Project: Red Wine Quality: vectorized implementation with NumPy

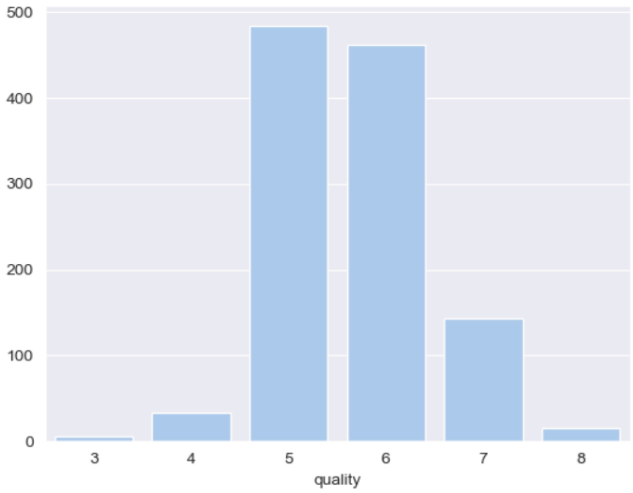
Sajjad Zangiabadi 99222046

Data Description:

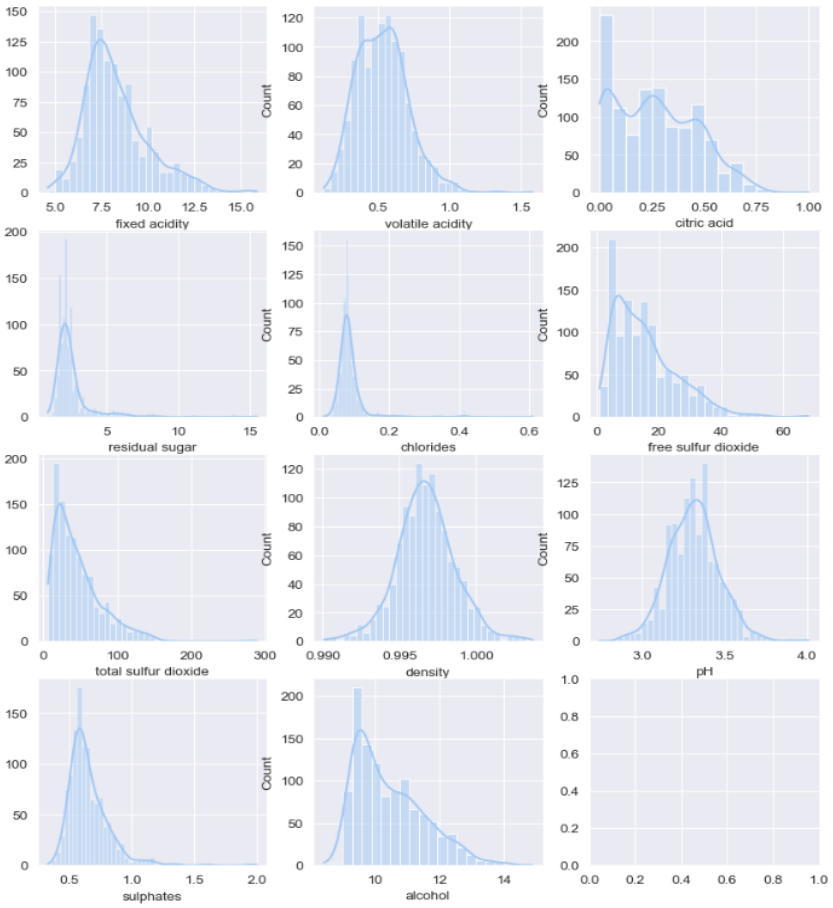
The dataset used for this project is the "Red Wine Quality" dataset obtained from Kaggle. It comprises physicochemical features of red wine samples and their corresponding quality ratings based on sensory evaluation. It contains 1143 samples and there aren't' not missing values. The dataset does not include information on grape types, wine brands, or selling prices, focusing on the characteristics of the wine itself. Our goal is to predict this quality score using the provided physicochemical properties, framing the problem as a multi-class classification task.

