Obesity Classification of Individuals

1. **Data Selection**

For this assignment, I have chosen a dataset from Kaggle that contains information about the obesity classification of individuals. The dataset includes the following columns:

a) ID: A unique identifier for each individual

b) Age: The age of the individual

c) Gender: The gender of the individual

d) Height: The height of the individual in centimeters

e) Weight: The weight of the individual in kilograms

f) BMI: The body mass index of the individual, calculated as weight divided by height squared

g) Label: The obesity classification of the individual, which can be one of the following: Normal Weight, Overweight, Obese or Underweight

```
import pandas as pd

# Load the dataset to examine its contents
file_path = 'Obesity Classification.csv'
data = pd.read_csv(file_path)

# Display the first few rows of the dataset
data.head()
```

	ID	Age	Gender	Height	Weight	BMI	Label
0	1	25	Male	175	80	25.3	Normal Weight
1	2	30	Female	160	60	22.5	Normal Weight
2	3	35	Male	180	90	27.3	Overweight
3	4	40	Female	150	50	20.0	Underweight
4	5	45	Male	190	100	31.2	Obese

2. **Model Development and Evaluation**

```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import classification_report

# Preprocess the dataset
# Encode the 'Gender' and 'Label' columns
le_gender = LabelEncoder()
```

```
le_label = LabelEncoder()

data['Gender'] = le_gender.fit_transform(data['Gender'])
data['Label'] = le_label.fit_transform(data['Label'])

# Split the data into features (X) and target (y)
X = data[['Age', 'Gender', 'Height', 'Weight', 'BMI']]
y = data['Label']

# Scale the features
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2, rando

# Initialize the models with Logistic Regression having increased max_iter
log_reg = LogisticRegression(max_iter=1000)
knn = KNeighborsClassifier()
dtree = DecisionTreeClassifier()

# Train the models
log_reg.fit(X_train, y_train)
knn.fit(X_train, y_train)
dtree.fit(X_train, y_train)

# Predict on the test data
y_pred_log_reg = log_reg.predict(X_test)
y_pred_knn = knn.predict(X_test)
y_pred_dtree = dtree.predict(X_test)

# Correctly generate the class names as strings for the classification report
target_names = [str(class_name) for class_name in le_label.inverse_transform(range(le

# Generate the classification reports
report_log_reg = classification_report(y_test, y_pred_log_reg, target_names=target_na
report_knn = classification_report(y_test, y_pred_knn, target_names=target_names)
report_dtree = classification_report(y_test, y_pred_dtree, target_names=target_names)

# Display the reports
print("Logistic Regression:\n", report_log_reg)
print("k-Nearest Neighbors:\n", report_knn)
print("Decision Tree:\n", report_dtree)
```

```
Logistic Regression:
                precision    recall  f1-score   support

Normal Weight       1.00      1.00      1.00         6
        Obese       1.00      0.75      0.86         4
   Overweight       0.80      1.00      0.89         4
  Underweight       1.00      1.00      1.00         8

     accuracy                           0.95        22
   macro avg       0.95      0.94      0.94        22
weighted avg       0.96      0.95      0.95        22

k-Nearest Neighbors:
                precision    recall  f1-score   support

Normal Weight       0.83      0.83      0.83         6
        Obese       1.00      0.75      0.86         4
   Overweight       0.60      0.75      0.67         4
  Underweight       1.00      1.00      1.00         8

     accuracy                           0.86        22
    macro avg       0.86      0.83      0.84        22
 weighted avg       0.88      0.86      0.87        22

Decision Tree:
                precision    recall  f1-score   support

Normal Weight       1.00      1.00      1.00         6
        Obese       1.00      1.00      1.00         4
   Overweight       1.00      1.00      1.00         4
  Underweight       1.00      1.00      1.00         8

     accuracy                           1.00        22
    macro avg       1.00      1.00      1.00        22
 weighted avg       1.00      1.00      1.00        22
```

In these findings, we can see that the decision tree performs the best, with perfect precision, recall, and F1-scores across all classes, and 100% accuracy. This suggests that the model is very effective in classifying instances correctly without any errors.

On the other hand, the logistic regression has high performance metrics but is slightly less effective than the Decision Tree. It provides a good balance of precision and recall, with very high accuracy.

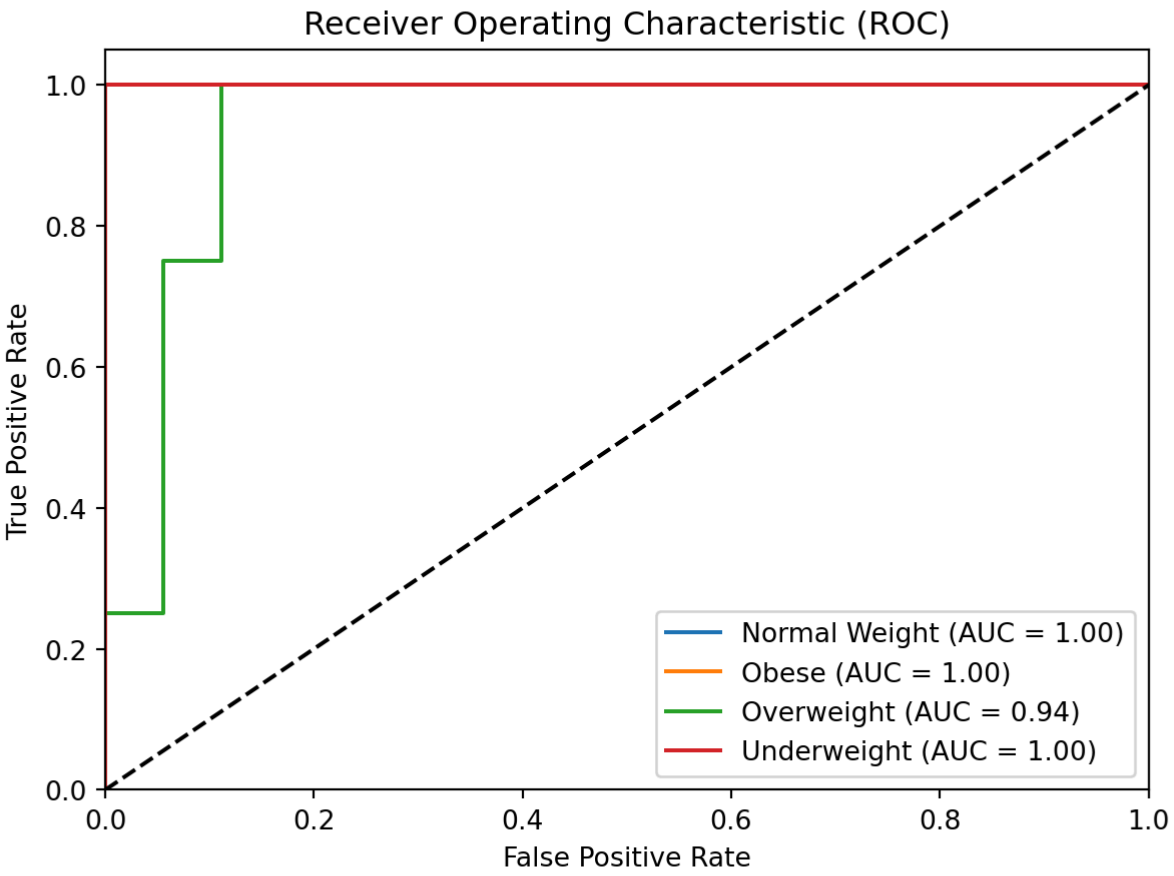
Lastly, the k-Nearest Neighbors performs well but is not as effective as Logistic Regression or Decision Tree, especially for class 2 where it has lower precision and recall.

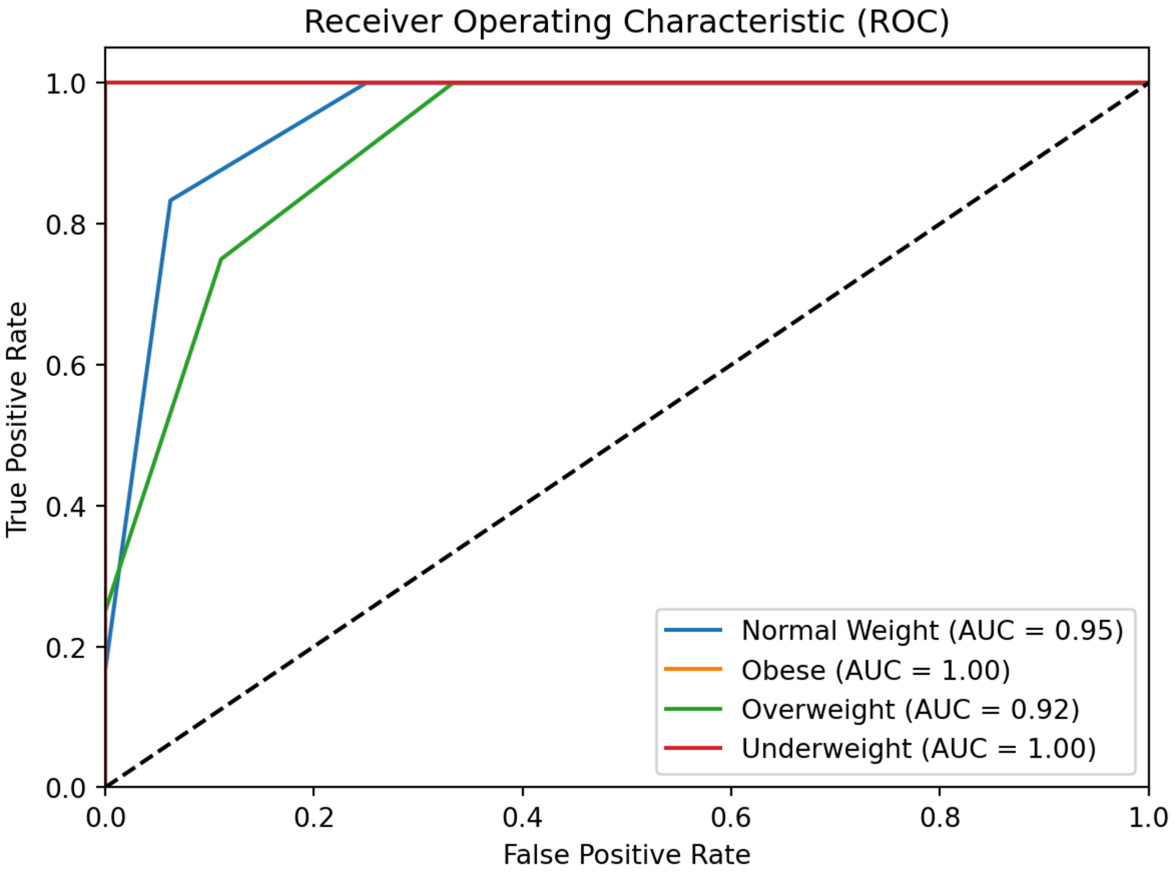
In summary, the Decision Tree model is the most accurate and balanced in this case, followed by Logistic Regression, and then k-NN.

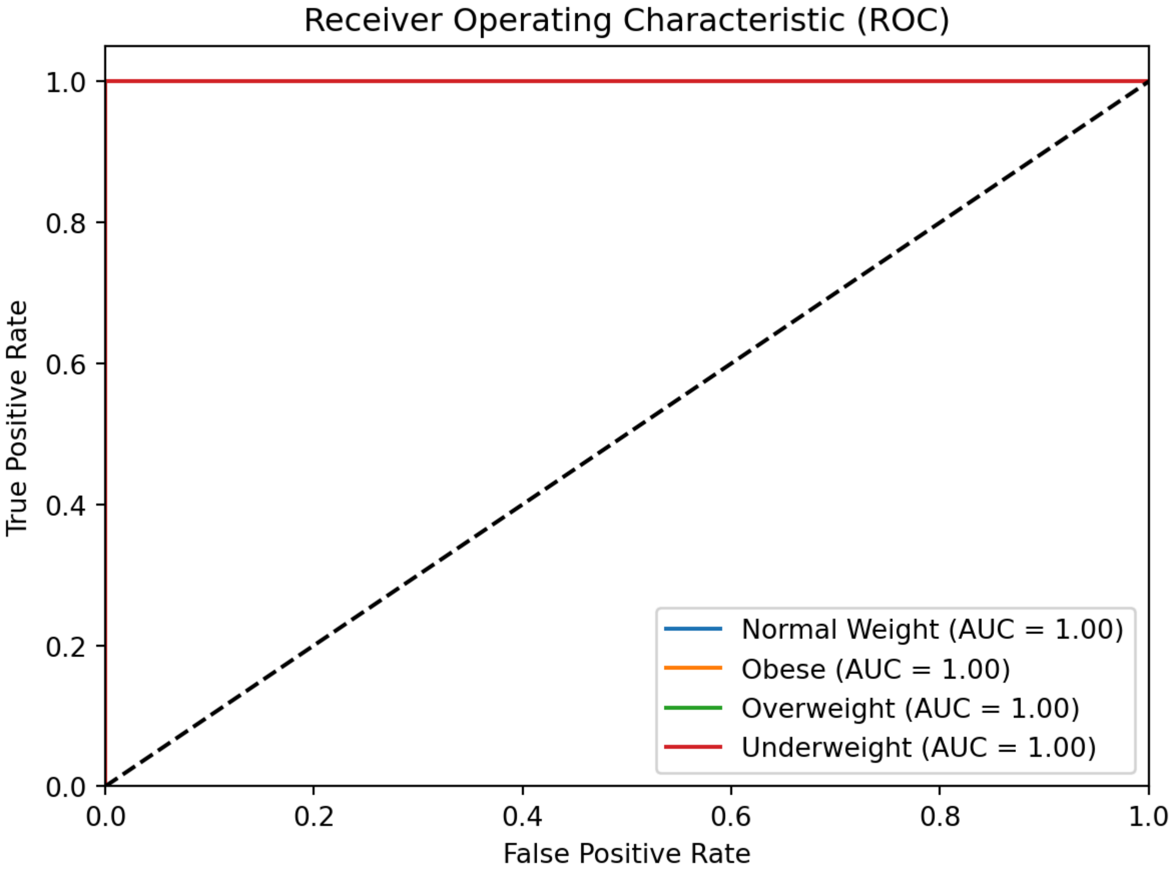
3. **Diving deeper with Confusion Matrix & ROC AUC Scores**

