**Playground\_Series\_Season3\_Episode5**

**Created by Kaushik Kar**

**Employment id- 2216027**

1. **Importing Libraries:**

First, importing the important external Python packages using the pip package manager.

1. NumPy is used for mathematical operations like addition, subtraction, multiplication, division, etc. on arrays and matrices.
2. Pandas provides data structures for efficiently storing and manipulating large datasets, and tools for reading and writing data to and from various file formats, including CSV, Excel, and SQL databases
3. Seaborn is a data visualization library based on Matplotlib which is a plotting library used for creating static, interactive, and animated visualizations in Python.
4. The import math statement in Python is used to import the math module, which is a built-in module in Python. The math module provides various mathematical functions and constants, such as trigonometric functions, logarithmic functions, and mathematical constants like pi and e.
5. **Upload and Display the data:**

With the help of pandas library we are able to upload the data and with the .head() method we are able to display the data.

here you can see we are using str.replace() to convert the space in the columns into the underscored and again display the data.

1. **EDA:**

Exploratory data analysis has several things such as .info(), .shape. .describe()

* .info() is a method in pandas that can be used to display information about a DataFrame, such as the number of rows and columns, data types, and memory usage. It can help you quickly identify missing values or inconsistencies in the data, and can also give you an idea of the size and complexity of the dataset you're working with.
* .shape is an attribute in pandas that can be used to get the number of rows and columns in a DataFrame. It returns a tuple that contains the number of rows followed by the number of columns, and can be used to quickly check the size and dimensions of a DataFrame.
* The .describe() method provides summary statistics for a DataFrame, such as the count, mean, standard deviation, minimum, and maximum values for each numeric column. It can help you quickly understand the distribution of the data and identify any outliers or unusual values.

1. **Label:**

Here we consider the term ‘Quality’ as label. Here we are try to compare the quality of train and the original data and we can see from the observation that both of them almost same in this result and we can use value\_counts() just to take out the number number of a particular unique values in the data.

the differences between the two distributions are relatively small, indicating that the training dataset is likely a representative sample of the original database.

1. **Distributions:**

We creates a plot to compare the distribution of each feature in the training data, test data, and original data.

The plot is created using seaborn's kdeplot function, which creates a kernel density estimate plot of the data. The plot shows the probability density of each dataset for different values of each feature.

The plot is divided into subplots, with each feature having its own subplot. The number of subplots is determined by the number of features in the dataset. The plot also includes a legend to differentiate between the different datasets.

The purpose of this plot is to help visualize the similarities and differences in the distribution of each feature in the training, test, and original datasets. By comparing the distributions of the features in the training and test datasets, we can get an idea of how well the model will generalize to new data. Comparing the distributions of the features in the training and original datasets can help us determine if the training dataset is representative of the original dataset, and if the model is likely to perform well on new data.

**Train Vs Test vs Original:**

Insights:

At first glance the only discernable difference lies in the distribution of free sulfur dioxide free sulfur dioxide. On the other hand train and test datasets does not reveal any substantial variations, suggesting that there is no need to conduct an extensive adversarial validation..

The distributions of Chlorides and residual sugars exhibit a skewed pattern, which may require rectification through the use of the clip function. This technique can be especially beneficial for machine learning models that rely on feature scaling, as it can mitigate the impact of outliers on model performance.

Concating original dataset is a good idea in this problem.

We can also visualize the feature of the dataset through Boxplots summarize key features of a dataset, including outliers, central tendency, distribution shape, and group comparisons. They are a valuable tool for exploratory data analysis, helping to quickly identify important features of the data.

Insights:

Two features, alcohol and sulphates, have been identified as having a linear relationship with wine quality. This suggests that as the concentration of alcohol and sulphates in a wine increases, the quality of the wine tends to improve.

On the other hand, .density and volatile acidity has been identified as having an inverse relationship with wine quality. This suggests that as the density of a wine increases, the quality of the wine tends to decrease.

Qualities 5 and 6 have more outliers than the rest of the categories. Maybe to clip features is a bad idea and it's better to either add an indicator feature or to use tree-based models.

**Outliers:**

Outliers can have a significant impact on statistical analysis and machine learning models. Outliers are observations that lie far away from the bulk of the data points in a dataset. They can be caused by a variety of factors such as measurement errors, data entry errors, or simply represent rare events.

The code computes the z-score for each feature in the training dataset after grouping the data by quality score, using the groupby() method. The z-score is a measure of how many standard deviations a value is away from the mean. The formula to calculate the z-score is:

z = (x - μ) / σ

where x is the data point, μ is the mean of the distribution, and σ is the standard deviation.

