ESTIMATION OF OBESITY LEVELS BASED ON EATING HABITS AND PHYSICAL CONDITION

Project Presentation

DATASET

Source: Retrieved dataset from (machine learning repo) https://archive.ics.uci.edu/

Data: Collected via a comprehensive individual survey

Goal: Develop predictive models for obesity levels

DATA ATTRIBUTES OVERVIEW

1. Numeric Data:

- Age [14, 61]
- Height [1.45, 1.98]
- Weight [39.0, 173.0]
- NCP (Number of Main Meals) [1..4]
- CH2O (Daily Water Consumption) [1, 3]
- FAF (Frequency of Physical Activity, how many days a week) [0..3]
- TUE (Time Using Technology Devices) [0..3]

DATA ATTRIBUTES OVERVIEW

2. Categorical Data:

- Gender
  - {Male, Female}
- FCVC (Frequency of Vegetable Consumption)
  - {1 (low), 2 (medium), 3 (high)}
- CAEC (Consumption of Food Between Meals)
  - {Sometimes, Frequently, Always, No}
- CALC (Alcohol Consumption)
  - {Sometimes, Frequently, Always, No}
- MTRANS (Mode of Transportation)
  - {Public_Transportation, Automobile, Walking, Motorbike, Bike}

DATA ATTRIBUTES OVERVIEW

2. Categorical Data:

- NObeyesdad (Obesity Level)
  - {Normal_Weight, Overweight_Level_1, Overweight_Level_2, Obesity_Type_1, Obesity_Type_2. Obesity_Type_3. Insufficient_Weight}

DATA ATTRIBUTES OVERVIEW

3. Binary Columns: {Yes, No}

- Family History with Overweight
- SMOKE (Smoking Habits)
- SCC (Monitoring of Calorie Intake)
- FAVC (Frequent Consumption of High Caloric Food)

DATA PREPROCESSING

DATA PREPROCESSING CHECKLIST

- Missing Values
- Imbalanced Classes
- Encoding
- Outliers
- Generalization

DATA PREPROCESSING CHECKLIST

- Missing Values

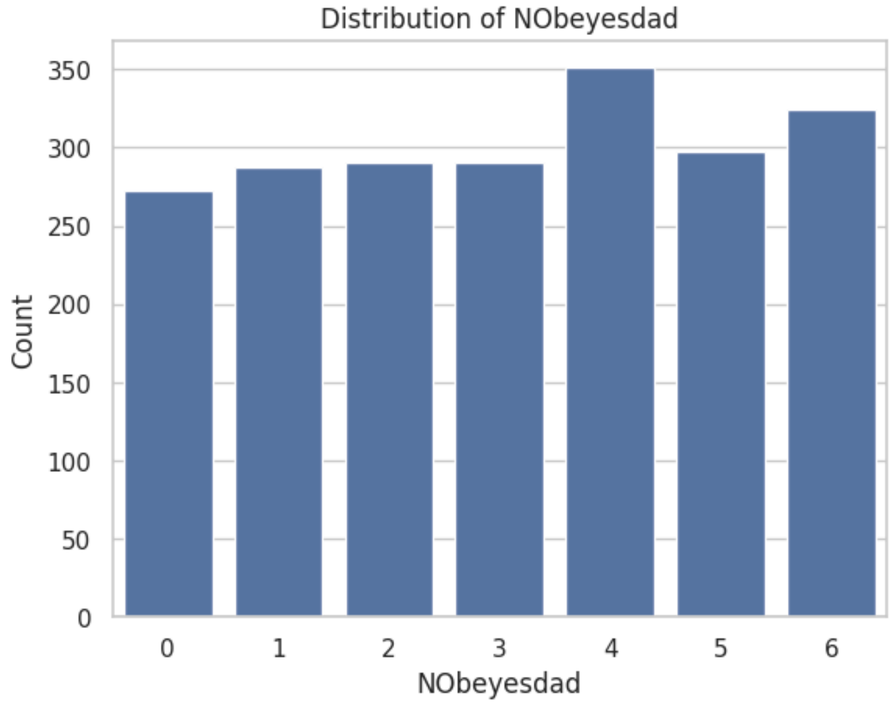
```
#checking for missing values
df.isnull().sum()
```

```
Gender                            0
Age                               0
Height                            0
Weight                            0
family_history_with_overweight    0
FAVC                              0
FCVC                              0
NCP                               0
CAEC                              0
SMOKE                             0
CH2O                              0
SCC                               0
FAF                               0
TUE                               0
CALC                              0
MTRANS                            0
NObeyesdad                        0
dtype: int64
```

No missing values were found in the dataset.

DATA PREPROCESSING CHECKLIST

- Imbalanced Classes



The distribution of the target variable 'NObeyesdad' indicates a balanced class, with similar counts across different obesity levels.

DATA PREPROCESSING CHECKLIST

- Encoding

```
df = pd.get_dummies(df, columns=['Gender'])
```

```
#encoding the column  we use label encoding
from sklearn.preprocessing import LabelEncoder

# Initialize the label encoder
label_encoder = LabelEncoder()

# Fit and transform the 'family_history_with_overweight' column
df['family_history_with_overweight'] = label_encoder.fit_transform(df['family_history_with_overweight'])

# This will convert 'yes' to 1 and 'no' to 0
```

```
obesity_mapping = {
    'Insufficient_Weight': 0,
    'Normal_Weight': 1,
    'Overweight_Level_I': 2,
    'Overweight_Level_II': 3,
    'Obesity_Type_I': 4,
    'Obesity_Type_II': 5,
    'Obesity_Type_III': 6
}

# Applying the mapping to the 'NObeyesdad' column
df['NObeyesdad'] = df['NObeyesdad'].map(obesity_mapping)
```

1.One-Hot Encoding: Creating columns for each categorical value.
2. Label Encoding: For cases of two values, assign 0 or 1.
3.Mapping encoding

DATA PREPROCESSING CHECKLIST

- Outliers

```
#Height column validation
print(df['Height'].min())
print(df['Height'].max())
```

```
1.45
1.98
```

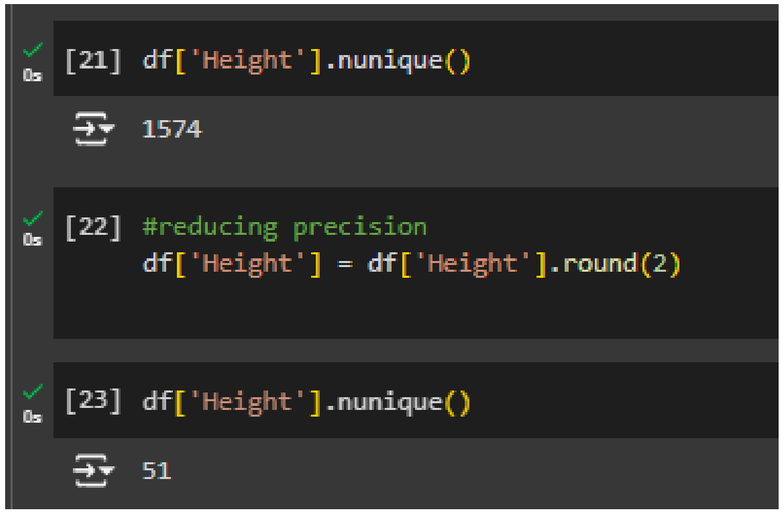
```
#Age Column Validation
print('min age ' ,df['Age'].min())
print('max age ', df['Age'].max())

print('Number of participants for each a
df['Age'].value_counts()
```

```
min age  14.0
max age  61.0
```

The minimum and maximum values for each column are significant. There is no sign of outliers.

DATA PREPROCESSING CHECKLIST

- Generalization



Most columns are too specific for unnecessary topics, like the number of meals a day or liters of water you drink a day. We need to generalize them using the round function.

REGRESSION MODELS

LINEAR REGRESSION

- Target Variable : Weight
- Model Evaluation

```
[11] # Make predictions
     y_pred_linear = linear_reg.predict(X_test)


     # Mean Squared Error
     mse_linear = mean_squared_error(y_test, y_pred_linear)
     print("Mean Squared Error (Linear Regression):", mse_linear)

     # Root Mean Squared Error
     rmse_linear = mean_squared_error(y_test, y_pred_linear, squared=False)
     print("Root Mean Squared Error (Linear Regression):", rmse_linear)

     # Mean Absolute Error
     mae_linear = mean_absolute_error(y_test, y_pred_linear)
     print("Mean Absolute Error (Linear Regression):", mae_linear)

     # R-squared
     r2_linear = r2_score(y_test, y_pred_linear)
     print("R-squared (Linear Regression):", r2_linear)
```

```
Mean Squared Error (Linear Regression): 28.488775191352858
Root Mean Squared Error (Linear Regression): 5.3374877228292394
Mean Absolute Error (Linear Regression): 3.8531181038417337
R-squared (Linear Regression): 0.959568179452467
```

RANDOM FOREST

- Target Variable : Weight
- Model Evaluation

```
[14] # Make predictions
     y_pred_rf = rf_regressor.predict(X_test)



     # Mean Squared Error
     mse_rf = mean_squared_error(y_test, y_pred_rf)
     print("Mean Squared Error (Random Forest Regression):", mse_rf)

     # Root Mean Squared Error
     rmse_rf = mean_squared_error(y_test, y_pred_rf, squared=False)
     print("Root Mean Squared Error (Random Forest Regression):", rmse_rf)

     # Mean Absolute Error
     mae_rf = mean_absolute_error(y_test, y_pred_rf)
     print("Mean Absolute Error (Random Forest Regression):", mae_rf)

     # R-squared
     r2_rf = r2_score(y_test, y_pred_rf)
     print("R-squared (Random Forest Regression):", r2_rf)
```

```
Mean Squared Error (Random Forest Regression): 8.7444993784335
Root Mean Squared Error (Random Forest Regression): 2.9571099706357726
Mean Absolute Error (Random Forest Regression): 1.819277103043363
R-squared (Random Forest Regression): 0.9875896374178224
```

CONCLUSION

Random Forest Regression outperforms Linear Regression, making it better for predicting weight based on the given features.

WHY RANDOM FOREST PERFORMS BETTER?

- **Handles Non-Linearity:** Captures complex relationships between features like age, height, diet, and activity.
- **Robust to Outliers:** Less sensitive to extreme values.
- **Categorical Data:** Effectively manages categorical variables without extensive preprocessing.
- **Reduces Overfitting:** Averages multiple decision trees for more accurate predictions.

CLASSIFICATION MODELS

SVM & RANDOM FOREST

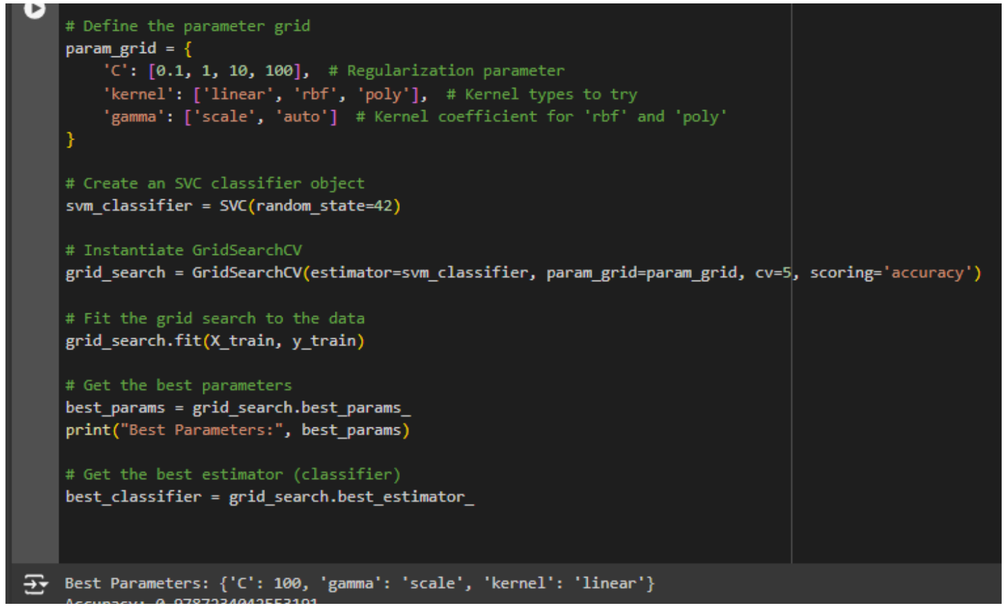
SVM

- Target Variable : NObeyesdad (7 levels of obesity)
- Model Evaluation (before tunning)

```
Accuracy: 0.8841607565011821

Classification Report:
              precision    recall  f1-score   support

           0       0.84      1.00      0.91        56
           1       0.91      0.63      0.74        62
           2       0.76      0.86      0.81        56
           3       0.83      0.78      0.80        50
           4       0.93      0.91      0.92        78
           5       0.91      1.00      0.95        58
           6       1.00      1.00      1.00        63

    accuracy                           0.88       423
   macro avg       0.88      0.88      0.88       423
weighted avg       0.89      0.88      0.88       423


Confusion Matrix:
[[56  0  0  0  0  0  0]
 [11 39  9  3  0  0  0]
 [ 0  4 48  4  0  0  0]
 [ 0  0  6 39  5  0  0]
 [ 0  0  0  1 71  6  0]
 [ 0  0  0  0  0 58  0]
 [ 0  0  0  0  0  0 63]]
```

SVM

- Target Variable : NObeyesdad (7 levels of obesity)
- Grid Search for hyperparameter tunning

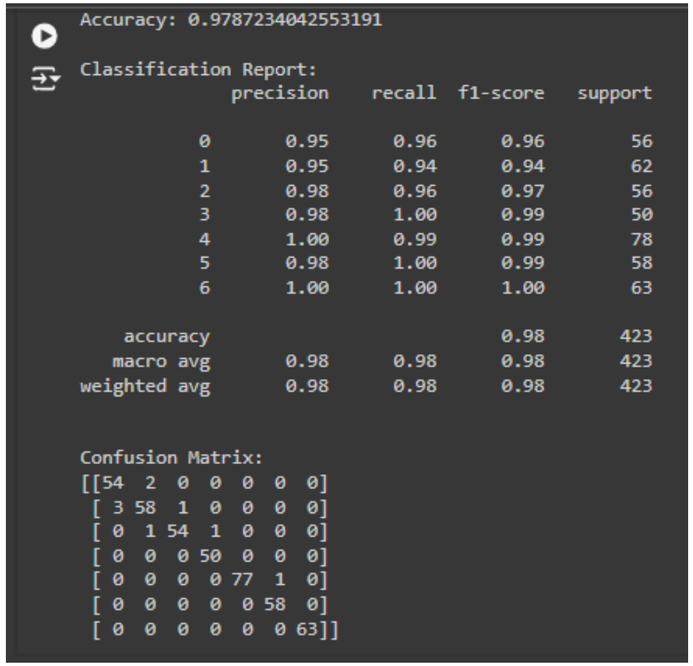


```
# Define the parameter grid
param_grid = {
    'C': [0.1, 1, 10, 100],  # Regularization parameter
    'kernel': ['linear', 'rbf', 'poly'],  # Kernel types to try
    'gamma': ['scale', 'auto']  # Kernel coefficient for 'rbf' and 'poly'
}

# Create an SVC classifier object
svm_classifier = SVC(random_state=42)

# Instantiate GridSearchCV
grid_search = GridSearchCV(estimator=svm_classifier, param_grid=param_grid, cv=5, scoring='accuracy')

# Fit the grid search to the data
grid_search.fit(X_train, y_train)

# Get the best parameters
best_params = grid_search.best_params_
print("Best Parameters:", best_params)

# Get the best estimator (classifier)
best_classifier = grid_search.best_estimator_
```

```
Best Parameters: {'C': 100, 'gamma': 'scale', 'kernel': 'linear'}
```

SVM

- Target Variable : NObeyesdad (7 levels of obesity)
- Model Evaluation (After tunning)



```
Accuracy: 0.9787234042553191

Classification Report:
              precision    recall  f1-score   support

           0       0.95      0.96      0.96        56
           1       0.95      0.94      0.94        62
           2       0.98      0.96      0.97        56
           3       0.98      1.00      0.99        50
           4       1.00      0.99      0.99        78
           5       0.98      1.00      0.99        58
           6       1.00      1.00      1.00        63

    accuracy                           0.98       423
   macro avg       0.98      0.98      0.98       423
weighted avg       0.98      0.98      0.98       423


Confusion Matrix:
[[54  2  0  0  0  0  0]
 [ 3 58  1  0  0  0  0]
 [ 0  1 54  1  0  0  0]
 [ 0  0  0 50  0  0  0]
 [ 0  0  0  0 77  1  0]
 [ 0  0  0  0  0 58  0]
 [ 0  0  0  0  0  0 63]]
```