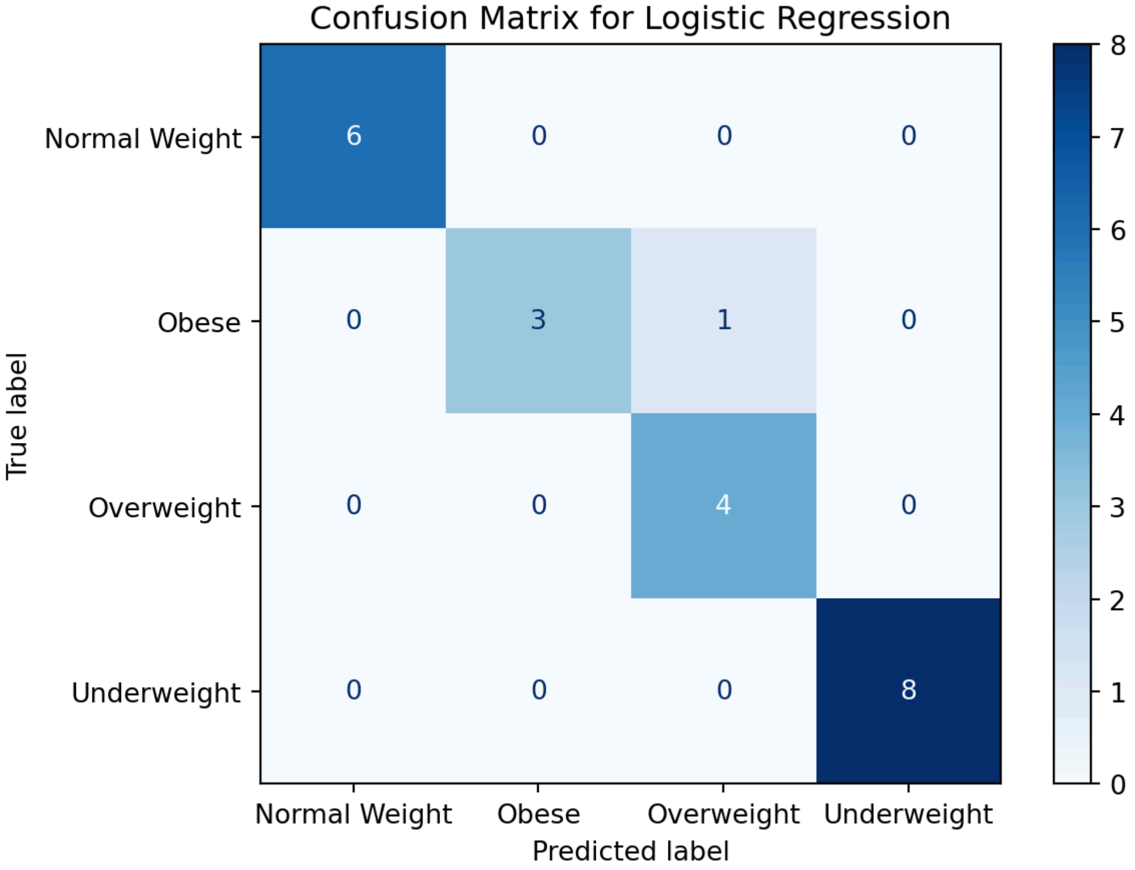
```
from sklearn.metrics import roc_curve, auc, confusion_matrix, ConfusionMatrixDisplay
import matplotlib.pyplot as plt
from sklearn.preprocessing import label_binarize

# ROC Curve for Multiclass Classification
def plot_roc_curve_multiclass(y_true, y_prob, classes):
    y_true_binarized = label_binarize(y_true, classes=range(len(classes)))
    n_classes = len(classes)

    plt.figure()
    auc_scores = {}
    for i in range(n_classes):
        fpr, tpr, _ = roc_curve(y_true_binarized[:, i], y_prob[:, i])
        roc_auc = auc(fpr, tpr)
        auc_scores[classes[i]] = roc_auc
        plt.plot(fpr, tpr, label=f'{classes[i]} (AUC = {roc_auc:.2f})')

    plt.plot([0, 1], [0, 1], 'k--')
    plt.xlim([0.0, 1.0])
    plt.ylim([0.0, 1.05])
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title('Receiver Operating Characteristic (ROC)')
    plt.legend(loc='lower right')
    plt.show()

    return auc_scores

# Predict probabilities
y_prob_log_reg = log_reg.predict_proba(X_test)
y_prob_knn = knn.predict_proba(X_test)
y_prob_dtree = dtree.predict_proba(X_test)

roc_auc_log_reg = plot_roc_curve_multiclass(y_test, y_prob_log_reg, target_names)
roc_auc_knn = plot_roc_curve_multiclass(y_test, y_prob_knn, target_names)
roc_auc_dtree = plot_roc_curve_multiclass(y_test, y_prob_dtree, target_names)

# Confusion Matrix
def plot_confusion_matrix(y_true, y_pred, label):
    cm = confusion_matrix(y_true, y_pred)
    disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=target_names)
    disp.plot(cmap=plt.cm.Blues, values_format='d')
    plt.title(f'Confusion Matrix for {label}')
    plt.show()

plot_confusion_matrix(y_test, y_pred_log_reg, 'Logistic Regression')
plot_confusion_matrix(y_test, y_pred_knn, 'k-Nearest Neighbors')
plot_confusion_matrix(y_test, y_pred_dtree, 'Decision Tree')

# Confusion Matrix
def print_confusion_matrix(y_true, y_pred, label):
    cm = confusion_matrix(y_true, y_pred)
    print(f"\nConfusion Matrix for {label}:")
    print(cm)
    return cm

cm_log_reg = print_confusion_matrix(y_test, y_pred_log_reg, 'Logistic Regression')
cm_knn = print_confusion_matrix(y_test, y_pred_knn, 'k-Nearest Neighbors')
cm_dtree = print_confusion_matrix(y_test, y_pred_dtree, 'Decision Tree')

# Display ROC AUC Scores
def print_roc_auc_scores(roc_auc_scores, label):
    print(f"\nROC AUC Scores for {label}:")
    for class_name, score in roc_auc_scores.items():
        print(f"{class_name}: {score:.2f}")
```

```
print_roc_auc_scores(roc_auc_log_reg, 'Logistic Regression')
print_roc_auc_scores(roc_auc_knn, 'k-Nearest Neighbors')
print_roc_auc_scores(roc_auc_dtree, 'Decision Tree')
```

