In [54]:
```
import warnings
warnings.filterwarnings("ignore")
```

In [55]:
```
import pandas as pd

data = pd.read_csv(r'C:\Users\jayle\Downloads\estimation+of+obesity+levels+based+on+eating+habits+and+physical+condit

data.head()
```

Out[55]:

	Gender	Age	Height	Weight	family_history_with_overweight	FAVC	FCVC	NCP	CAEC	SMOKE	CH2O	SCC	FAF	TU
0	Female	21.0	1.62	64.0	yes	no	2.0	3.0	Sometimes	no	2.0	no	0.0	1.
1	Female	21.0	1.52	56.0	yes	no	3.0	3.0	Sometimes	yes	3.0	yes	3.0	0.
2	Male	23.0	1.80	77.0	yes	no	2.0	3.0	Sometimes	no	2.0	no	2.0	1.
3	Male	27.0	1.80	87.0	no	no	3.0	3.0	Sometimes	no	2.0	no	2.0	0.
4	Male	22.0	1.78	89.8	no	no	2.0	1.0	Sometimes	no	2.0	no	0.0	0.

In [56]:
```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import LabelEncoder

# Check for missing values and summary statistics
missing_values = data.isnull().sum()
summary_stats = data.describe(include='all')

# Convert numerical columns to numeric types
numerical_columns = ["Age", "Height", "CH2O", "FAF", "TUE", "NCP", "FCVC"]

for col in numerical_columns:
    data[col] = pd.to_numeric(data[col], errors='coerce')

missing_values, summary_stats
```

Out[56]: (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,

	Gender	Age	Height	Weight
count	2111	2111.000000	2111.000000	2111.000000
unique	2	NaN	NaN	NaN
top	Male	NaN	NaN	NaN
freq	1068	NaN	NaN	NaN
mean	NaN	24.312600	1.701677	86.586058
std	NaN	6.345968	0.093305	26.191172
min	NaN	14.000000	1.450000	39.000000
25%	NaN	19.947192	1.630000	65.473343
50%	NaN	22.777890	1.700499	83.000000
75%	NaN	26.000000	1.768464	107.430682
max	NaN	61.000000	1.980000	173.000000

	family_history_with_overweight	FAVC	FCVC	NCP
count	2111	2111	2111.000000	2111.000000
unique	2	2	NaN	NaN
top	yes	yes	NaN	NaN
freq	1726	1866	NaN	NaN
mean	NaN	NaN	2.419043	2.685628
std	NaN	NaN	0.533927	0.778039
min	NaN	NaN	1.000000	1.000000
25%	NaN	NaN	2.000000	2.658738
50%	NaN	NaN	2.385502	3.000000
75%	NaN	NaN	3.000000	3.000000

```
max                           NaN    NaN     3.000000     4.000000

             CAEC SMOKE         CH2O   SCC          FAF          TUE  \
count        2111  2111  2111.000000  2111  2111.000000  2111.000000
unique          4     2          NaN     2          NaN          NaN
top     Sometimes    no          NaN    no          NaN          NaN
freq         1765  2067          NaN  2015          NaN          NaN
mean          NaN   NaN     2.008011   NaN     1.010298     0.657866
std           NaN   NaN     0.612953   NaN     0.850592     0.608927
min           NaN   NaN     1.000000   NaN     0.000000     0.000000
25%           NaN   NaN     1.584812   NaN     0.124505     0.000000
50%           NaN   NaN     2.000000   NaN     1.000000     0.625350
75%           NaN   NaN     2.477420   NaN     1.666678     1.000000
max           NaN   NaN     3.000000   NaN     3.000000     2.000000

             CALC                MTRANS      NObeyesdad
count        2111                  2111            2111
unique          4                     5               7
top     Sometimes  Public_Transportation  Obesity_Type_I
freq         1401                  1580             351
mean          NaN                   NaN             NaN
std           NaN                   NaN             NaN
min           NaN                   NaN             NaN
25%           NaN                   NaN             NaN
50%           NaN                   NaN             NaN
75%           NaN                   NaN             NaN
max           NaN                   NaN             NaN  )
```

In [57]:

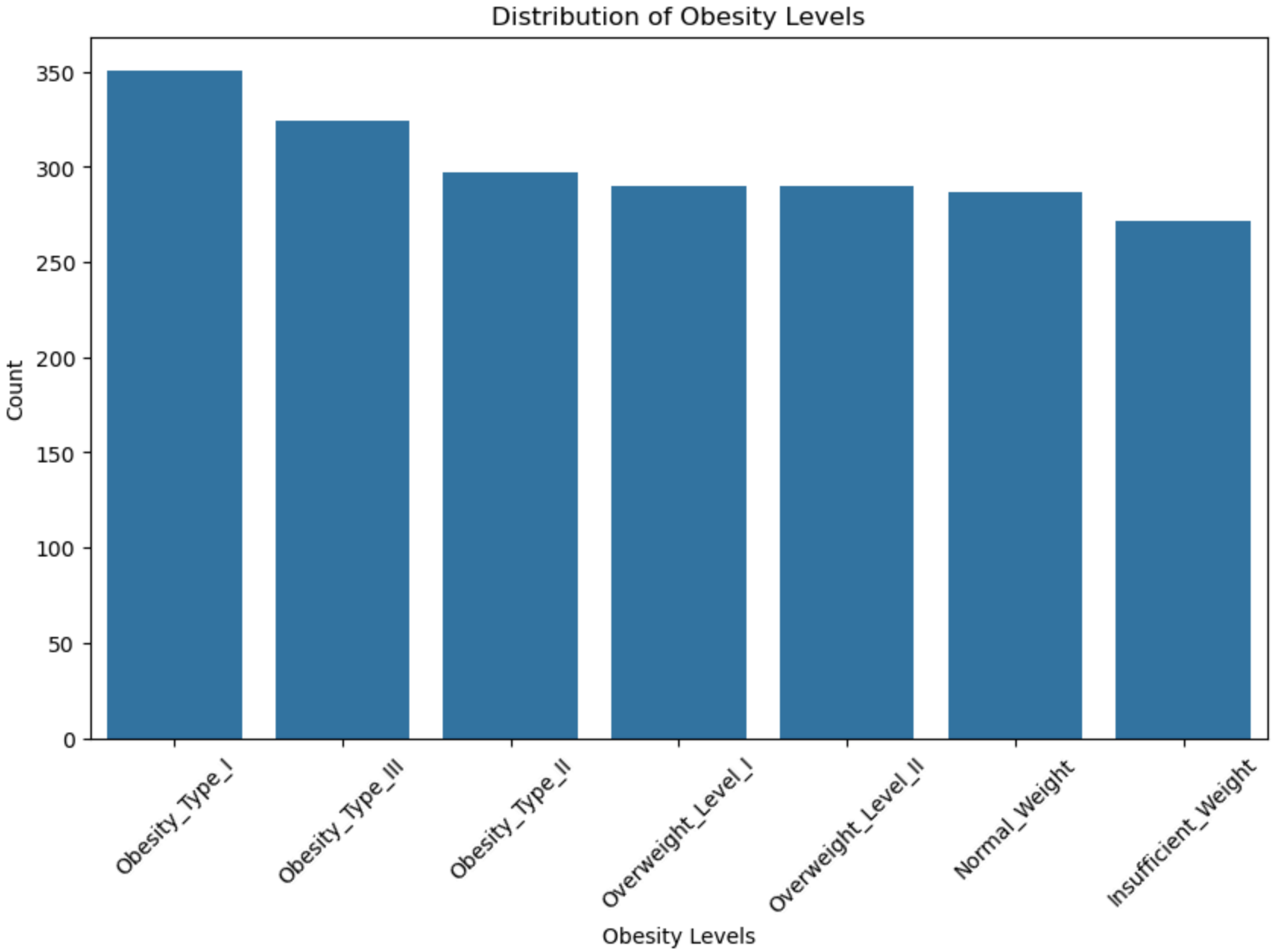
```
# Encode the target variable 'NObeyesdad'
label_encoder = LabelEncoder()
data['NObeyesdad_encoded'] = label_encoder.fit_transform(data['NObeyesdad'])
# Encode categorical features

categorical_columns = ["Gender", "family_history_with_overweight", "FAVC", "CAEC", "SMOKE", "SCC", "CALC", "MTRANS"]

for col in categorical_columns:
    data[col] = data[col].astype("category").cat.codes


# Visualize the distribution of obesity levels
plt.figure(figsize=(10, 6))
sns.countplot(data=data, x="NObeyesdad", order=data["NObeyesdad"].value_counts().index)
```

```
plt.title("Distribution of Obesity Levels")
plt.xlabel("Obesity Levels")
plt.ylabel("Count")
plt.xticks(rotation=45)
plt.show()
```



