In this exercise, you will design and implement an obesity detection system using individuals' dietary habits and physical condition data, leveraging the XGBoost algorithm. The exercise will start with data analysis, followed by data preprocessing and model training. Bayesian optimization will be used to tune the hyperparameters for optimal performance.
The primary goal of this exercise is to optimize the model's parameters to achieve the highest possible accuracy in predicting the target variable based on the training data. Bayesian optimization will be utilized to adjust key hyperparameters, reducing overfitting and improving model stability. The final evaluation of the model will be conducted using test data.
It is recommended to review the dataset description provided at (link) before starting. The dataset, this explanatory document, and a notebook (.ipynb format) that needs to be completed (#TODO) are included in the exercise files. This notebook, along with the exercise report, will be submitted as the final practice deliverable. Detailed instructions for each required section are included in the notebook.

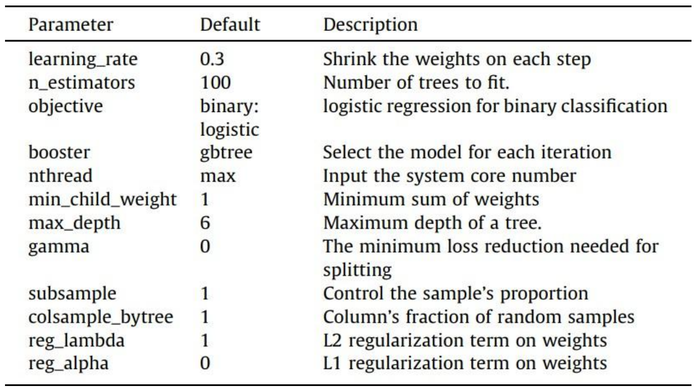
Steps:

1) Data Preprocessing and Data Analysis:
In this section, you need to explore the dataset for a better understanding of the data.

- provide brief explanations for each figure.
- Create an additional statistical visualization at the end and explain its insights.
- Use one-hot encoding (get_dummies) to transform categorical features.
- Standardize the dataset to ensure consistency before training.

2) Model Implementation and Hyperparameter Optimization:
Implement the model using relevant library functions. you do not need to implement XGBoost from scratch. Hyperparameter tuning is crucial for enhanced model performance.

- Use Bayesian optimization to tune the model. The xgb_evaluate function in the notebook includes nine hyperparameters to be optimized (Table 1)



Parameter	Default	Description
learning_rate	0.3	Shrink the weights on each step
n_estimators	100	Number of trees to fit.
objective	binary: logistic	logistic regression for binary classification
booster	gbtree	Select the model for each iteration
nthread	max	Input the system core number
min_child_weight	1	Minimum sum of weights
max_depth	6	Maximum depth of a tree.
gamma	0	The minimum loss reduction needed for splitting
subsample	1	Control the sample's proportion
colsample_bytree	1	Column's fraction of random samples
reg_lambda	1	L2 regularization term on weights
reg_alpha	0	L1 regularization term on weights

- Each optimization step should be evaluated using the AUC metric with K-fold cross-validation.

3) Model Evaluation and Performance Comparison:

During and after the implementation, ensure the following outputs are recorded:

- Display the parameter values and corresponding AUC at each step, as shown in the notebook.
- List the best hyperparameters and the highest AUC at the end of optimization. Calculate and report:
- Accuracy
- Recall
- Precision
- Specificity
- Confusion matrix

➢ Repeat the implementation and evaluation (Step 2-3) with the Decision Tree model. Extract and report the same output metrics and compare the performance between the two models. The dt_evaluate function in the notebook will be used for optimizing the Decision Tree model's hyperparameters.

Notes:

- The code utilizes libraries such as pandas, numpy, XGBoost, BayesianOptimization and...
- Data preprocessing involves:
    - Encoding categorical features using get_dummies() and LabelEncoder.
    - Scaling data with StandardScaler to standardize features.
- The model is evaluated using metrics like AUC, accuracy, recall, precision, and a confusion matrix, visualized with a heatmap for clarity.
- Bayesian optimization searches for the best set of hyperparameters in XGBoost, including max_depth, gamma, subsample, and more, to enhance the model's predictive performance.