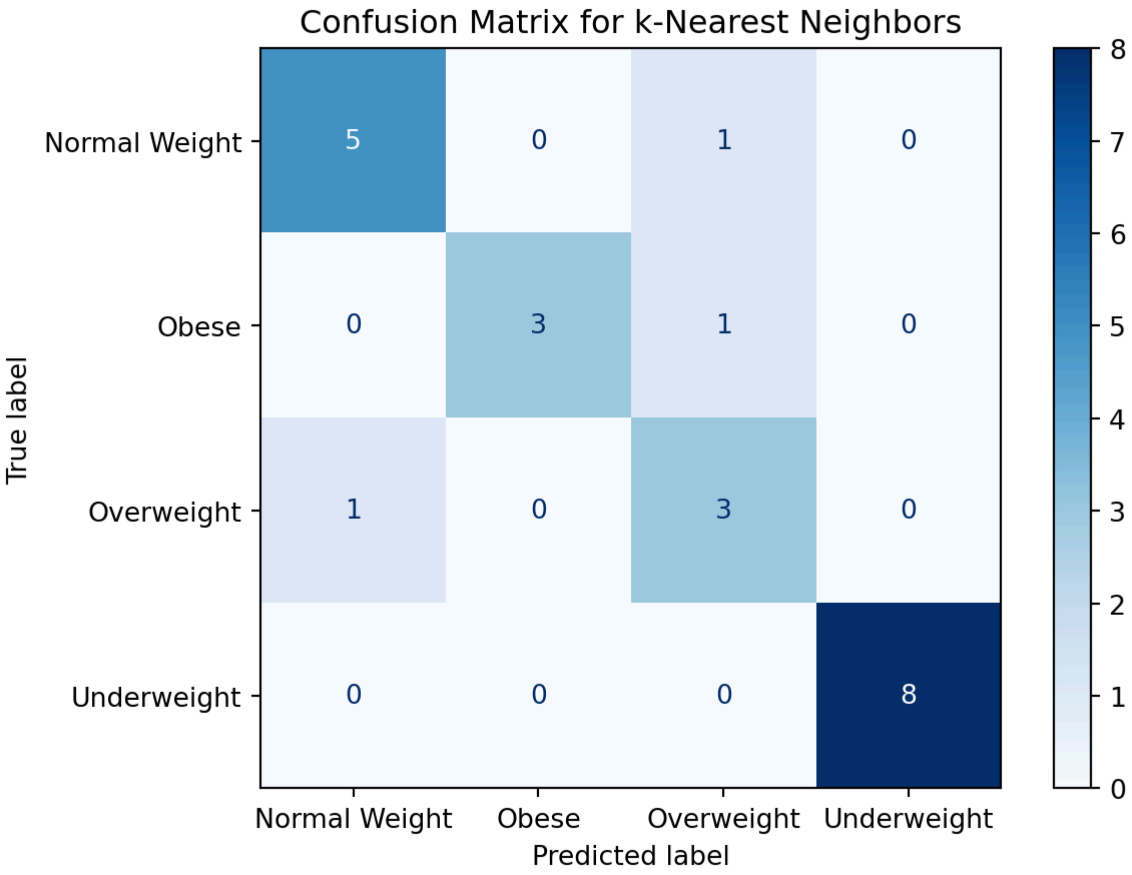


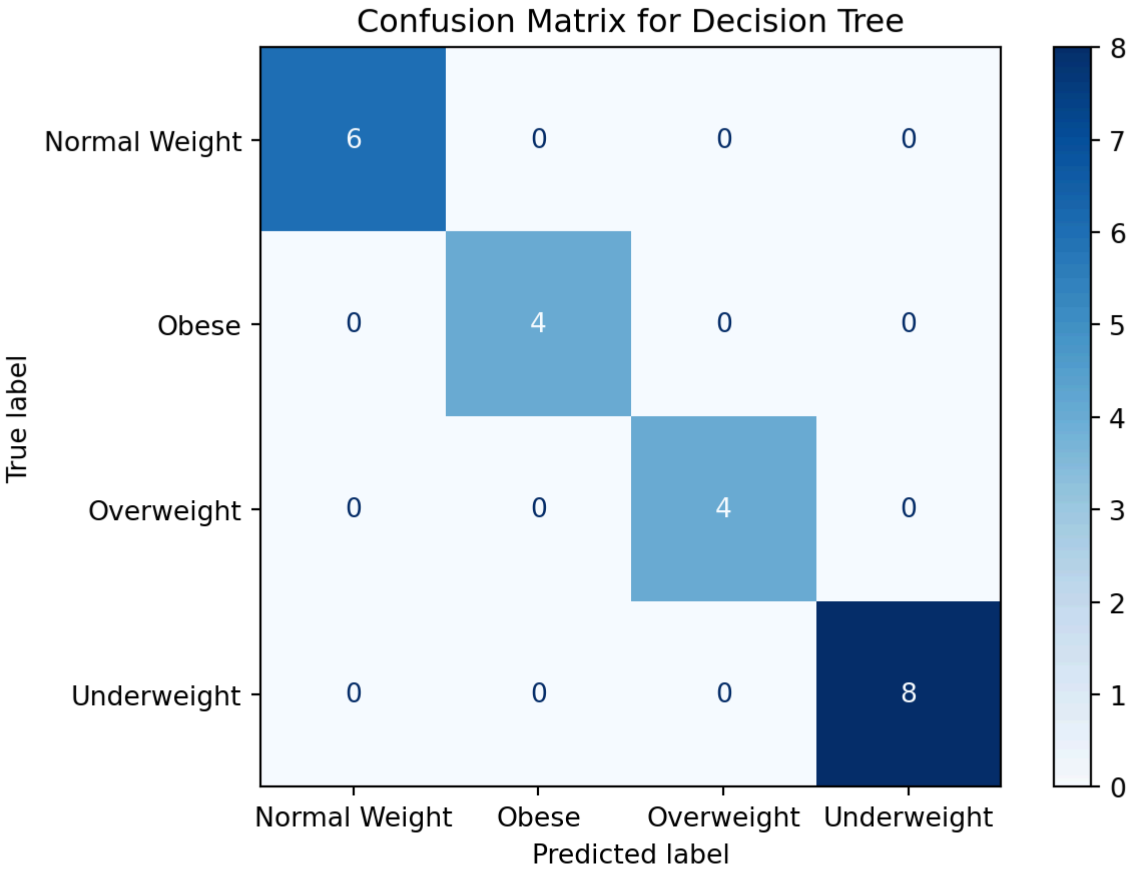






True label \ Predicted label	Normal Weight	Obese	Overweight	Underweight
Normal Weight	6	0	0	0
Obese	0	3	1	0
Overweight	0	0	4	0
Underweight	0	0	0	8





```
Confusion Matrix for Logistic Regression:
[[6 0 0 0]
 [0 3 1 0]
 [0 0 4 0]
 [0 0 0 8]]

Confusion Matrix for k-Nearest Neighbors:
[[5 0 1 0]
 [0 3 1 0]
 [1 0 3 0]
 [0 0 0 8]]

Confusion Matrix for Decision Tree:
[[6 0 0 0]
 [0 4 0 0]
 [0 0 4 0]
 [0 0 0 8]]

ROC AUC Scores for Logistic Regression:
Normal Weight: 1.00
Obese: 1.00
Overweight: 0.94
Underweight: 1.00

ROC AUC Scores for k-Nearest Neighbors:
Normal Weight: 0.95
Obese: 1.00
Overweight: 0.92
Underweight: 1.00

ROC AUC Scores for Decision Tree:
Normal Weight: 1.00
Obese: 1.00
Overweight: 1.00
Underweight: 1.00
```

Confusion Matrix Analysis

a) Logistic Regression

- **Class 0**: Perfectly classified (6 true positives, no false positives or negatives).
- **Class 1**: One instance of misclassification (1 instance incorrectly predicted as class 2).
- **Class 2**: Perfectly classified (4 true positives, no false positives or negatives).
- **Class 3**: Perfectly classified (8 true positives, no false positives or negatives).

