{Quantlyzer }

“get_info”: The function calculates summary statistics, checks for missing values, and provides an overview of the data structure. It also prints the number and percentage of missing values for each variable in the dataset, along with a list of column names.

Example: data <- get_info (data)

“drop_na”: This function handles missing values in a dataset by applying one of the following methods: omission, mean imputation, median imputation, or mode imputation.

Example: data <- get_info(data, method = c(“mean”, “omit”, “median”, “mode”))

“data_selection”: This function selects specific columns from a dataset (pos_X and pos_y), combines them, and processes missing values using the specified method, returning the cleaned data with adjusted column names (such as: “a b” = “a_b”).

Example: data <- data_selection (data, pos_X = c(1,2, 4:6), pos_y = 3, method = “omit”)

“data_into_five”: This function randomly shuffles and divides a dataset into five approximately equal parts, providing the row indices for each part, which can be useful for 5-fold cross-validation or splitting data for training and testing purposes.

Example: data <- data_into_five (data)

"dist_plot": The function creates histogram plots for all factor and character variables in the dataset. It iterates through each of these variables, generating a bar plot displaying the distribution of categories, using the ggplot2 package for visualization.

Example: dist_plot (data)

"violin_plot": The **violin_plot** function creates violin plots for all numerical variables in the dataset. It iterates through each of these variables, generating a violin plot displaying the distribution of values, along with mean and median represented as crossbars. The **ggplot2** package is used for visualization, and the plots are presented with a light theme and customizable text size.

Example: violin_plot (data)

"visual_summary1": The **visual_summary1** function provides a visual summary of the dataset by displaying violin plots for numerical variables and bar plots for categorical variables. It first separates the dataset into numerical and non-numerical variables. The function then calls **violin_plot** and **dist_plot** to generate the respective visualizations for these variables. Finally, it creates a correlation plot for the numerical variables using the **corrplot** package, displaying the correlations with pie charts in the upper triangle of the matrix, clustered using hierarchical clustering, and with rotated axis labels for readability.

Example: visual_summary1 (data)

“stat_ml”: The **stat_ml** function performs either regression or classification tasks on the dataset, depending on the given objective. It handles missing values using the specified **na_method** and optionally scales the data using the provided **scale_method**. The function calls either **all_reg_together** or **all_class_together** for regression and classification tasks, respectively, and their scaled counterparts if a scaling method is provided. The function returns the averaged results based on cross-validation, along with a warning indicating that the results have been averaged. It also calculates and prints the time taken to execute the function.

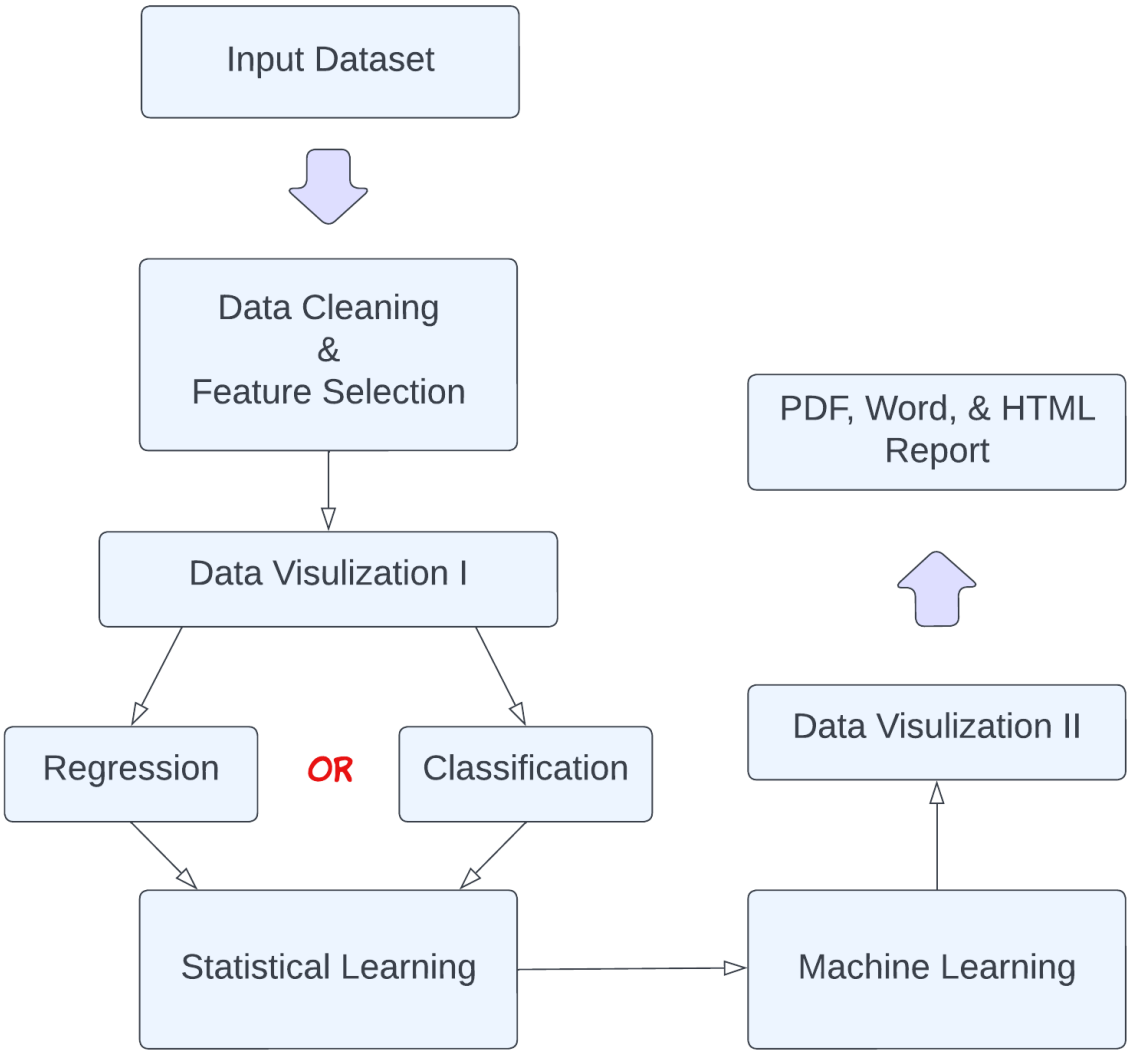
Example: result <- Quantlyzer::stat_ml(data = df, pos_x = c(1:7), pos_y = 8, objective = c(“class”, “reg”), na_method = “omit”, scale_method = c(“center”, “scale”, .etc)

stat_ml contains four sub-functions: “all_class_together”, “all_class_together_scaled”, “all_reg_together_scaled”, “all_reg_together”. Each of these functions contains many different algorithms STAT/ML. For more information, CHECK:

https://github.com/tianfengkai/Quantlyzer/blob/master/R/Quantlyzer.R

Machine learning (ML) and statistical algorithms have been significantly used in various applications, such as data classification, predictive regression, and feature selection. As the need for data-driven insights continues to grow, there is an increasing demand for exploratory and predictive data analysis to support business decision-making, academic research, and other applications. Identifying the best model that has optimal performance for a specific dataset usually consumes much time depending on the purpose of analysis. Although there are many packages that provide pre-built machine learning or statistical models, users still need time to load various suitable packages or functions, optimization of hyperparameters, validate the model, acknowledge the statistical relationship between each random or bivariate variable, and so on. This paper presents a package for R, "Quantlyzer" that contains various popular algorithms from machine learning and statistics. This tool aims to make automated data analysis more convenient for all different levels of users from no data analytics experience to domain experts to improve the efficiency of analyzing data. A workflow pipeline of exploratory analytics that contains various popular descriptive analysis techniques (e.g., Pearson Correlation Coefficient, a statistical summary of each variable, data visualization), statistical algorithms (e.g. Ridge regression, Ordinary Least Squares (OLS), Decision tree), machine learning models (e.g. Support Vector Machine (SVM), Random Forest, eXtreme Gradient Boosting (XGBoost), Gradient Boosting Machines (GBMs)), and Automated machine learning (AutoML) were ensembled in this package. The five-fold cross-validation is always used in every machine learning model to avoid overfitting or selection bias. A dataset with the index of soil organic carbon as the dependent variable and

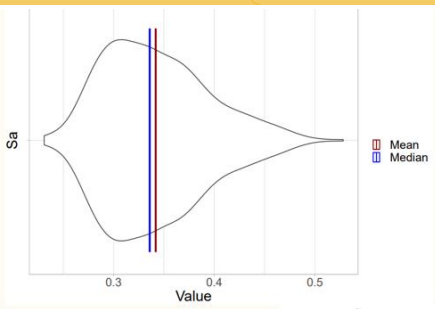
Near-Infrared Spectroscopy (NIRS) as independent variables was used to evaluate the performance of the predictive regressions for the package. Another dataset, which is based on estimating different levels of dicamba damage on the soybean plot with extracted image features as independent variables were used to testify classification performance by using the Quantlyzer. The result shows that a pipeline using ensembled various machine learning and statistical algorithms can not only generate the report within 2 hours but also provide broader information including data visualization, statistical analysis, and machine learning results with a few lines of code. This study demonstrates a possible method to develop an automated data analysis platform with various techniques to help both academics and industry to discover patterns in data. The goal of this project is designed to enhance the data analysis efficiency for users by minimizing the need for manual code input, ultimately reducing the effort and time consumption.

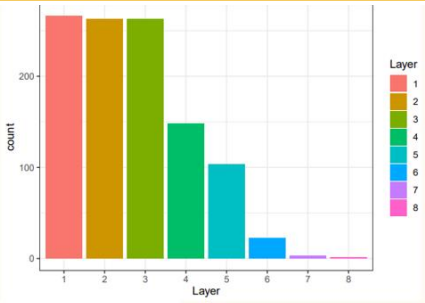

Input Dataset
Data Cleaning
&
Feature Selection
Data Visulization I
Regression
OR
Classification
Statistical Learning
Machine Learning
Data Visulization II
PDF, Word, & HTML
Report

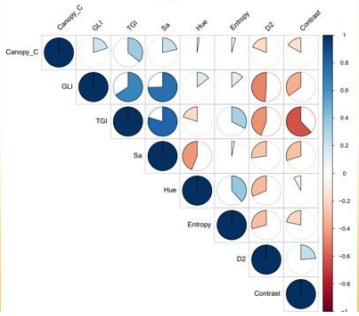
Methodology

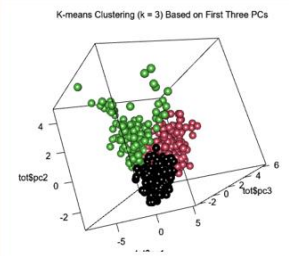
EDA

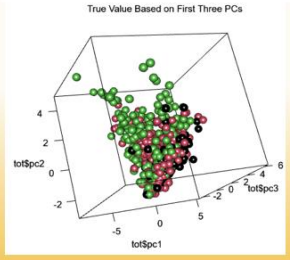
- **get_info**
  - stats_summary
  - na_value position
  - na_value %
  - type and pos of each var
- **visual_summary**
  - if factor | character → **dist_plot** → Histogram by var
  - if numeric
    - **violin_plot** → Violin by var
    - Pearson Cor Visual
- **data_selection**
  - pos of x
  - pos of y
  - **data_into_five** → Shuffle; evenly cut to 5,
  - **drop_na** → Options: omit, mean, mode, median
- **kme_pca**
  - PCA to 3 Dims ;
  - ↓ Kmeans;
  - ↓ Visual

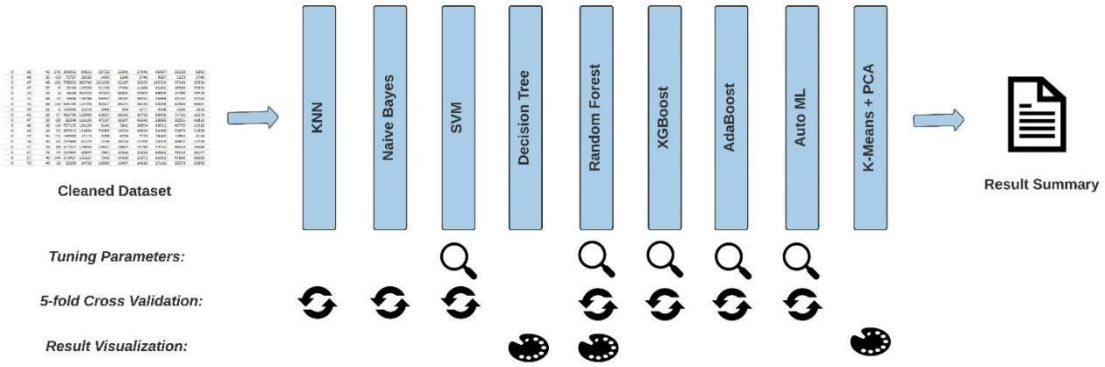


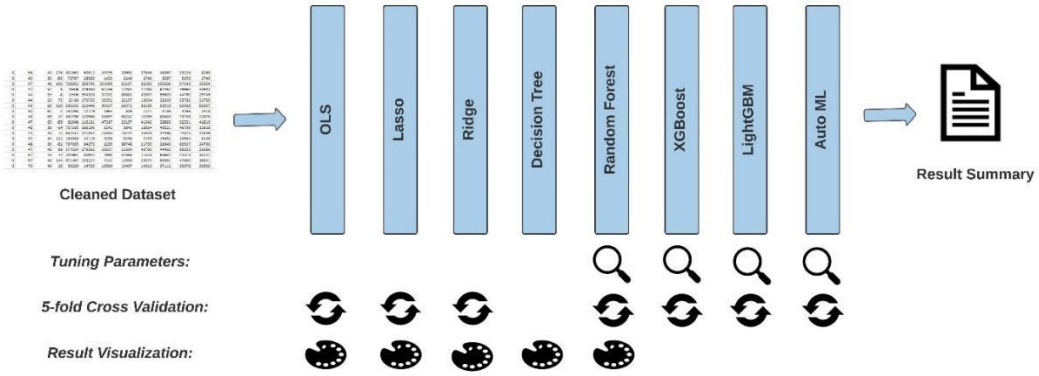


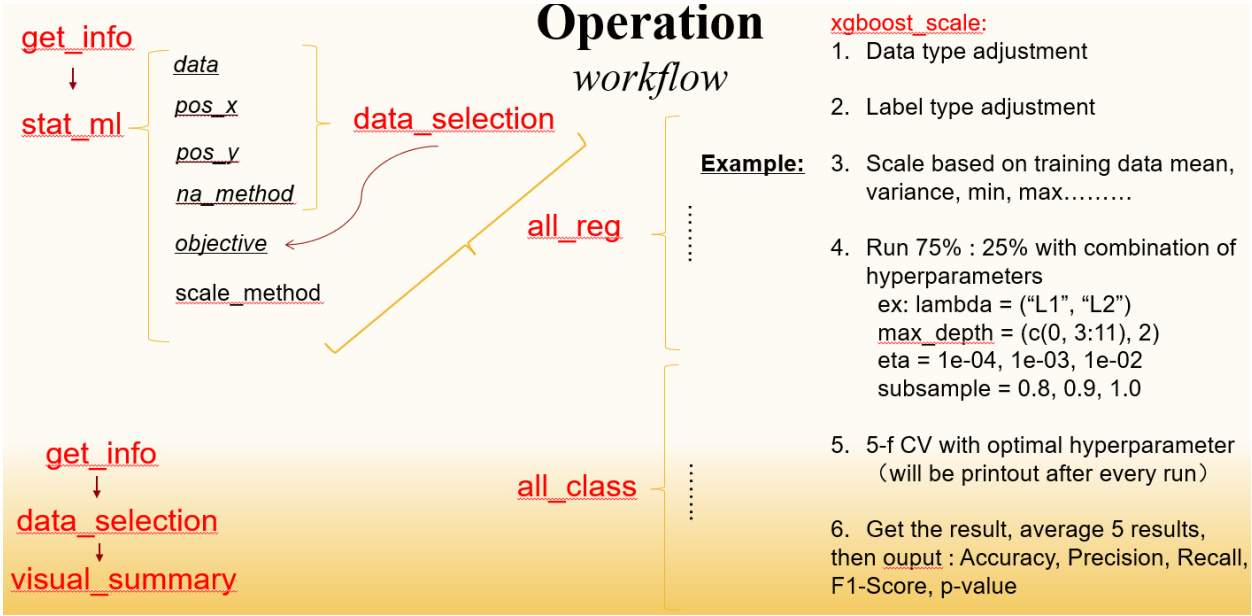







Cleaned Dataset
KNN
Naive Bayes
SVM
Decision Tree
Random Forest
XGBoost
AdaBoost
Auto ML
K-Means + PCA
Result Summary
Tuning Parameters:
5-fold Cross Validation:
Result Visualization:


Cleaned Dataset
OLS
Lasso
Ridge
Decision Tree
Random Forest
XGBoost
LightGBM
Auto ML
Result Summary
Tuning Parameters:
5-fold Cross Validation:
Result Visualization:


Operation
workflow
get_info
stat_ml
data
pos_x
pos_y
na_method
objective
scale_method
data_selection
all_reg
all_class
Example:
get_info
data_selection
visual_summary
xgboost_scale:
1. Data type adjustment
2. Label type adjustment
3. Scale based on training data mean, variance, min, max……
4. Run 75% : 25% with combination of hyperparameters
ex: lambda = (“L1”, “L2”)
max_depth = (c(0, 3:11), 2)
eta = 1e-04, 1e-03, 1e-02
subsample = 0.8, 0.9, 1.0
5. 5-f CV with optimal hyperparameter （will be printout after every run）
6. Get the result, average 5 results, then ouput : Accuracy, Precision, Recall, F1-Score, p-value

Chapter 3

Results & Discussion

3.2 Data Pre-processing

Data pre-processing is an essential step before running it through a machine learning or statistical models. The data pre-processing might significantly impact the accuracy and performance of machine learning models. Several data pre-processing functions in Quantlyzer package are designed to be user-friendly and effective in handling data of various types and formats.

The function "drop_na" is designed to handle the missing data for every variable. It offers four methods for dealing with missing data: 'omit', which removes the entire row containing missing data; 'mean', which replaces missing data with the mean value of the variable; 'median', which replaces missing data point with the median value of the variable, and 'mode', which

replaces missing data point with the mode value of the variable. The function applies the chosen method to each column in the input dataset, ensuring that all missing values are handled in a consistent manner.

The second function, "data_selection", is used for selecting specific variable from a dataset and handling missing data using the "drop_na" function (default is “omit”). Moreover, to ensure compatibility with certain algorithms that do not accept variable names with spaces, the function automatically replaces any spaces with underscores. This function also requires users to specify the index of the dependent variable and the independent variables. The output from the function is a modified dataframe where the independent variable is moved to the last column in order to ensure that further analysis can identify the independent variable in the dataset. Moreover, the "data_into_five" function is designed to create five subsets for the purpose of five-fold cross-validations which is an essential step in machine learning that helps to evaluate the performance of a model.

This function splits the data into five equal subsets based on the number of rows. The machine learning algorithm will run five times in total. During each iteration, 20% of the dataset is used for testing purpose, while the remaining 80% is used for training. Once all five iterations are completed, the final performance metrics for the algorithm, such as accuracy, F1-value, pvalue, mean square error, and others (depending on the classification / regression) will be averaged across the five iterations.

The package also includes three data scaling options for independent variables such as normalization, standardization, and min-max scaling. These scaling techniques are available in the function “stat_ml()”, which will be discussed further in section 3.3 and 3.4.

3.3 Performance Evaluation of the Classification Task

In the classification part of our study, I employed a range of statistical and machine learning algorithms, including Support Vector Machine, Naive Bayes, K-Nearest Neighbor, Decision Tree, Random Forest, XGBoost, AdaBoost, Auto-Keras classification, and K-Means with PCA.

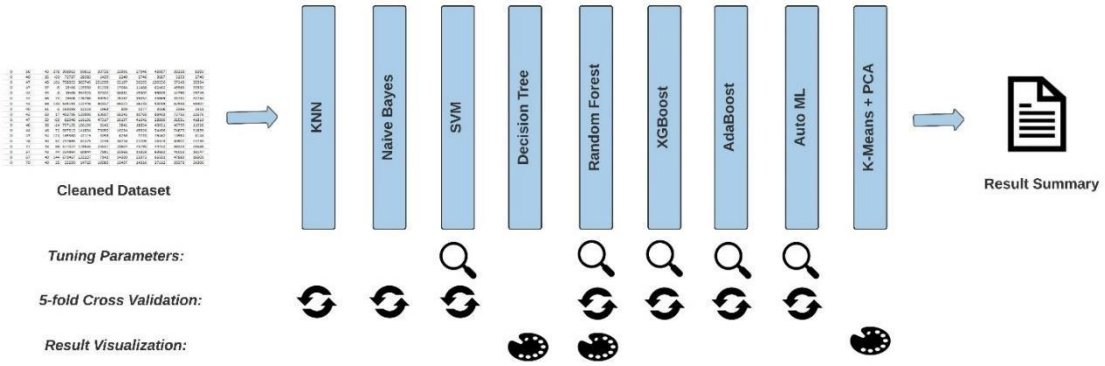


Figure 6. Pipeline of stat_ml(objective = “class”) function

Figure 6 shows that the Quantlyzer package has combined all the algorithms into a single, fully-automated function “stat_ml”. This integration streamlines the process of model optimization and evaluation, providing users with an efficient way while performing data analysis on classification tasks. By fitting the dataset into each model, the result can be displayed in the RStudio console consistently. Every algorithm has different tasks. For example, the random forest should determine its optimal parameters using a 75% training dataset and a 25% testing dataset firstly, and then compute the average confusion matrices based on the optimal parameters through five-fold cross-validation. Lastly, it will also visualize the importance of feature importance based on the mean decrease in the impurity of each variable.

In this section, I’m going to evaluate the stat_ml(objective = “class”) by using the first dataset which I mentioned in section 3.1.

```
R> result <- Quantlyzer::stat_ml(data = df, pos_x = c(1:7), pos_y = 8, objective = "class",
                                 na_method = "omit", scale_method = "center")
```

The above example designates columns 1 to 7 as independent variables while column 8 serves as the dependent variable (three classes). By setting the objective to "class", it defines the goal is to perform classification. The data pre-processing steps is also included; it specifies the function to remove rows with missing values (NA) (there is no NA value in the current dataset) and normalize the independent variables within the training dataset. The testing dataset is then scaled using the same normalization parameters derived from the training dataset. This ensures a consistent data transformation and prevents data leakage between the training and testing datasets, ultimately leading to a more reliable evaluation. The default setting of scale_method is "None" meaning no data scaling technique input. While running these models, the optimal parameters will be documented when parameter tuning is required. Visualizations, such as "feature importance" from the Random Forest and "decision-making flow" from the Decision Tree, and so on will also be shown as they might also provide valuable insights on the dataset.

```
## $`$ Accuracy, p-value, kappa`
##                Accuracy      p.value     kappa
## KNN           0.5251356 8.706410e-01 0.1572206
## Naive Bayes   0.7449664 1.211519e-03 0.4808446
## SVM           0.6029420 4.040366e-01 0.3275092
## Random Forest 0.7458398 6.504629e-04 0.4857467
## XGBoost       0.7025834 1.191005e-01 0.4606602
## Adaboost      0.7785235 5.470105e-05 0.5890654
## AutoML        0.6844718 2.120910e-02 0.4456942
##
## $precision
##                    [,1]      [,2]      [,3]
## KNN           0.2478071 0.6666340 0.3075594
## Naive Bayes   0.4736842 0.8061224 0.7187500
## SVM           0.2403863 0.6997828 0.5519947
## Random Forest 0.5975279 0.7987236 0.6767028
## XGBoost       0.5373999 0.7639170 0.6391471
## Adaboost      0.5795796 0.8362374 0.7413961
## AutoML        0.4996522 0.7444447 0.6513172
##
## $Recall
##                    [,1]      [,2]      [,3]
## KNN           0.2478071 0.6666340 0.3075594
## Naive Bayes   0.4736842 0.8061224 0.7187500
## SVM           0.2403863 0.6997828 0.5519947
## Random Forest 0.5975279 0.7987236 0.6767028
## XGBoost       0.5373999 0.7639170 0.6391471
## Adaboost      0.5795796 0.8362374 0.7413961
## AutoML        0.4996522 0.7444447 0.6513172
##
## $`F1 value`
##                    [,1]      [,2]      [,3]
## KNN           0.2336190 0.7139595 0.2781742
## Naive Bayes   0.4500000 0.8494624 0.6388889
## SVM           0.1948571 0.7424867 0.5444323
## Random Forest 0.4786667 0.8683791 0.6328244
## XGBoost       0.6269444 0.7292238 0.6663449
## Adaboost      0.5400000 0.8838710 0.6611111
## AutoML        0.4703333 0.7630174 0.6476347
```

Figure 7. Final Result of "*stat_ml(objective = "class")*"

Once the function has successfully executed, the result will be shown similar to the Figure 7. It consists of several sections, starting with "Accuracy & p-value", which demonstrates the average accuracy and p-value for each classification algorithm based on five-fold crossvalidation. Following that, the next three sections display the reuslt of F1- Socre, Precision, and

Recall, each containing three colums representing class 1, 2, and 3.

The goal of combining this essential information is to offer users a straightforward comparison of the performance of various machine learning algorithms. For example, while AdaBoost achieves the highest accuracy compared to others, Random Forest indicates considerabley higher precision value for the class 1. Therefore, it is relatively important to comprehensively compare the performance of various models from multiple angles in order to make further decisions.

R> *data %>% kme_pca()*

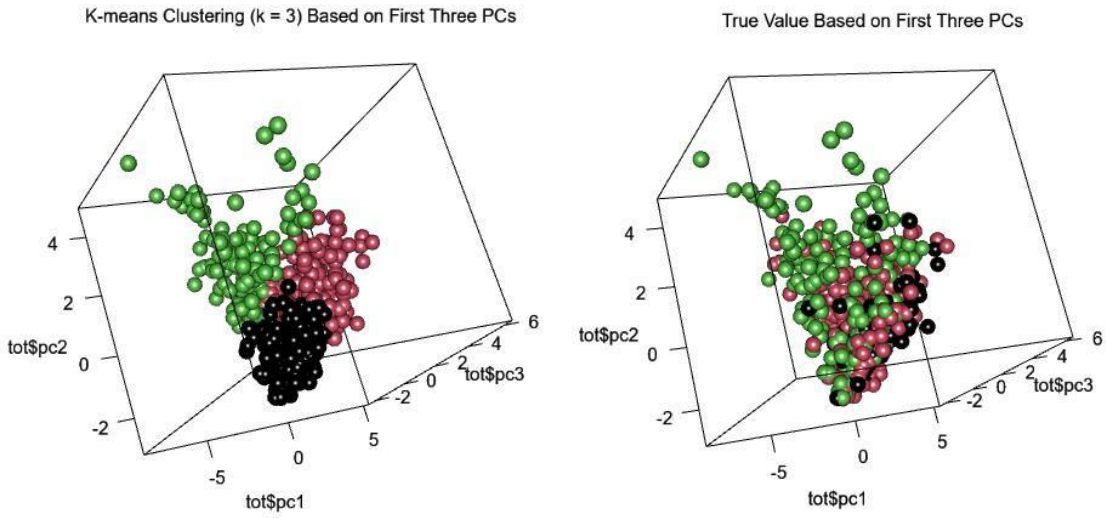


Figure 8. 3D plots from the function “kme_pca()” to visualize the first dataset

The Quantlyzer package includes a function called "kme_pca" (also contained in stat_ml) that applies k-means clustering of a specific dataset, with dimensionality reduction using Principal Component Analysis (PCA). The first three principal components are extracted to generate a 3D plot. Then, the K-means clustering algorithm is performed using Euclidean distance, with the "clusters" parameter determined by the number of unique values in the dependent variable. Eventually, the function produces two 3D plots: the first shows the k-means clustering results, while the second shows the original labels. The first three principal

components are used as the x, y, and z axes for both plots. Users can drag the image to view the plot from different angles in "plot" window in RSutido. The aim of the function is to help users identify natural clusters in high-dimensional datasets (the number of independent variables > 3) and have more understanding of the dataset.

Moreover, executing the "*stat_ml(objective = "class")*" function on the computer configuration described in Table 2, using the first dataset outlined in Section 3.1, takes 1 hr, 2 min, 13 sec to complete. The result showed a slightly improvement compared to the report in our previously published paper. However, I took a few weeks to finish the data processing pipeline during this Dicamba project. Based on that, I can conclude that the Quantlyzer package and its functions provide a more efficient and comprehensive approach to obtaining classification resulsts within a short period of time.

3.4 Performance Evaluation of the Regression Task

This section shares some similarities with the Section 3.3. When dealing with a dependent variable which is continuous, users should set the objective equal to "reg" in the function "stat_ml". Since the function "get_info" can output column indices, it is relatively helpful for users to specify the option pos_x while dealing with hundreds of independent variables.

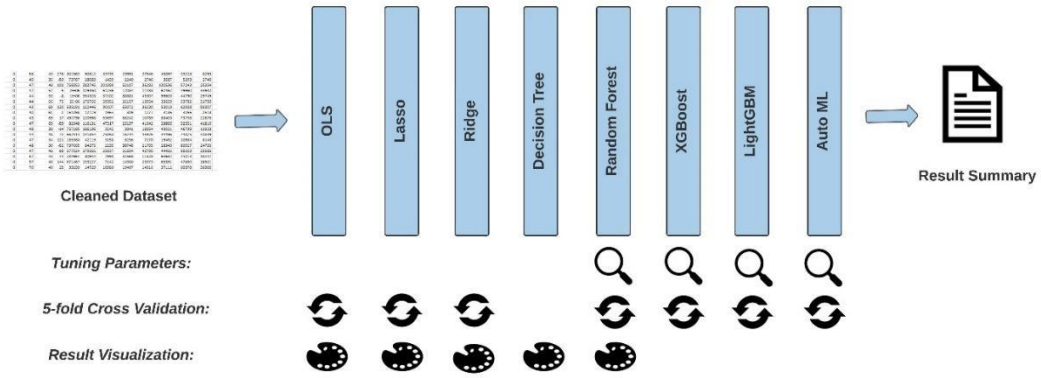


Figure 9. Pipeline of stat_ml(objective = "reg") function

Based on Figure 8, five algorithms offer insightful visualization. For OLS, there are four default diagnostic plots to help users evaluate model assumptions and performance, including residuals vs. fitted, normal Q-Q, scale-location, and residual vs. leverage. For LASSO and Ridge regression, the impact of the regularization parameter lambda on the model's performance can be visualized through the coefficient paths plot. In the case of decision tree (CART), the decisionmaking process will be illustrated in the tree diagram. The random forest can provide a feature importance visualization plot to convey the significance of each independent variable. The option to fine-tune parameters and employ five-fold cross-validation is also incorporated into several models based on their specific characteristics.

R> *Quantlyzer::stat_ml(data = data, pos_x = c(7, 10:14, 19:377), pos_y = 8, objective = "reg",*

na_method = "omit", scale_method = "None")

In this example, the user specifies columns 7, 10, 14, and 19 to 377 as the independent variables, while column 8 is set as the continuous dependent variable. The objective is set to

"reg" for regression. The function can also remove rows containing missing values (NA) and perform no scaling on the dataset (scale_method = "None"). The "stat_ml" function will then run a series of regression algorithms that was shown in figure 8. Similar to the section 3.3, the function will also perform model optimization, parameter tunining (if necessary), and evaluation.

```
##          r_square R_square  rmse   mae
## lm          0.384   -2.233 0.741 0.583
## lasso       0.763    0.548 0.306 0.228
## ridge       0.746    0.525 0.309 0.236
## rf          0.666    0.381 0.359 0.280
## xgboost     0.751    0.245 0.314 0.235
## automl      0.758    0.541 0.310 0.226
## lightgbm    0.779    0.591 0.287 0.211
```

Figure 10. Pipeline of stat_ml(objective = "reg") function

Upon successful execution of the function, the result will be displayed, similar to Figure 10. In contrast to classification, regression part has only one section containing four columns, which include "r^2", "R^2", "RMSE", and "MAE" based on five-fold cross validation. These results effectively represent the performance of each model.

Executing the "*stat_ml(objective = "reg")*" function on the computer configuration described in Table 2, using the second dataset mentioned in Section 3.1, takes 1 hr, 59 min, and 29 sec to get the final summary. The result from Figure 10 displays worse results compared to the previous report. Nevertheless, this doesn't imply the weakness of the package. In the previous work, I employed 1D-CNN and 2D-CNN manually to build the deep learning model, and the architecture consisted of multiple convolutional layers, pooling layers, batch normalization layers, and more. While Auto-Keras also utilizes Neural Architecture Search (NAS) to fit the

dataset and fine-tune hyperparameters, determining the optimal model for making prediction typically demands a large set of data and computational resources than manual keras for optimal performance according to the [34]. Thus, manually building the deep learning can provide greater control over the model design, which could be a limitation in this package. However, all of the algorithms in Quantlyzer do not necessitate users to manually create the architecuture. A trade-off exists between speed and performance when comparing using Quantlyzer and deveploing a DCNN. While a well-designed deep learning model might outperform the performance of Quantlyzer, an ensemble of various algorithms package, it generally demands a significant amount of time for testing and constructing the model.