- Input Variables (Physicochemical Tests):
    - Fixed Acidity: The amount of non-volatile acids in the wine, contributing to its overall tartness.
    - Volatile Acidity: The amount of volatile acids in the wine, which can contribute to unpleasant flavors such as vinegar.
    - Citric Acid: The presence of citric acid, which can impart freshness and acidity to the wine.
    - Residual Sugar: The amount of sugar remaining in the wine after fermentation, influencing its sweetness.
    - Chlorides: The concentration of chloride ions in the wine, which can affect its taste and mouthfeel.
    - Free Sulfur Dioxide: The amount of sulfur dioxide presents in its free form, which acts as an antimicrobial agent and antioxidant.
    - Total Sulfur Dioxide: The total amount of sulfur dioxide, including both free and bound forms.
    - Density: The density of the wine, which is influenced by its alcohol and sugar content.
    - pH: The acidity or alkalinity of the wine, affecting its stability and taste perception.
    - Sulphates: The concentration of sulphates, which can contribute to wine preservation and aroma.
    - Alcohol: The percentage of alcohol in the wine, influencing its body and flavor profile.
- Target Variable (Quality):
    - Quality: A score between 0 and 10 assigned to each wine sample based on sensory evaluation. This serves as the target variable for our multi-class classification task, where higher scores indicate better perceived quality.

- Quality distribution:
    - the number of wines in the "quality" categories 5 and 6 are significantly higher (483 and 462 respectively) compared to categories like 3 (6) and 8 (16). This imbalance can negatively impact the performance of some machine learning models, where the model might prioritize predicting the majority class (5 and 6) more accurately. We can use techniques Like over sampling, under sampling to tackle this problem, but here we don't do this.



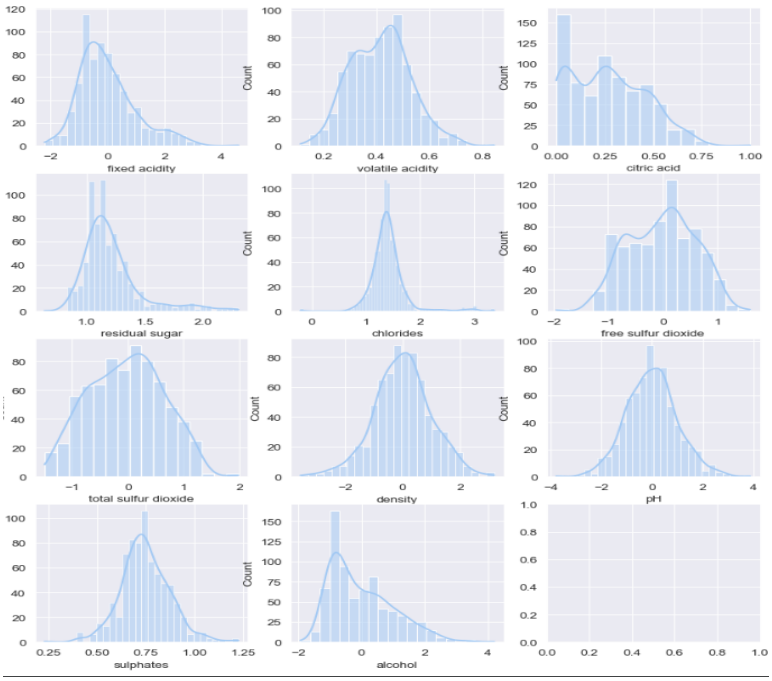
- Features distribution:



    - "Fixed acidity," "volatile acidity," "density," "pH," and "alcohol" exhibit characteristics of a normal distribution.
    - "Residual sugar," "chlorides," "free sulfur dioxide," and "total sulfur dioxide" display right-skewed distributions, indicating a concentration towards lower values with a few instances of higher values.
    - "Citric acid" demonstrates no distinctive distribution pattern.

- preprocessing:
    - Initial Train-Test Split (80% Train, 20% Test):
        - The data (data) is first divided into training and testing sets using a split ratio of 80% for training and 20% for testing. This ensures a representative portion of the data is reserved for final model evaluation (test_df).
        - The random_state parameter is set to 42 to ensure reproducibility, meaning if you run the code again with the same seed (42), you'll get the same split.
        - Importantly, the stratify parameter is set to data['quality']. This indicates stratified sampling, which ensures the proportion of classes in the training and testing sets reflects the original class distribution in your data. This is crucial in your case because you identified class imbalance in the "quality" variable. Stratification helps mitigate the biasing effects of imbalanced data on model training.
    - Creating a Development Set from the Training Data (Further Split):
        - The training set (train_df) is further split into two subsets: a new training set (train_df) and a development set (dev_df). Similar to the initial split, the ratio is 80% for the new training set and 20% for the development set.
        - The purpose of this additional split is to create a separate dataset (dev_df) that can be used for model validation and hyperparameter tuning while preserving the original train-test split proportions.
        - Stratification is again employed using train_df['quality'] to maintain the class distribution within the training and development sets.
    - Standardization:
        - Features such as "fixed acidity," "density," "pH," and "alcohol" are standardized using the StandardScaler from the sklearn.preprocessing module.
        - Standardization ensures that these features have a mean of 0 and a standard deviation of 1, making them comparable and facilitating model convergence during training.
        - Both the training, development, and testing sets undergo the same standardization procedure to maintain consistency across datasets.
    - Robust Scaling:
        - Features including "chlorides," "free sulfur dioxide," "total sulfur dioxide," and "sulphates" are scaled using the RobustScaler from the sklearn.preprocessing module.
        - Robust scaling is employed to handle outliers effectively by scaling the data based on quartiles rather than mean and standard deviation, making it more robust to extreme values.
        - Similar to standardization, both the training, development, and testing sets undergo robust scaling to ensure uniform treatment of features across datasets.
    - Log Transformation:
        - Features like "volatile acidity," "residual sugar," "free sulfur dioxide," "total sulfur dioxide," and "sulphates" undergo a log transformation using np.log1p.
        - Log transformation is applied to features with right-skewed distributions to reduce the skewness and make the data more normally distributed, which can improve the performance of certain machine learning algorithms.
        - Additionally, a constant value is added before taking the logarithm to handle negative values in the data, preventing potential issues with undefined logarithms.
        - The log transformation is applied to all three sets: training, development, and testing, to maintain consistency in preprocessing across datasets.

Distribution of features of applying scalers



- Split to X and y (features and target)
- Target Variable Encoding:
  - This step employs a technique known as one-hot encoding to transform the target variable ("quality") into a multi-class representation suitable for machine learning algorithms that handle multi-class classification tasks.

Modeling:

- MLP Classifier implementation:
  - Parameter Initialization:
    - The class initializes weights and biases, essential for model learning, using random values for a more diverse exploration of the solution space.
  - Forward Propagation: This involves a multi-step process:
    - Input data is fed into the input layer.
    - Each layer receives input, performs a linear transformation (weighted sum), and applies the specified activation function to introduce non-linearity.
    - This process is repeated until the final output layer produces predictions.
  - Backward Propagation:
    - During training, this process calculates gradients of loss with respect to the model's parameters (weights and biases) using backpropagation, guiding adjustments to improve predictions.
  - Parameter Updates:
    - Weights and biases are updated using an optimization algorithm (gradient descent in this case) to gradually minimize loss and enhance model accuracy.
  - Training Function:
    - The train function facilitates model training using input features and target labels, integrating forward and backward propagation, parameter updates, and loss/accuracy calculation for progress tracking.

- Prediction Function:
    - The predict function generates class predictions for new, unseen data using the trained model.
- Loss Function:
    - Categorical cross-entropy, a common loss function for multi-class classification, is implemented to quantify error during training, guiding model optimization.
- Accuracy Assessment:
    - The model's accuracy is measured by comparing predicted labels to true labels, providing a direct assessment of performance.

- Explanation of Model Training and Evaluation:
    - We conduct model training over multiple epochs using the training dataset (X_train and y_train) and subsequently evaluate its performance using the development dataset (X_dev and y_dev). Throughout the training process, we monitor key metrics such as training loss, training accuracy, development loss, and development accuracy.
    - By iterating over epochs, we aim to observe how the model's performance evolves over time and identify any trends or patterns. The recorded training loss and accuracy provide insights into how effectively the model is learning from the training data. Similarly, the development loss and accuracy offer a measure of the model's generalization performance on unseen data.
    - The gathered training and development metrics are saved in a history dictionary, allowing us to visualize the model's progress and assess its performance. We plot the training and development loss as well as the training and development accuracy over epochs. This visualization aids in understanding the model's learning dynamics and diagnosing potential issues such as overfitting or underfitting.
    - Following the training and evaluation phases, we further evaluate the finalized model's performance on the unseen test dataset (X_test). This final evaluation provides a measure of the model's accuracy on completely new data, gauging its effectiveness in real-world scenarios.