b) k-Nearest Neighbors (k-NN)

- **Class 0**: One misclassified instance (1 instance incorrectly predicted as class 2).
- **Class 1**: One instance of misclassification (1 instance incorrectly predicted as class 2).
- **Class 2**: One instance of misclassification (1 instance incorrectly predicted as class 0).
- **Class 3**: Perfectly classified (8 true positives, no false positives or negatives).

c) Decision Tree

- **Class 0**: Perfectly classified (6 true positives, no false positives or negatives).
- **Class 1**: Perfectly classified (4 true positives, no false positives or negatives).
- **Class 2**: Perfectly classified (4 true positives, no false positives or negatives).
- **Class 3**: Perfectly classified (8 true positives, no false positives or negatives)

ROC AUC Scores Analysis

a) Logistic Regression

Excellent performance across all classes with perfect AUC scores for most classes. Slightly lower AUC for Overweight (0.94), indicating it is not as well distinguished as other classes.

b) k-Nearest Neighbors (k-NN)

Good performance with perfect AUC scores for some classes but slightly lower for Normal Weight (0.95) and Overweight (0.92), indicating these classes are less well distinguished compared to others.

c) Decision Tree

Perfect performance with AUC scores of 1.00 across all classes, indicating ideal separation for all classes.

To conclude, the best performing model is the decision tree Best performance overall with perfect confusion matrices and ROC AUC scores. It accurately classifies all instances without any misclassification and provides the best ROC AUC scores, suggesting it is the most reliable model in this scenario.

4. **Conclusion of the Study**

   The Decision Tree model outperformed Logistic Regression and k-Nearest Neighbors in this specific dataset, achieving perfect precision, recall, and F1-scores across all classes, confusion matrices, and ROC AUC scores. However, its exceptional performance raises concerns about overfitting, especially in small or perfectly separable datasets.

   Logistic Regression also performed admirably, offering a balance between interpretability and accuracy, making it a solid choice for datasets where linear relationships are sufficient. On the other hand, k-Nearest Neighbors, while flexible and simple, showed some variability in performance, particularly with overlapping classes.

   For future applications, Decision Trees may be best suited when non-linear relationships are present, and careful pruning or ensemble methods are used to prevent overfitting. Logistic Regression remains a reliable choice for linear problems, while k-NN could be beneficial when non-parametric methods are needed, but with attention to scaling and class balance.

5. **Strengths and Weaknesses of each model**

   a) Logistic Regression

Strengths	Weaknesses
Interpretability –> Log. Regression provides easily interpretable results,	**Performance on Overweight Class –>** Slightly lower ROC AUC score for the Overweight class (0.94),

Strengths	Weaknesses
making it easier to understand the effect of each feature on the outcome.	indicating that the model might have more difficulty distinguishing this class from others compared to other classes.
Performance –> High precision rate and recall for most classes, with high overall accuracy (95%) and good F1-scores.	**Linear Boundaries –>** Logistic Regression assumes linear decision boundaries between classes, which might not capture complex relationships in the data.

b) k-Nearest Neighbors (k-NN)

Strengths	Weaknesses
Flexibility –> k-NN can capture non-linear decision boundaries and adapt to various patterns in the data.	**Performance –>** Lower performance compared to other models, with some misclassifications (e.g., Normal Weight and Overweight classes). Lower ROC AUC scores for Normal Weight (0.95) and Overweight (0.92) indicate less effective class separation.
Simplicity –> Conceptually simple and easy to implement.	**Sensitivity to Data Scaling –>** k-NN performance can be significantly impacted by the scale of the features, requiring careful feature scaling.

c) Decision Tree

Strengths	Weaknesses
Performance –> Perfect performance in confusion matrices and ROC AUC scores (1.00) across all classes. It accurately classifies all instances without misclassifications.	**Overfitting –>** Decision Trees can easily overfit the training data, especially with deep trees and small datasets. They might not generalize well to unseen data if not properly pruned or regularized.
Non-linear Relationships –> Can capture complex, non-linear relationships between features.	**Stability –>** Small changes in the data can lead to significantly different tree structures, making Decision Trees less stable compared to other models.