The z-score is then used to identify outliers, which are defined as values that are more than two standard deviations away from the mean. This is done by applying the abs() method to the z-score matrix, and checking which values are greater than or equal to 2. The ge() method stands for "greater than or equal to".

The resulting matrix of True and False values is then grouped again by quality score and summed along the rows to count the number of outliers for each feature and quality score. This count is then plotted as a bar chart for each feature, grouped by quality score, using the sns.barplot() method.

**Null Values:** isnull() is the process by which we can check whether there is any null value present in the data or not i.e. isnull() is part of the data cleaning stage. Here in the data, we found that there is no missing values.

**Correlation:** Correlation is a statistical measure that indicates the extent to which two or more variables are related to each other. In other words, it measures how much one variable changes when the other variable changes. The correlation coefficient ranges from -1 to 1, where -1 indicates a perfect negative correlation, 0 indicates no correlation, and 1 indicates a perfect positive correlation.

Insights:

Correlations from train and original datasets are different.

Pairwise correlation between features and Quality are similar between train and original dataset.

**Feature Engineering:**

The code defines a function FE which creates new features from the existing features in the dataset train\_df. The function FE takes the dataset as input, creates new features from the existing features and returns the dataset with the new features.

The new features created by the function FE are:

total\_acid: the sum of fixed\_acidity, volatile\_acidity, and citric\_acid.

acid/density: the ratio of total\_acid to density.

alcohol\_density: the product of alcohol and density.

sulphate/density: the ratio of total\_acid to density.

sulphates/acid: the ratio of sulphates to volatile\_acidity.

sulphates/chlorines: the ratio of sulphates to chlorides.

sulphates\*alcohol: the product of sulphates and alcohol.

The code then applies the FE function to train\_df and saves the resulting dataset as X. It then creates a list of the new features created by FE and saves it as new\_cols.

The code then plots box plots of the new features against the quality column in X. The box plots show the distribution of each new feature for each quality score in the dataset. The x-axis shows the quality score and the y-axis shows the values of the new feature.

**Basic Model:**

The code above performs cross-validation using stratified k-fold with 5 splits on the training dataset. The LightGBM algorithm is used as the machine learning model, with hyperparameters set to max\_depth=4, random\_state=42, and class\_weight='balanced'.

During each fold, feature engineering is performed on the training and validation sets using the FE() function, which creates new features based on combinations of existing ones. Then the model is trained on the transformed training data, and its performance is evaluated on the validation set using the quadratic weighted kappa score.

The mean quadratic weighted kappa score across all folds is printed as the final result 0.487035349323918.

The code defines a custom class called LGBMRegressorWithRounder that extends the LGBMRegressor class from the lightgbm package. This custom class adds functionality for rounding the predicted values to integer values that correspond to the levels of a categorical variable. This is useful when the target variable is a categorical variable with ordered levels, where the order between the levels is meaningful. The rounding is performed using an optimization method that finds the coefficients that minimize the quadratic weighted kappa loss between the rounded predicted values and the true values.

The LGBMRegressorWithRounder class overrides the fit() method of the LGBMRegressor class to add the rounding functionality. After fitting the model using the fit() method, the coefficients that minimize the quadratic weighted kappa loss are found using the minimize() method from the scipy.optimize package.

The class also adds two methods set\_params() and predict\_discrete(). The set\_params() method is used to reset the coefficients, and the predict\_discrete() method performs the rounding of the predicted values to integer values.

Overall, this code is useful for dealing with regression problems where the target variable is a categorical variable with ordered levels.

**Regressor Baseline:**

The code is performing cross-validation on a dataset using the LightGBMRegressor model with a custom wrapper, LGBMRegressorWithRounder, which deals with the issue of regression output needing to be rounded to integer values. The dataset is split into five folds using StratifiedKFold, with each fold used for validation once while the other four folds are used for training.

For each fold, the original dataset is concatenated with the training dataset to increase the size of the training set, and feature engineering is applied to both the training and validation datasets. The model is then fit on the training data using LGBMRegressorWithRounder, and the predicted values are rounded to integers using the custom predict\_discrete method. The performance of the model is evaluated using the quadratic weighted kappa score, with the results for each fold printed to the console.

At the end of the cross-validation loop, the mean quadratic kappa score is 0.5417366926533618.

**Confusion Matrix:** This code creates a heatmap using Seaborn's heatmap function to visualize the confusion matrix of the out-of-fold (OOF) predictions made by the trained LightGBM models. The confusion matrix is computed using Scikit-learn's confusion\_matrix function, which takes the true labels and predicted labels as input. The resulting confusion matrix is then plotted as a heatmap with annotations for the counts in each cell. The x and y tick labels are set to the unique values in the quality column of the training data, which are the possible labels for the wine quality. The plot is titled "Confusion Matrix

of OOF Predictions".