In [66]:
```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.tree import DecisionTreeRegressor
```

```
from sklearn.ensemble import RandomForestRegressor
from xgboost import XGBRegressor
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
import numpy as np

# Use 'Weight' as the target variable
X = data[numerical_columns + categorical_columns].drop(columns=['Weight'], errors='ignore')
y = data['Weight']

# Split dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3)

# Function to evaluate regression models
def evaluate_regression_model(model, X_test, y_test, predictions):
    mape = np.mean(np.abs((y_test - predictions) / y_test)) * 100  # Calculate MAPE
    print(f"{model.__class__.__name__} Results:")
    print("Mean Absolute Error (MAE):", mean_absolute_error(y_test, predictions))
    print("Mean Squared Error (MSE):", mean_squared_error(y_test, predictions))
    print("R-Squared (R²):", r2_score(y_test, predictions))
    print("Mean Absolute Percentage Error (MAPE):", f"{mape:.2f}%")
    print("-" * 50)

# Linear Regression
linear_model = LinearRegression()
linear_model.fit(X_train, y_train)
linear_predictions = linear_model.predict(X_test)
evaluate_regression_model(linear_model, X_test, y_test, linear_predictions)

# Decision Tree Regressor
tree_model = DecisionTreeRegressor(random_state=42)
tree_model.fit(X_train, y_train)
tree_predictions = tree_model.predict(X_test)
evaluate_regression_model(tree_model, X_test, y_test, tree_predictions)

# Random Forest Regressor
rf_model = RandomForestRegressor(n_estimators=100, random_state=42)
rf_model.fit(X_train, y_train)
rf_predictions = rf_model.predict(X_test)
evaluate_regression_model(rf_model, X_test, y_test, rf_predictions)

# XGBoost Regressor
xgb_model = XGBRegressor(n_estimators=100, random_state=42)
```

```
xgb_model.fit(X_train, y_train)
xgb_predictions = xgb_model.predict(X_test)
evaluate_regression_model(xgb_model, X_test, y_test, xgb_predictions)
```

```
LinearRegression Results:
Mean Absolute Error (MAE): 14.176576691267291
Mean Squared Error (MSE): 312.661085381815
R-Squared (R²): 0.5628600314134603
Mean Absolute Percentage Error (MAPE): 18.28%
--------------------------------------------------
DecisionTreeRegressor Results:
Mean Absolute Error (MAE): 7.043174927444794
Mean Squared Error (MSE): 171.8108623195123
R-Squared (R²): 0.7597865597330067
Mean Absolute Percentage Error (MAPE): 9.43%
--------------------------------------------------
RandomForestRegressor Results:
Mean Absolute Error (MAE): 5.519428959290221
Mean Squared Error (MSE): 89.34673889231884
R-Squared (R²): 0.8750818939139726
Mean Absolute Percentage Error (MAPE): 7.43%
--------------------------------------------------
XGBRegressor Results:
Mean Absolute Error (MAE): 6.306181148155622
Mean Squared Error (MSE): 103.8129982304516
R-Squared (R²): 0.8548562232171745
Mean Absolute Percentage Error (MAPE): 8.45%
--------------------------------------------------
```

In [68]:

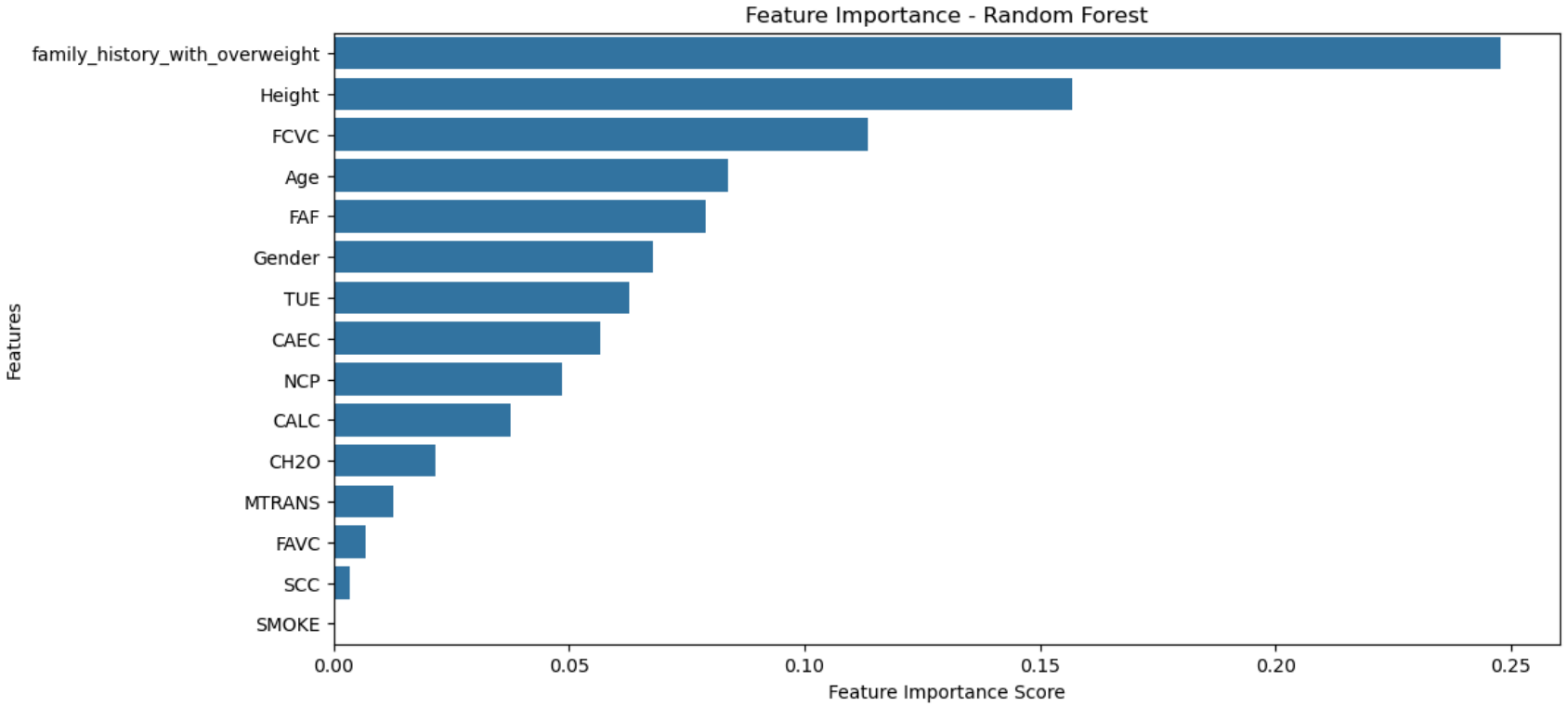
```
import numpy as np

# Get feature importances from Random Forest
feature_importances = rf_model.feature_importances_

# Sort features by importance
sorted_indices = np.argsort(feature_importances)[::-1]
sorted_features = np.array(X.columns)[sorted_indices]

# Plot feature importance
plt.figure(figsize=(12, 6))
sns.barplot(x=feature_importances[sorted_indices], y=sorted_features)
plt.xlabel("Feature Importance Score")
```

```
plt.ylabel("Features")
plt.title("Feature Importance - Random Forest")
plt.show()
```



In [70]:
```
import pandas as pd
import numpy as np

# Get feature names
feature_names = X_train.columns

# Get coefficients from trained Linear Regression model
coefficients = linear_model.coef_

# Create a DataFrame for better visualization
coefficients_df = pd.DataFrame({"Feature": feature_names, "Coefficient": coefficients})

# Sort by absolute value of the coefficient to see the most impactful variables
coefficients_df["Absolute Impact"] = coefficients_df["Coefficient"].abs()
```

```
coefficients_df = coefficients_df.sort_values(by="Absolute Impact", ascending=False)

coefficients_df
```

Out[70]:

	Feature	Coefficient	Absolute Impact
1	Height	126.976767	126.976767
8	family_history_with_overweight	19.913219	19.913219
10	CAEC	11.337589	11.337589
6	FCVC	9.487723	9.487723
12	SCC	-6.227814	6.227814
9	FAVC	5.629211	5.629211
7	Gender	-4.931248	4.931248
13	CALC	-4.823068	4.823068
14	MTRANS	3.701295	3.701295
3	FAF	-3.237818	3.237818
11	SMOKE	-1.888175	1.888175
4	TUE	-1.817945	1.817945
2	CH2O	1.341186	1.341186
0	Age	0.828074	0.828074
5	NCP	0.603985	0.603985

In [ ]: