Table of Contents

Task description

Develop a machine learning model capable of generating multi-class predictions on a health-related dataset.

The evaluation metric is the accuracy score.

My objective is to built a model that achieves a validation score exceeding 90.5%, providing a stable result across test and train data sets.

IMPORTS

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
from sklearn.metrics import accuracy_score, roc_auc_score, f1_score, precision_score, re
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import train_test_split
import lightgbm as lgb
from sklearn.metrics import roc_auc_score
import os
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))
```

```
/kaggle/input/playground-series-s4e2/sample_submission.csv
/kaggle/input/playground-series-s4e2/train.csv
/kaggle/input/playground-series-s4e2/test.csv
```

In [2]:
```
train=pd.read_csv("/kaggle/input/playground-series-s4e2/train.csv")
test=pd.read_csv("/kaggle/input/playground-series-s4e2/test.csv")
```

Dataset overview

In [3]:
```
print('Train shape:', train.shape)
print('Test shape: ', test.shape)
```
```
Train shape: (20758, 18)
Test shape:  (13840, 17)
```

In [4]:
```
train.head()
```

Out[4]:

	id	Gender	Age	Height	Weight	family_history_with_overweight	FAVC	FCVC	NCP	CA
0	0	Male	24.443011	1.699998	81.669950	yes	yes	2.000000	2.983297	Sometir
1	1	Female	18.000000	1.560000	57.000000	yes	yes	2.000000	3.000000	Frequei
2	2	Female	18.000000	1.711460	50.165754	yes	yes	1.880534	1.411685	Sometir
3	3	Female	20.952737	1.710730	131.274851	yes	yes	3.000000	3.000000	Sometir
4	4	Male	31.641081	1.914186	93.798055	yes	yes	2.679664	1.971472	Sometir

In [5]:
```
train.info()
```
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 20758 entries, 0 to 20757
Data columns (total 18 columns):
 #   Column                          Non-Null Count  Dtype  
---  ------                          --------------  -----  
 0   id                              20758 non-null  int64  
 1   Gender                          20758 non-null  object 
 2   Age                             20758 non-null  float64
 3   Height                          20758 non-null  float64
 4   Weight                          20758 non-null  float64
 5   family_history_with_overweight  20758 non-null  object 
 6   FAVC                            20758 non-null  object 
 7   FCVC                            20758 non-null  float64
 8   NCP                             20758 non-null  float64
 9   CAEC                            20758 non-null  object 
 10  SMOKE                           20758 non-null  object 
 11  CH2O                            20758 non-null  float64
 12  SCC                             20758 non-null  object 
 13  FAF                             20758 non-null  float64
 14  TUE                             20758 non-null  float64
 15  CALC                            20758 non-null  object 
 16  MTRANS                          20758 non-null  object 
 17  NObeyesdad                      20758 non-null  object 
dtypes: float64(8), int64(1), object(9)
memory usage: 2.9+ MB
```

In [6]:
```
train.describe().T
```

Out[6]:

	count	mean	std	min	25%	50%	75%	max
id	20758.0	10378.500000	5992.462780	0.00	5189.250000	10378.500000	15567.750000	20757.000000
Age	20758.0	23.841804	5.688072	14.00	20.000000	22.815416	26.000000	61.000000
Height	20758.0	1.700245	0.087312	1.45	1.631856	1.700000	1.762887	1.975663
Weight	20758.0	87.887768	26.379443	39.00	66.000000	84.064875	111.600553	165.057269
FCVC	20758.0	2.445908	0.533218	1.00	2.000000	2.393837	3.000000	3.000000
NCP	20758.0	2.761332	0.705375	1.00	3.000000	3.000000	3.000000	4.000000
CH2O	20758.0	2.029418	0.608467	1.00	1.792022	2.000000	2.549617	3.000000

FAF	20758.0	0.981747	0.838302	0.00	0.008013	1.000000	1.587406	3.000000
TUE	20758.0	0.616756	0.602113	0.00	0.000000	0.573887	1.000000	2.000000

In [7]:

```
print('Train NONE elements:', train.isnull().sum().sum())
print('Test NONE elements:', test.isnull().sum().sum())
```

```
Train NONE elements: 0
Test NONE elements: 0
```

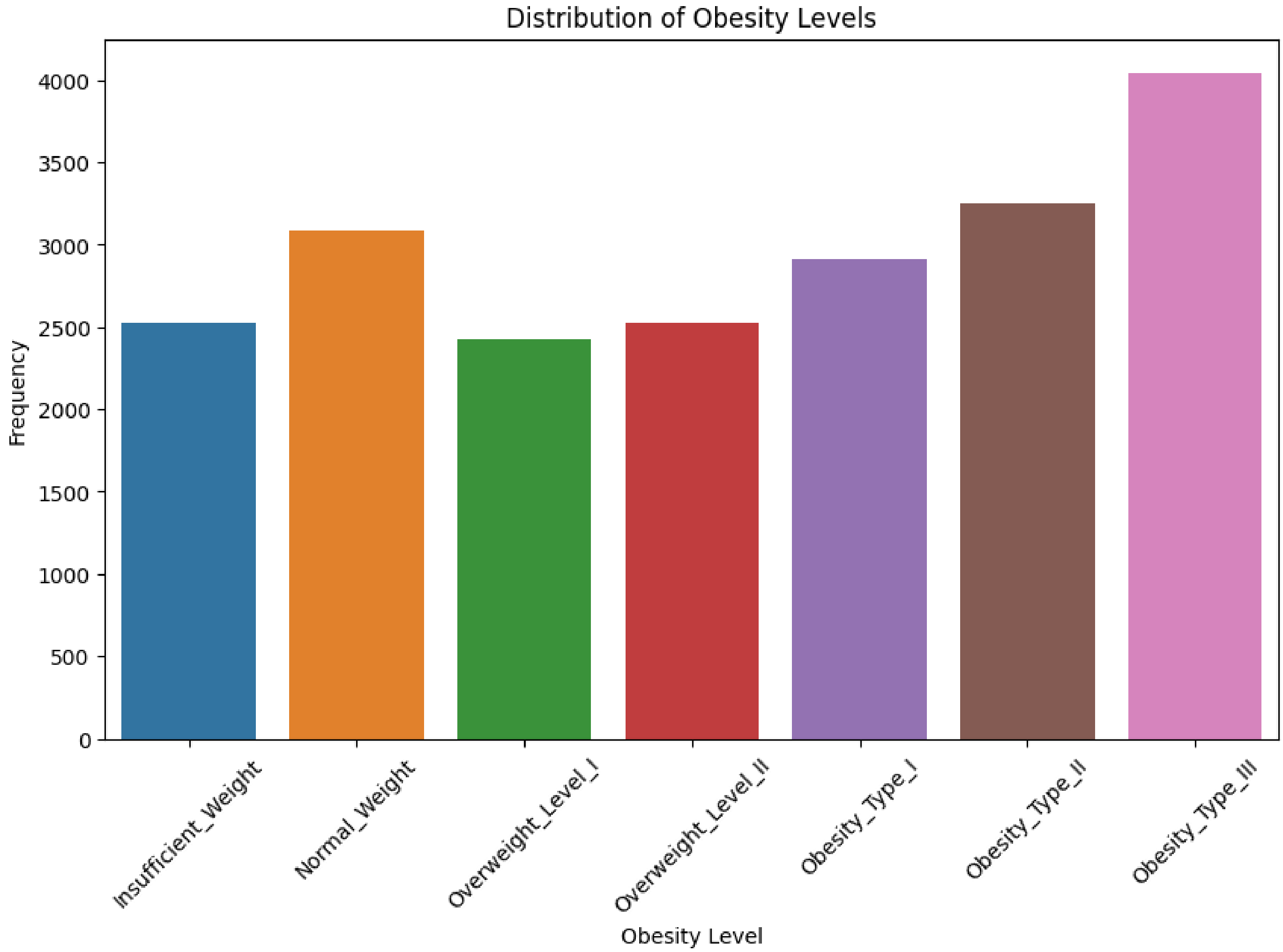
This dataset is relatively compact and free of missing values. However, it is characterized by a substantial number of categorical values that necessitate careful handling to accurately interpret and utilize.

Dataset analysis (EDA)

In [8]:

```
category_order = [
    'Insufficient_Weight',
    'Normal_Weight',
    'Overweight_Level_I',
    'Overweight_Level_II',
    'Obesity_Type_I',
    'Obesity_Type_II',
    'Obesity_Type_III']

plt.figure(figsize=(10, 6))
sns.countplot(data=train, x='NObeyesdad', order=category_order)
plt.title('Distribution of Obesity Levels')
plt.xlabel('Obesity Level')
plt.ylabel('Frequency')
plt.xticks(rotation=45)
plt.show()
```

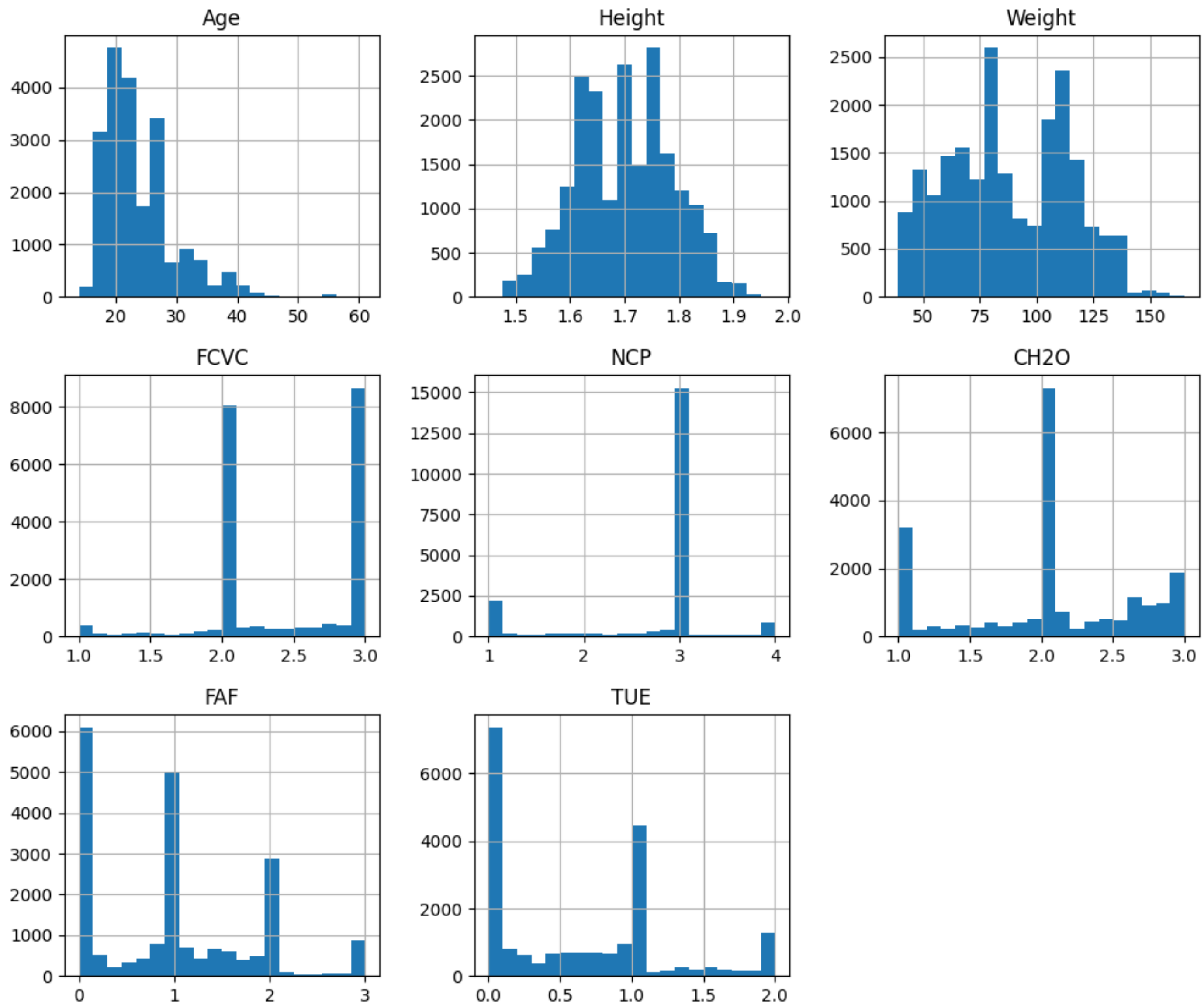


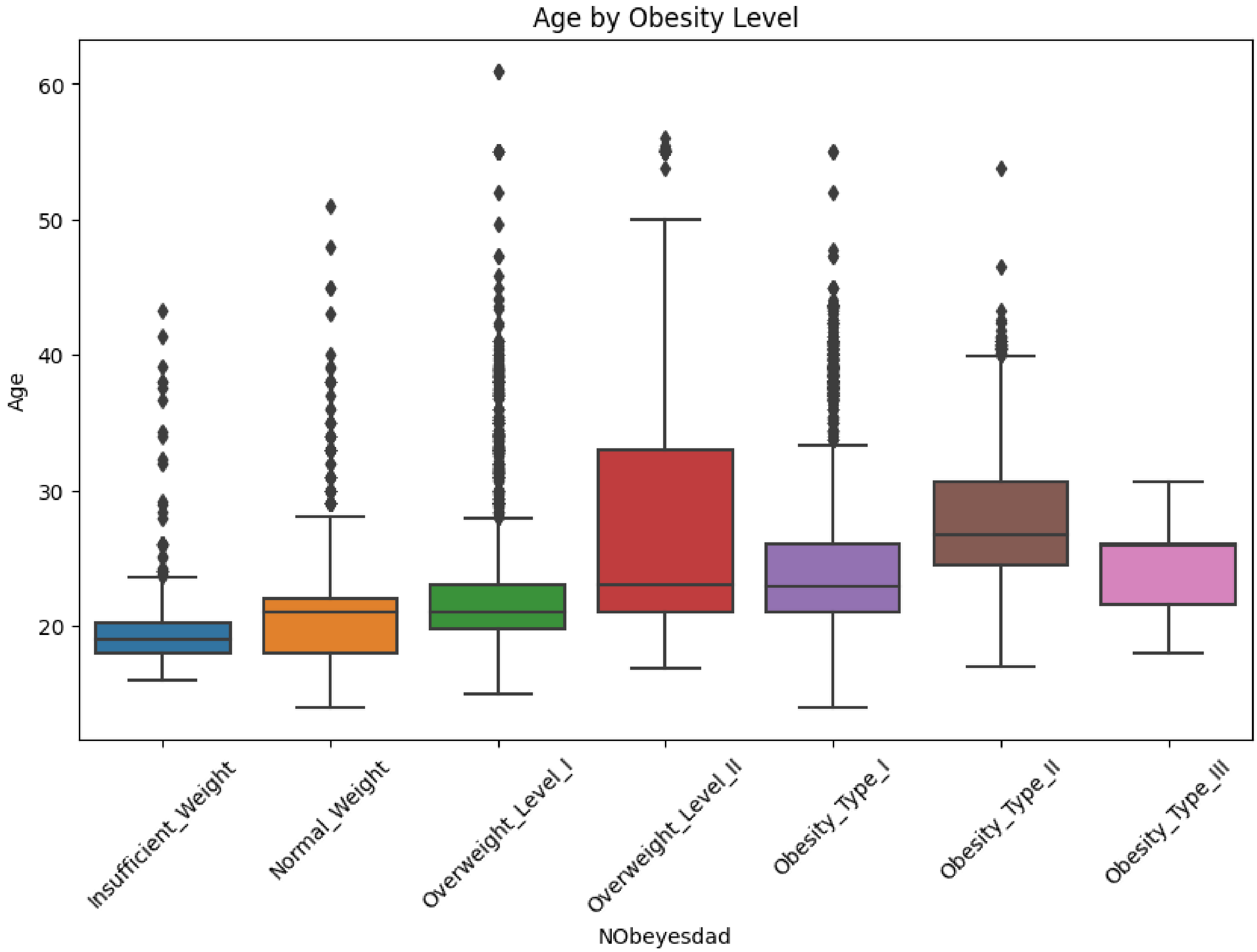
