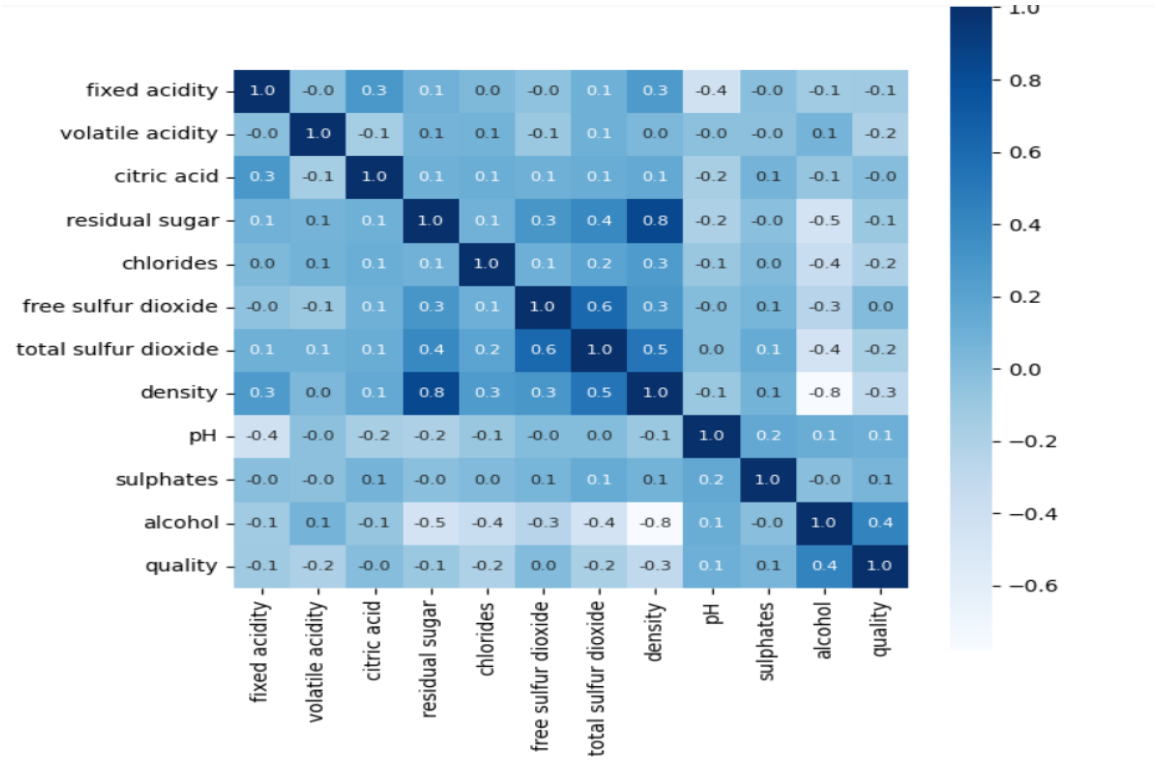

Figure 1. Correlation plot on initial data

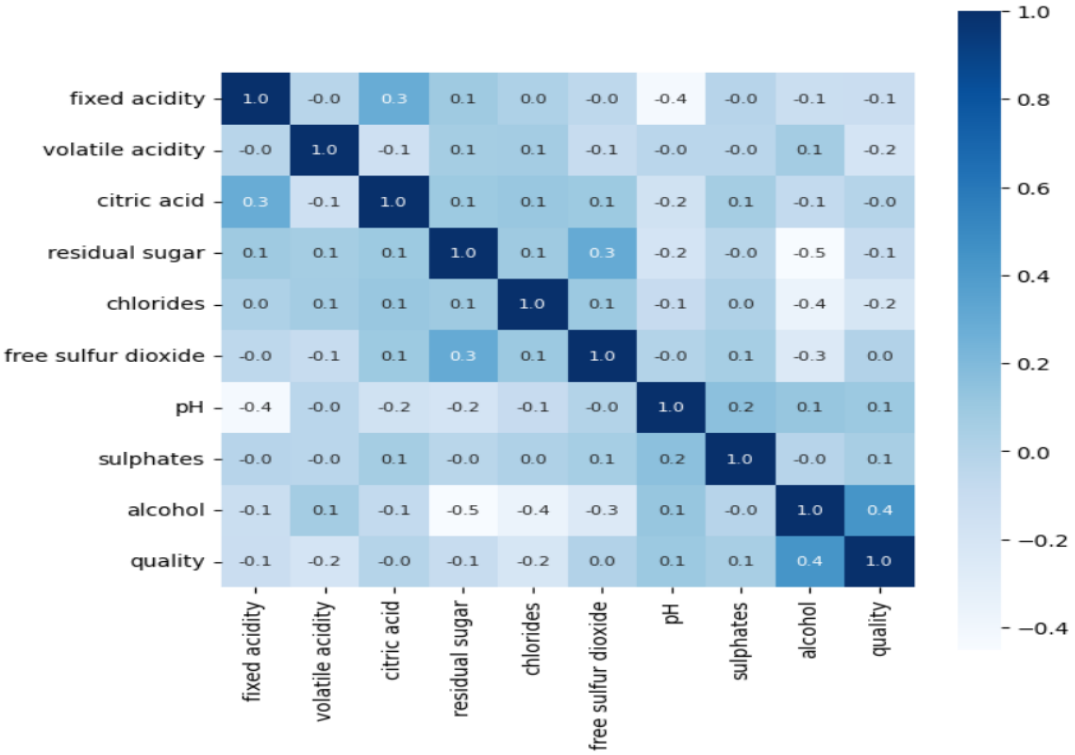

Figure 2. Correlation plot after removing correlated features in the data