RANDOM FOREST

- Target Variable : NObeyesdad (7 levels of obesity)
- Model Evaluation

```
[20]
Accuracy (Random Forest): 0.950354609929078

Classification Report (Random Forest):
              precision    recall  f1-score   support

           0       0.98      0.96      0.97        56
           1       0.87      0.89      0.88        62
           2       0.86      0.86      0.86        56
           3       0.94      0.96      0.95        50
           4       0.99      0.99      0.99        78
           5       1.00      0.98      0.99        58
           6       1.00      1.00      1.00        63

    accuracy                           0.95       423
   macro avg       0.95      0.95      0.95       423
weighted avg       0.95      0.95      0.95       423


Confusion Matrix (Random Forest):
[[54  2  0  0  0  0  0]
 [ 1 55  6  0  0  0  0]
 [ 0  6 48  2  0  0  0]
 [ 0  0  2 48  0  0  0]
 [ 0  0  0  1 77  0  0]
 [ 0  0  0  0  1 57  0]
 [ 0  0  0  0  0  0 63]]
```

CONCLUSION

- SVM achieves ~97% accuracy; Random Forest achieves ~95%.
- SVM has slower training times; Random Forest scales better with large datasets.
- SVM models are more interpretable; Random Forest models are harder to interpret.
- SVM can overfit, especially with nonlinear kernels; Random Forest is less prone to overfitting.
- SVM offers higher accuracy and interpretability.
- Random Forest excels in scalability and robustness against overfitting.

LOGISTIC REGRESSION & KNN

LOGISTIC REGRESSION

- **Target Variable : NObeyesdad (7 levels of obesity)**
- **Scaling the data**

We adjust the scale of data to ensure all numerical features have the same range, preventing any single feature from dominating and improving the performance of machine learning algorithms, like logistic regression.

```
[ ]  from sklearn.preprocessing import StandardScaler

     # Instantiate the StandardScaler
     scaler = StandardScaler()

     # Fit and transform the training data
     X_train_scaled = scaler.fit_transform(X_train)

     # Transform the test data using the same scaler
     X_test_scaled = scaler.transform(X_test)
```

LOGISTIC REGRESSION

- Target Variable : NObeyesdad (7 levels of obesity)
- Model Evaluation

```
Accuracy (Logistic Regression): 0.8676122931442081

Classification Report (Logistic Regression):
              precision    recall  f1-score   support

           0       0.88      1.00      0.93        56
           1       0.86      0.61      0.72        62
           2       0.70      0.77      0.74        56
           3       0.78      0.86      0.82        50
           4       0.94      0.87      0.91        78
           5       0.88      0.97      0.92        58
           6       1.00      1.00      1.00        63

    accuracy                           0.87       423
   macro avg       0.86      0.87      0.86       423
weighted avg       0.87      0.87      0.86       423


Confusion Matrix (Logistic Regression):
[[56  0  0  0  0  0  0]
 [ 8 38 13  3  0  0  0]
 [ 0  6 43  7  0  0  0]
 [ 0  0  5 43  2  0  0]
 [ 0  0  0  2 68  8  0]
 [ 0  0  0  0  2 56  0]
 [ 0  0  0  0  0  0 63]]
```

KNN

- **Target Variable : NObeyesdad (7 levels of obesity)**
- **Model Evaluation (before tunning)**

First Parameters

- knn__algorithm : “auto”
- k : 5
- k_weights : “uniform”

```
Accuracy: 0.7966903073286052
Classification Report:
              precision    recall  f1-score   support

           0       0.77      0.89      0.83        56
           1       0.67      0.52      0.58        62
           2       0.67      0.66      0.67        56
           3       0.73      0.72      0.73        50
           4       0.86      0.81      0.83        78
           5       0.82      0.97      0.89        58
           6       0.97      1.00      0.98        63

    accuracy                           0.80       423
   macro avg       0.79      0.79      0.79       423
weighted avg       0.79      0.80      0.79       423

Confusion Matrix:
[[50  2  4  0  0  0  0]
 [ 9 32  6  7  5  3  0]
 [ 5  8 37  2  2  1  1]
 [ 1  4  3 36  2  3  1]
 [ 0  2  5  3 63  5  0]
 [ 0  0  0  1  1 56  0]
 [ 0  0  0  0  0  0 63]]
```

KNN

- **Target Variable : NObeyesdad (7 levels of obesity)**
- **Grid Search**

Best Parameters

- knn__algorithm : “auto”
- k : 3
- k_weights : “distance”

```python
# Step 1: Define the parameter grid
param_grid = {
    'knn__n_neighbors': [3, 5, 7],  # Number of neighbors
    'knn__weights': ['uniform', 'distance'],  # Weight function
    'knn__algorithm': ['auto', 'ball_tree', 'kd_tree', 'brute'],  # Algorithm used to compute the nearest neighbors
}

# Step 2: Instantiate the KNN classifier within a pipeline
knn_pipeline = Pipeline([
    ('scaler', StandardScaler()),
    ('knn', KNeighborsClassifier())
])

# Step 3: Instantiate the GridSearchCV
grid_search = GridSearchCV(estimator=knn_pipeline, param_grid=param_grid, cv=5, scoring='accuracy')

# Step 4: Fit the GridSearchCV to the training data
grid_search.fit(X_train_scaled, y_train)

# Step 5: Get the best parameters and best estimator
best_params = grid_search.best_params_
best_estimator = grid_search.best_estimator_

print("Best Parameters:", best_params)
```

```
Best Parameters: {'knn__algorithm': 'auto', 'knn__n_neighbors': 3, 'knn__weights': 'distance'}
```

KNN

- Target Variable : NObeyesdad (7 levels of obesity)
- Model Evaluation (after tunning)

```
Accuracy: 0.8014184397163121
Classification Report:
              precision    recall  f1-score   support

           0       0.82      0.91      0.86        56
           1       0.67      0.48      0.56        62
           2       0.75      0.70      0.72        56
           3       0.65      0.74      0.69        50
           4       0.82      0.81      0.81        78
           5       0.85      0.97      0.90        58
           6       0.98      1.00      0.99        63

    accuracy                           0.80       423
   macro avg       0.79      0.80      0.79       423
weighted avg       0.80      0.80      0.80       423

Confusion Matrix:
[[51  2  3  0  0  0  0]
 [ 8 30  6 10  5  3  0]
 [ 2  8 39  3  4  0  0]
 [ 0  3  2 37  4  3  1]
 [ 1  2  2  6 63  4  0]
 [ 0  0  0  1  1 56  0]
 [ 0  0  0  0  0  0 63]]
```

CONCLUSION

Based on these evaluation metrics, Logistic Regression outperforms KNN in terms of accuracy, precision, recall, and F1-score. The confusion matrix for Logistic Regression also indicates better performance with fewer misclassifications.

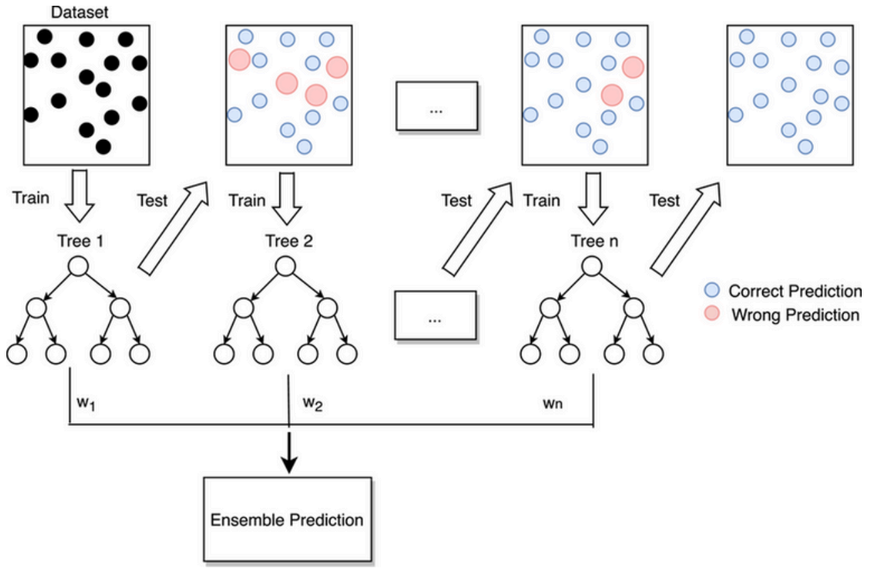
WHY DID LOGISTIC REGRESSION PERFORM BETTER?

- **Linearity:** Logistic Regression handles linear relationships well.
- **Categorical Data:** Effectively manages categorical variables after encoding.
- **Feature Scaling:** Less sensitive to feature scaling compared to KNN.
- **Noise Sensitivity:** More robust to noise than KNN.
- **Efficiency:** Handles larger datasets and higher dimensions more efficiently.
- **Overfitting:** Less prone to overfitting compared to KNN.

GRADIENT BOOSTING CLASSIFIER

HOW IT WORKS ?

- **Initialize the Model:** Set hyperparameters like the number of trees and learning rate.
- **Fit the First Base Model:** Train the first decision tree on the training data.
- **Calculate Residuals:** Compute the difference between predicted and actual values.
- **Fit New Base Model to Residuals:** Train a new model to predict the residuals.
- **Update Predictions:** Combine predictions from all models iteratively.
- **Repeat Steps 3-5:** Iteratively improve predictions for a specified number of iterations.
- **Final Prediction:** Aggregate predictions from all base models for the final prediction.
- **Evaluate the Model:** Assess model performance using validation data and appropriate metrics.



GRADIENT BOOSTING CLASSIFIER

- Target Variable : NObeyesdad (7 levels of obesity)
- Model Evaluation

```
Accuracy: 0.950354609929078

Classification Report:
              precision    recall  f1-score   support

           0       0.93      0.93      0.93        56
           1       0.90      0.85      0.88        62
           2       0.90      0.96      0.93        56
           3       1.00      0.96      0.98        50
           4       0.97      0.96      0.97        78
           5       0.95      0.98      0.97        58
           6       1.00      1.00      1.00        63

    accuracy                           0.95       423
   macro avg       0.95      0.95      0.95       423
weighted avg       0.95      0.95      0.95       423
```

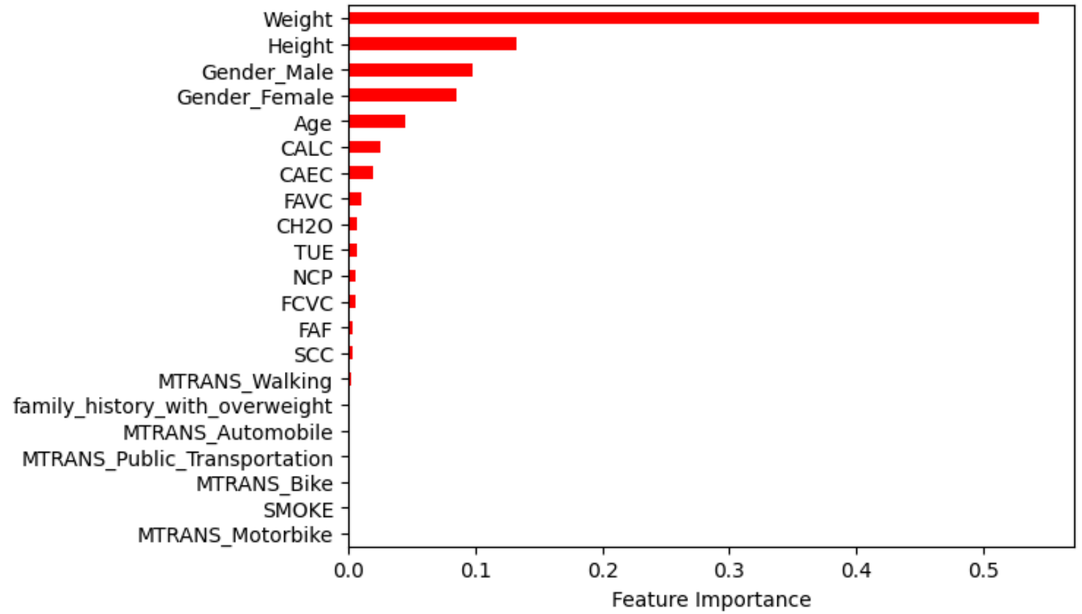
GRADIENT BOOSTING CLASSIFIER

- ## Interpretation

- **High Precision and Recall:** The model has high precision and recall across all classes, indicating that it is both accurate and comprehensive in identifying instances correctly.
- **Balanced Performance:** The F1-scores are high across all classes, suggesting balanced performance without a significant trade-off between precision and recall.
- **Perfect Scores:** For some classes (e.g., Class 4 and Class 5), the precision, recall, and F1-score are perfect (1.00), indicating that the model predicts these classes with perfect accuracy in the test set.

FEATURE IMPORTANCE

The analysis reveals that weight, gender, height, age, and alcohol consumption (CALC) are the most influential factors in predicting obesity levels.



XGBCLASSIFIER & CATBOOSTCLASSIFIER

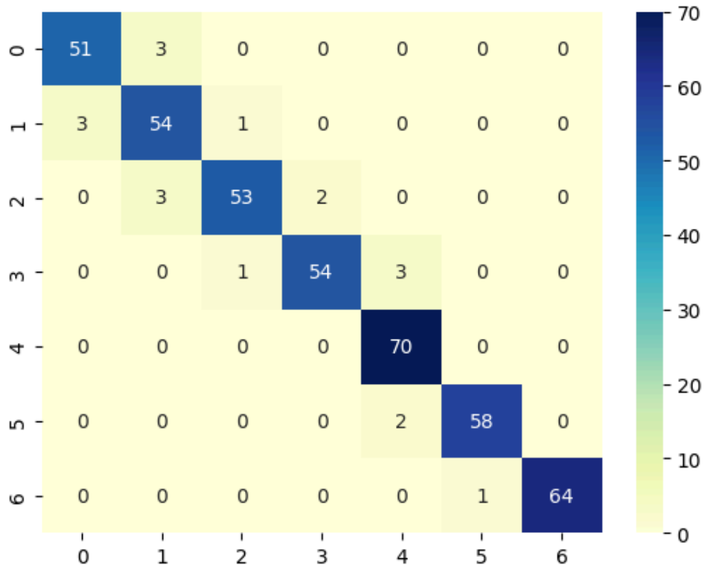
XGBOOSTCLASSIFIER

- Target Variable : NObeyesdad (7 levels of obesity)

Accuracy

```
xgb_model.score(X_test, y_test)
0.9550827423167849
```

Confusion matrix



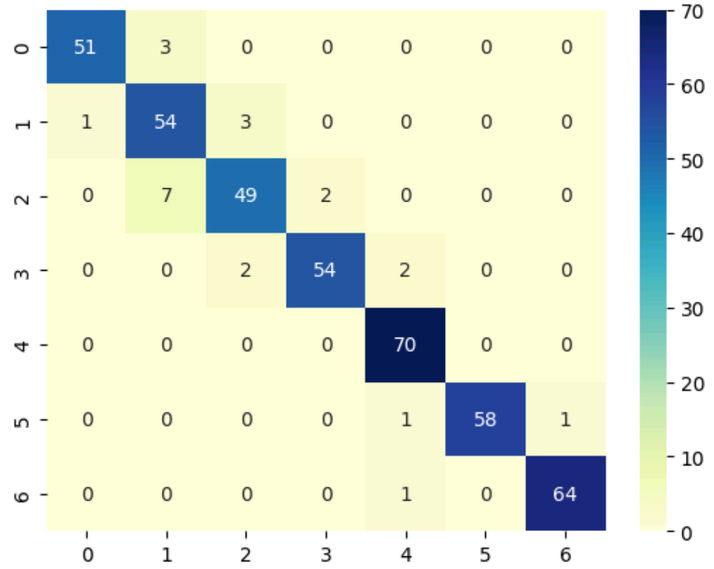
CATBOOSTCLASSIFIER

- Target Variable : NObeyesdad (7 levels of obesity)

Accuracy

```
[19] cat_model.score(X_test, y_test)
0.9456264775413712
```

Confusion matrix



STAY HEALTHY