The target value distribution is a little bit unbalanced, but not critical.

In [9]:
```
train = train.drop(['id'], axis=1)
```

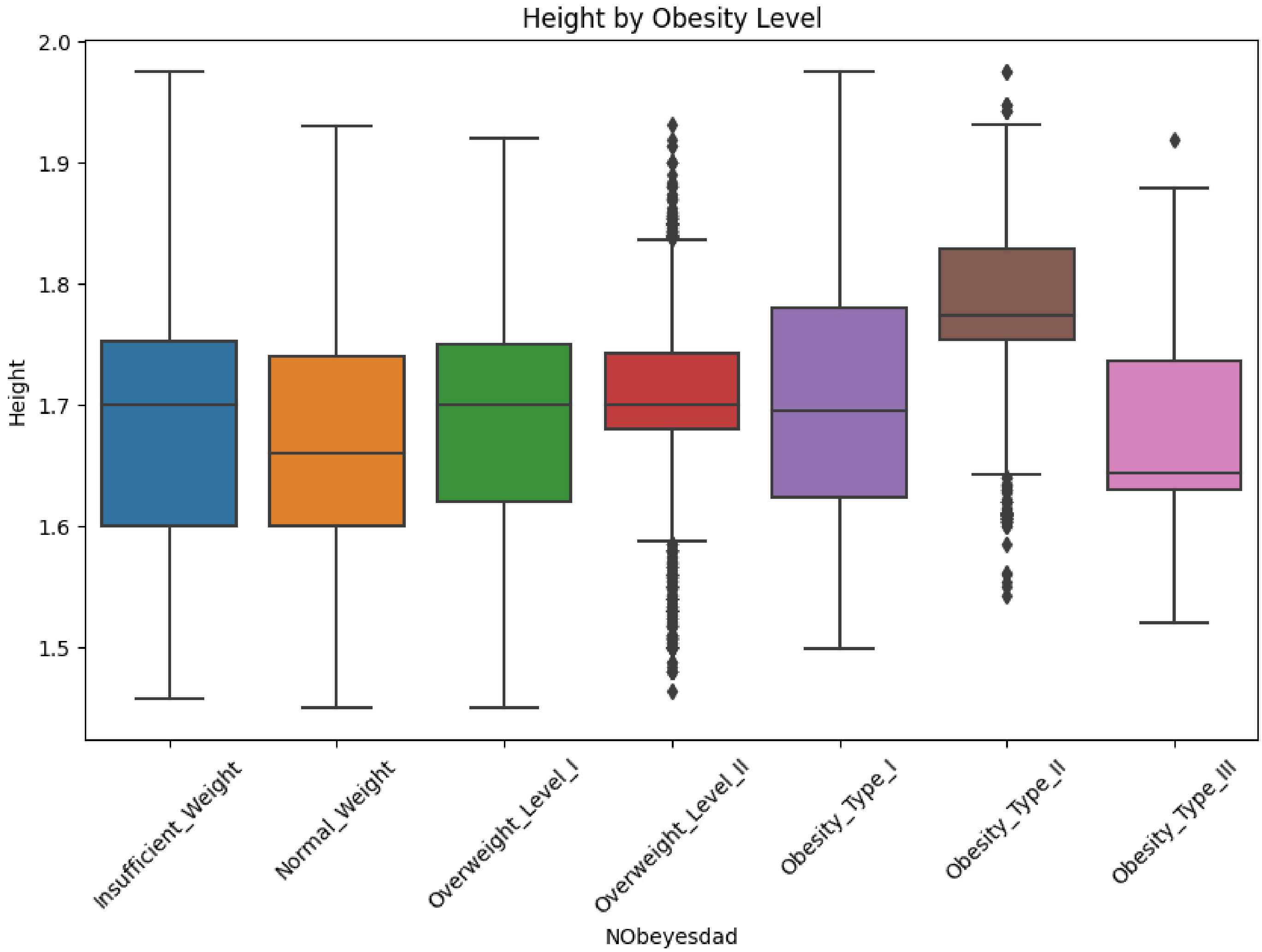
In [10]:
```
train.hist(figsize=(12, 10), bins=20)
plt.show()

# Boxplots for numerical features by 'NObeyesdad'
numerical_features = train.select_dtypes(include=['int64', 'float64']).columns.tolist()
for col in numerical_features:
    plt.figure(figsize=(10, 6))
    sns.boxplot(data=train, x='NObeyesdad', y=col, order=category_order)
    plt.title(f'{col} by Obesity Level')
    plt.xticks(rotation=45)
    plt.show()
```

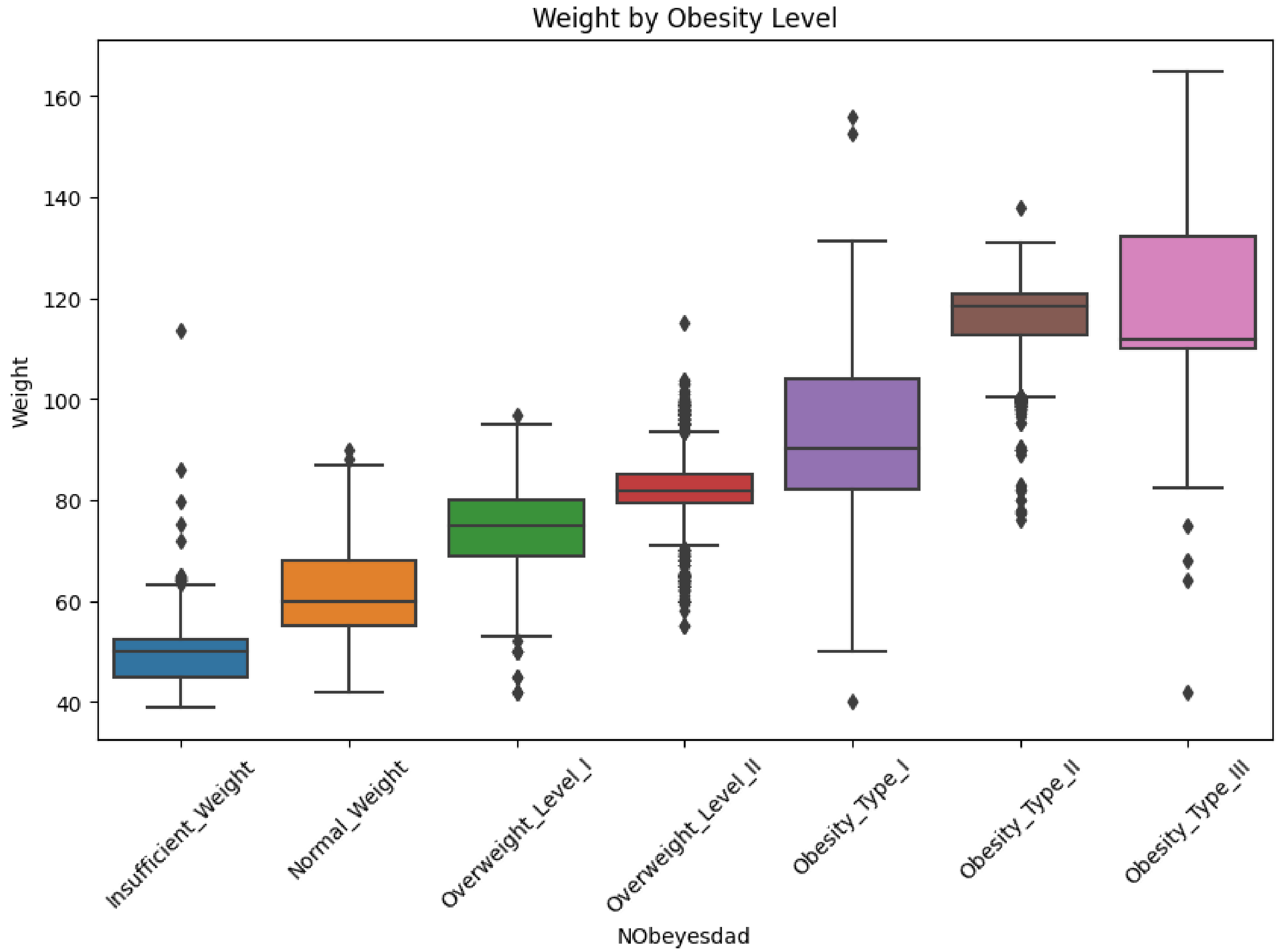

Age
Height
Weight
FCVC
NCP
CH2O
FAF
TUE
0
1000
2000
3000
4000
20
30
40
50
60
500
1500
2500
1.5
1.6
1.7
1.8
1.9
2.0
75
100
125
150
6000
8000
1.0
2.5
3.0
5000
7500
10000
12500
15000



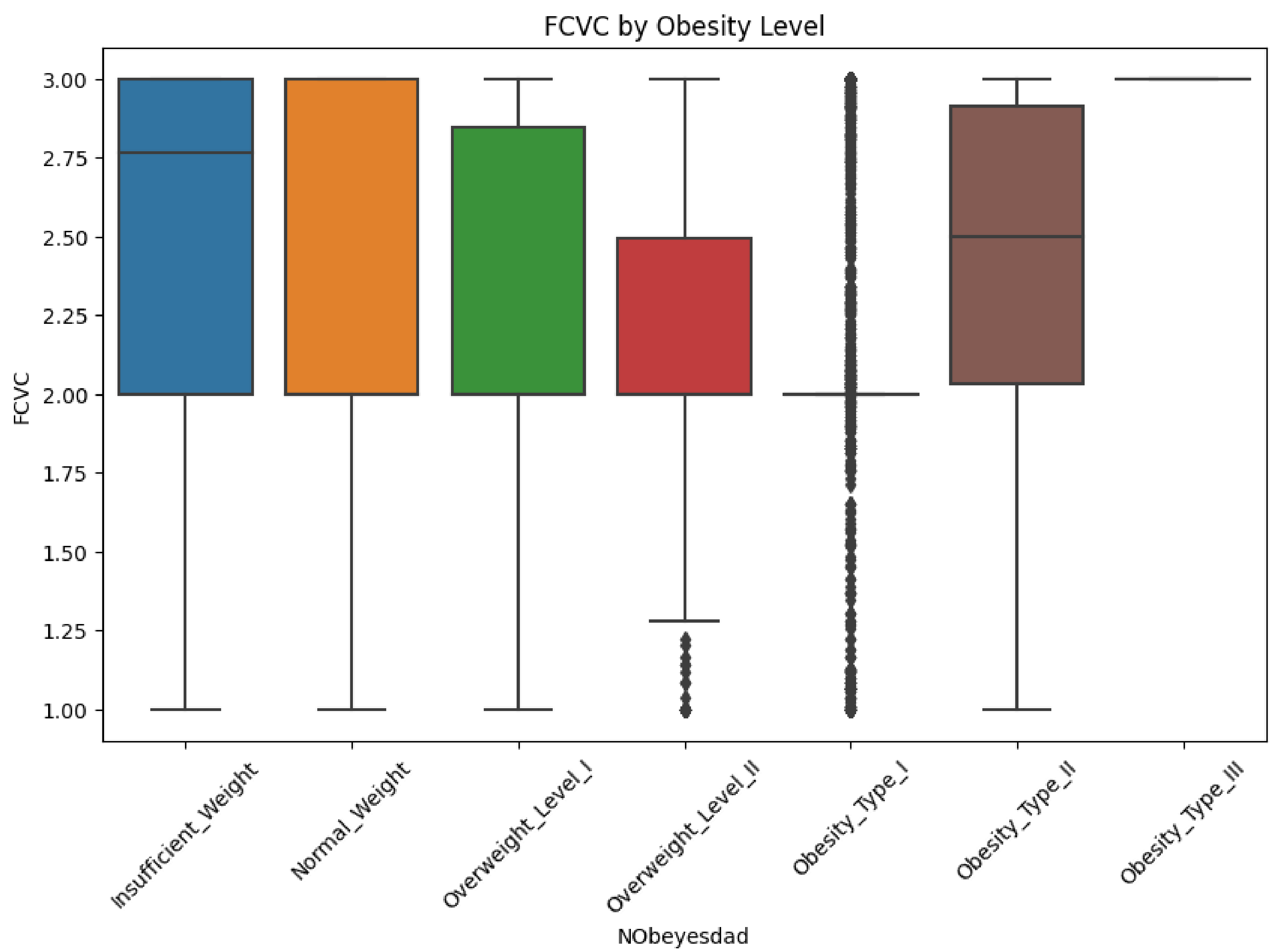
Age by Obesity Level
60
50
40
30
20
Age
Insufficient_Weight
Normal_Weight
Overweight_Level_I
Overweight_Level_II
Obesity_Type_I
Obesity_Type_II
Obesity_Type_III
NObeyesdad



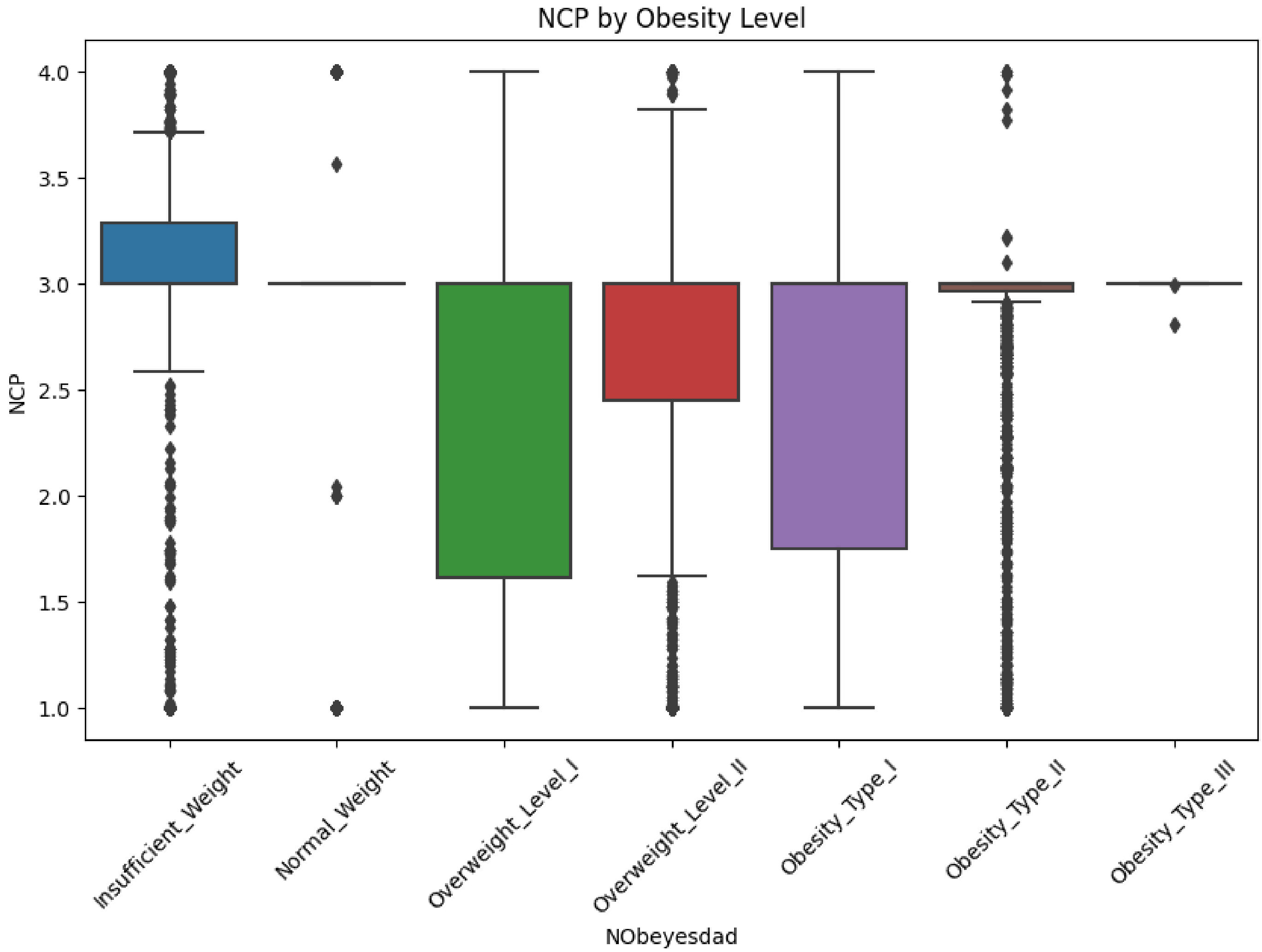
Height by Obesity Level
2.0
1.9
1.8
1.7
1.6
1.5
Height
Insufficient_Weight
Normal_Weight
Overweight_Level_I
Overweight_Level_II
Obesity_Type_I
Obesity_Type_II
Obesity_Type_III
NObeyesdad

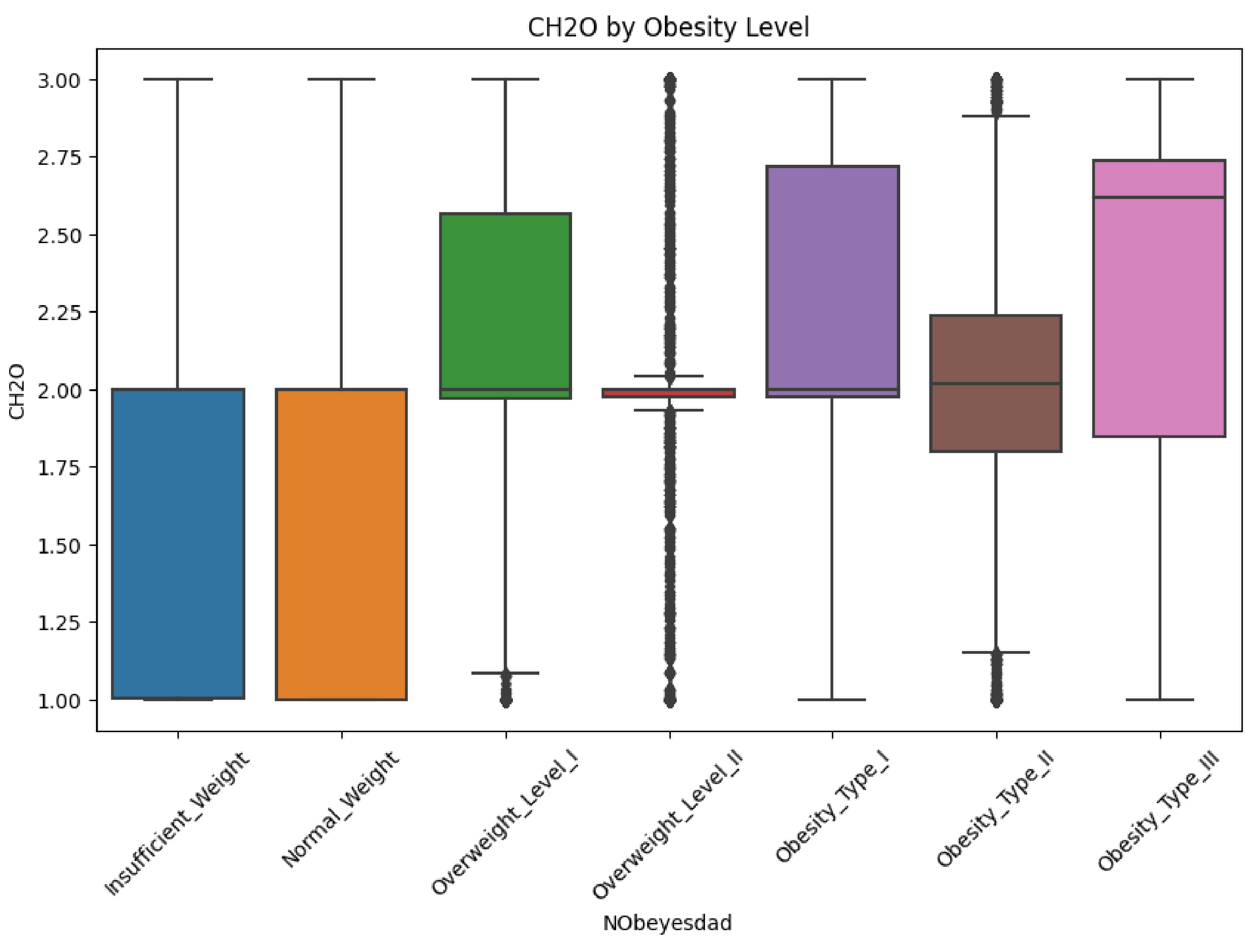


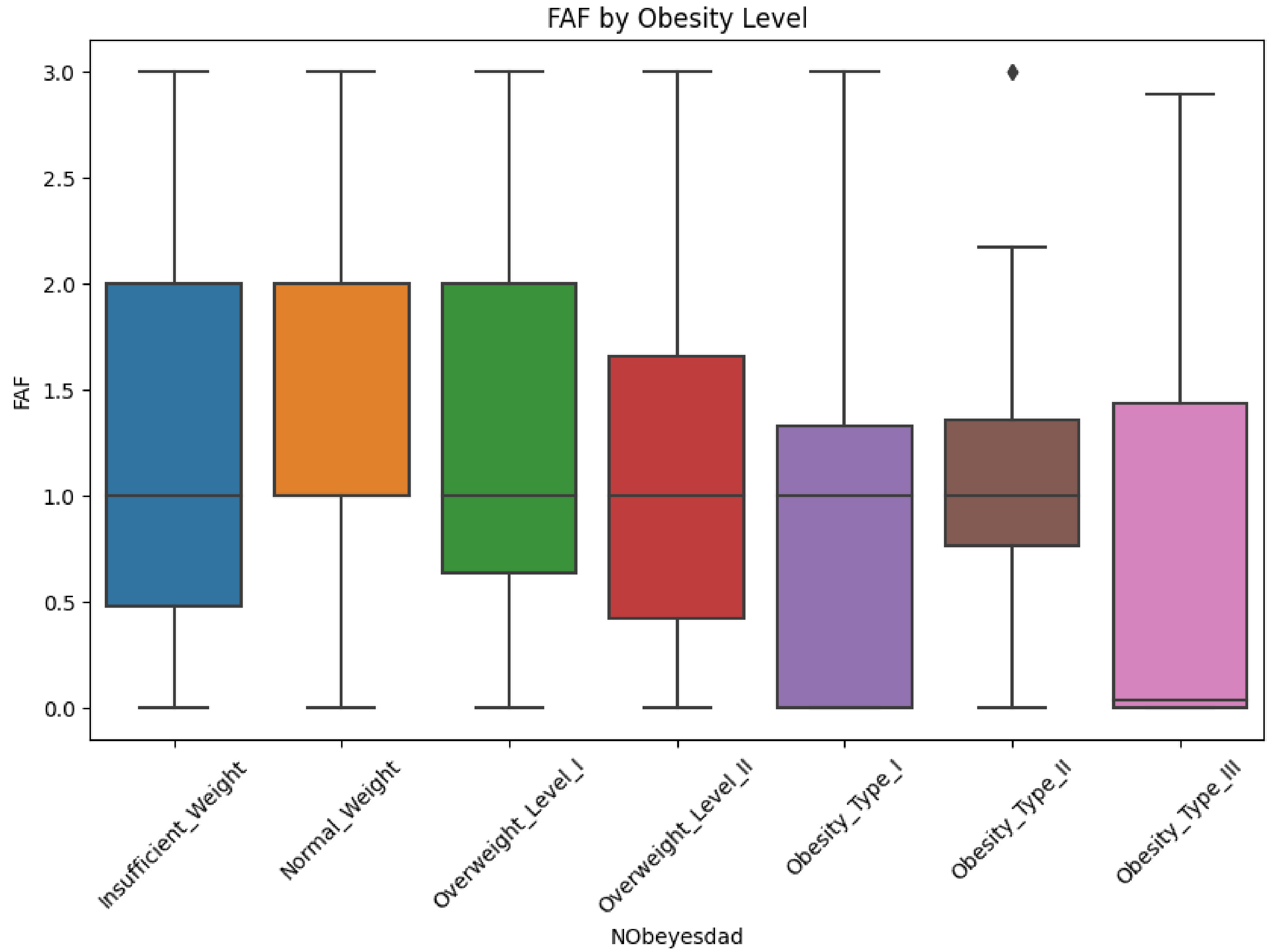
Weight by Obesity Level
Weight
160
140
120
100
80
60
40
Insufficient_Weight
Normal_Weight
Overweight_Level_I
Overweight_Level_II
Obesity_Type_I
Obesity_Type_II
Obesity_Type_III
NObeyesdad

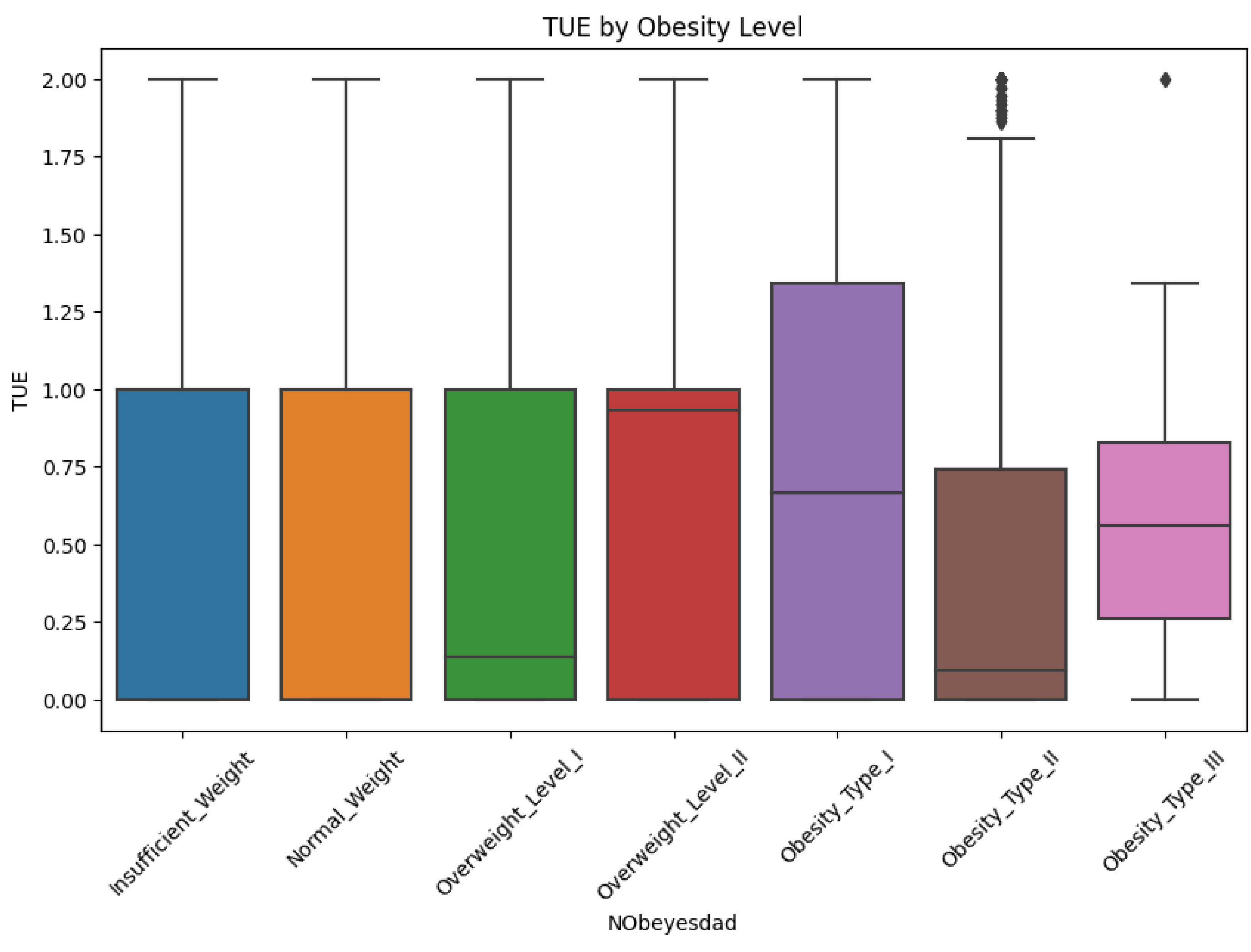


FCVC by Obesity Level
FCVC
3.00
2.75
2.50
2.25
2.00
1.75
1.50
1.25
1.00
Insufficient_Weight
Normal_Weight
Overweight_Level_I
Overweight_Level_II
Obesity_Type_I
Obesity_Type_II
Obesity_Type_III
NObeyesdad

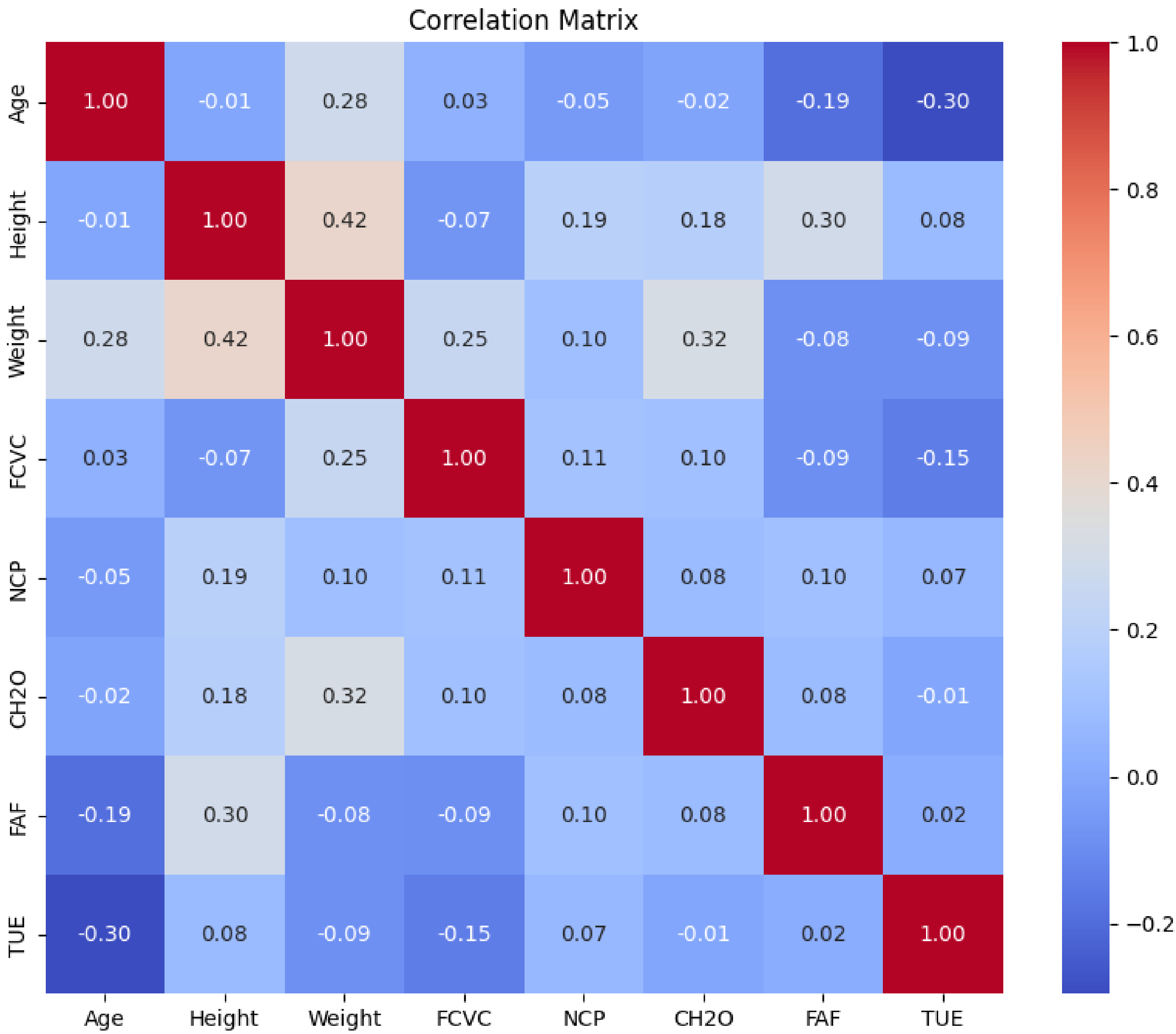

NCP by Obesity Level
NCP
4.0
3.5
3.0
2.5
2.0
1.5
1.0
Insufficient_Weight
Normal_Weight
Overweight_Level_I
Overweight_Level_II
Obesity_Type_I
Obesity_Type_II
Obesity_Type_III
NObeyesdad


CH2O by Obesity Level
CH2O
3.00
2.75
2.50
2.25
2.00
1.75
1.50
1.25
1.00
Insufficient_Weight
Normal_Weight
Overweight_Level_I
Overweight_Level_II
Obesity_Type_I
Obesity_Type_II
Obesity_Type_III
NObeyesdad


FAF by Obesity Level
3.0
2.5
2.0
1.5
1.0
0.5
0.0
FAF
Insufficient_Weight
Normal_Weight
Overweight_Level_I
Overweight_Level_II
Obesity_Type_I
Obesity_Type_II
Obesity_Type_III
NObeyesdad


TUE by Obesity Level
2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
TUE
Insufficient_Weight
Normal_Weight
Overweight_Level_I
Overweight_Level_II
Obesity_Type_I
Obesity_Type_II
Obesity_Type_III
NObeyesdad

In [11]:
```
plt.figure(figsize=(10, 8))
sns.heatmap(train[numerical_features].corr(), annot=True, fmt=".2f", cmap='coolwarm')
plt.title('Correlation Matrix')
plt.show()
```



Numerous features within the dataset deviate from normal distribution and are marked by the presence of outliers.

Despite these findings, efforts to address these characteristics through outlier removal or special handling for data normalization did not yield improvements in validation set performance. Additionally, insights garnered from competition discussions indicate that attempts at feature engineering have similarly failed to enhance predictive accuracy.

Preprocessing, data transformation, features engineering

In [12]:
```
# Target label encoding
target_encoding = {'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
                   }
train['NObeyesdad'] = train['NObeyesdad'].map(target_encoding)
```

In [13]:
```
cat_col = []

for i in test.columns:
    if test[i].dtype == 'object' :
        cat_col.append(i)
    else :
        continue

print("Categorical Columns : ", *cat_col,sep="\n")
```

```
Categorical Columns :
Gender
family_history_with_overweight
FAVC
CAEC
SMOKE
SCC
CALC
MTRANS
```

In [14]:
```
for col in cat_col:
    print(col, sorted(train[col].unique()))
    print(col, sorted(test[col].unique()))
```

```
Gender ['Female', 'Male']
Gender ['Female', 'Male']
family_history_with_overweight ['no', 'yes']
family_history_with_overweight ['no', 'yes']
FAVC ['no', 'yes']
FAVC ['no', 'yes']
CAEC ['Always', 'Frequently', 'Sometimes', 'no']
CAEC ['Always', 'Frequently', 'Sometimes', 'no']
SMOKE ['no', 'yes']
SMOKE ['no', 'yes']
SCC ['no', 'yes']
SCC ['no', 'yes']
CALC ['Frequently', 'Sometimes', 'no']
CALC ['Always', 'Frequently', 'Sometimes', 'no']
MTRANS ['Automobile', 'Bike', 'Motorbike', 'Public_Transportation', 'Walking']
MTRANS ['Automobile', 'Bike', 'Motorbike', 'Public_Transportation', 'Walking']
```

According to the information from previous steps, I'll adopt an encoding strategy for varia- bles, based on their health implications: lower numerical values signify healthier choice or behavior. Example:SCC - Smart Calories Consumption/ Calories monitoring is a positive health behavior, hence it is encoded with a value of "0" if a patient follows this practice.

In [15]:
```
def binary_encode(df, columns, positive_values):
    for column, positive_value in zip(columns, positive_values):
        df[column] = df[column].apply(lambda x: 1 if x == positive_value else 0)
    return df

def ordinal_encode(df, columns, orderings):
    for column, ordering in zip(columns, orderings):
```

```
        df[column] = df[column].apply(lambda x: ordering.index(x))
    return df

def onehot_encode(df, columns, prefixes):
    for column, prefix in zip(columns, prefixes):
        dummies = pd.get_dummies(df[column], prefix=prefix).astype(int)
        df = pd.concat([df, dummies], axis=1)
        df = df.drop(column, axis=1)
    return df

def preprocess_inputs(df):

    binary_features = [
        'family_history_with_overweight',
        'FAVC',
        'SMOKE',
        'SCC']
    positive_values = [
        'yes',
        'yes',
        'yes',
        'no']

    ordinal_features = [
        'CAEC',
        'CALC',
        'MTRANS']
    orderings = [
        ['no', 'Sometimes', 'Frequently', 'Always'],
        ['no', 'Sometimes', 'Frequently', 'Always'],
        ['Bike', 'Walking', 'Public_Transportation', 'Motorbike', 'Automobile']]

    nominal_features = [
        'Gender']
    prefixes = [
        'Gender']

    # Encodeing
    df = binary_encode(df, columns=binary_features, positive_values=positive_values)
    df = ordinal_encode(df, columns=ordinal_features, orderings=orderings)
    df = onehot_encode(df, columns=nominal_features, prefixes=prefixes)

    return df

X_train = preprocess_inputs(train)
X_VAL = preprocess_inputs(test)
```

Model training

Drawing on the insights from the research paper "Estimation of obesity levels based on dietary habits and condition physical using computational intelligence," available at: https://www.sciencedirect.com/science/article/pii/S2352914822000521

I've identified the Light Gradient Boosting Machine (LightGBM) as the most suitable model for our dataset.

In [16]:
```
X = X_train.drop(['NObeyesdad'], axis=1)
y = X_train['NObeyesdad']

scaler = MinMaxScaler()
```

```
X_norm = scaler.fit_transform(X)

X_train, X_test, y_train, y_test = train_test_split(X_norm, y, test_size=0.2, random_sta
```

In [17]:
```
lgb_classifier = lgb.LGBMClassifier(objective='multiclass',
                                    min_data_in_leaf=20,
                                    verbosity=-1,
                                    random_state=42,
                                    num_class=7,
                                    learning_rate=0.01,
                                    n_estimators=1000,
                                    max_depth=10,
                                    colsample_bytree=0.4)


lgb_classifier.fit(X_train, y_train)
```

Out[17]:
```
▾ LGBMClassifier
LGBMClassifier(colsample_bytree=0.4, learning_rate=0.01, max_depth=10,
               min_data_in_leaf=20, n_estimators=1000, num_class=7,
               objective='multiclass', random_state=42, verbosity=-1)
```

In [18]:
```
y_pred = lgb_classifier.predict(X_test)
accuracy_e = accuracy_score(y_test, y_pred)
print(f"Model Accuracy: {accuracy_e}")
print(classification_report(y_test, y_pred))
```

```
Model Accuracy: 0.9094412331406551
              precision    recall  f1-score   support

           0       0.95      0.94      0.95       524
           1       0.89      0.90      0.89       626
           2       0.80      0.79      0.80       484
           3       0.80      0.82      0.81       514
           4       0.88      0.87      0.88       543
           5       0.98      0.97      0.98       657
           6       1.00      1.00      1.00       804

    accuracy                           0.91      4152
   macro avg       0.90      0.90      0.90      4152
weighted avg       0.91      0.91      0.91      4152
```

Submission and conclusion

In [19]:
```
validation_ids = X_VAL['id']
X_validation = X_VAL.drop('id', axis=1)

X_test_scaled = scaler.fit_transform(X_validation)

y_test_preds = lgb_classifier.predict(X_test_scaled)
```

In [20]:
```
inverse_mapping = {v: k for k, v in target_encoding.items()}
y_pred = [inverse_mapping[label] for label in y_test_preds]
```

In [21]:
```
submission = pd.DataFrame({
    "id": validation_ids,
```

```
    "NObeyesdad": y_pred})
submission.to_csv("s4e02_0229_final.csv", index=False)
```

In the evaluation of our test dataset, the model achieved accuracy of 91%, which closely aligns with the 91.18% accuracy observed upon submission. This consistency in performance underscores the model's balanced capabilities and its proficiency in generalization across unseen data.

Such results affirmatively meet the project's targeted goals, demonstrating the model's effectiveness and reliability in making predictions.