In [21]:

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, OneHotEncoder

######## Step 1: Load the dataset

file_path = r"C:\Project@GAVATAR\Energy_consumption.csv"
data = pd.read_csv(file_path)
print("Data Loaded:")
```

```
Data Loaded:
```

In [5]:

```
######## Step 2: Data Preprocessing


# Convert categorical columns to numeric using one-hot encoding
categorical_cols = ['DayOfWeek', 'Holiday', 'HVACUsage', 'LightingUsage']
## encoder = OneHotEncoder(drop='first', sparse_output=False)


encoder = OneHotEncoder(drop='first', sparse=False)

encoded_cats = encoder.fit_transform(data[categorical_cols])

# Create a DataFrame from the encoded columns
encoded_df = pd.DataFrame(encoded_cats, columns=encoder.get_feature_names_out(categ

# Drop the original categorical columns and concatenate the encoded ones
data = data.drop(categorical_cols + ['Timestamp'], axis=1)
data = pd.concat([data, encoded_df], axis=1)

# Check for any non-numeric columns
print("Data types after preprocessing:")
print(data.dtypes)

# Separate features and target
X = data.drop('EnergyConsumption', axis=1)
y = data['EnergyConsumption']

# Feature scaling
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

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

# Output to check
print(X_train.shape, X_test.shape, y_train.shape, y_test.shape)

import matplotlib.pyplot as plt
import seaborn as sns
```

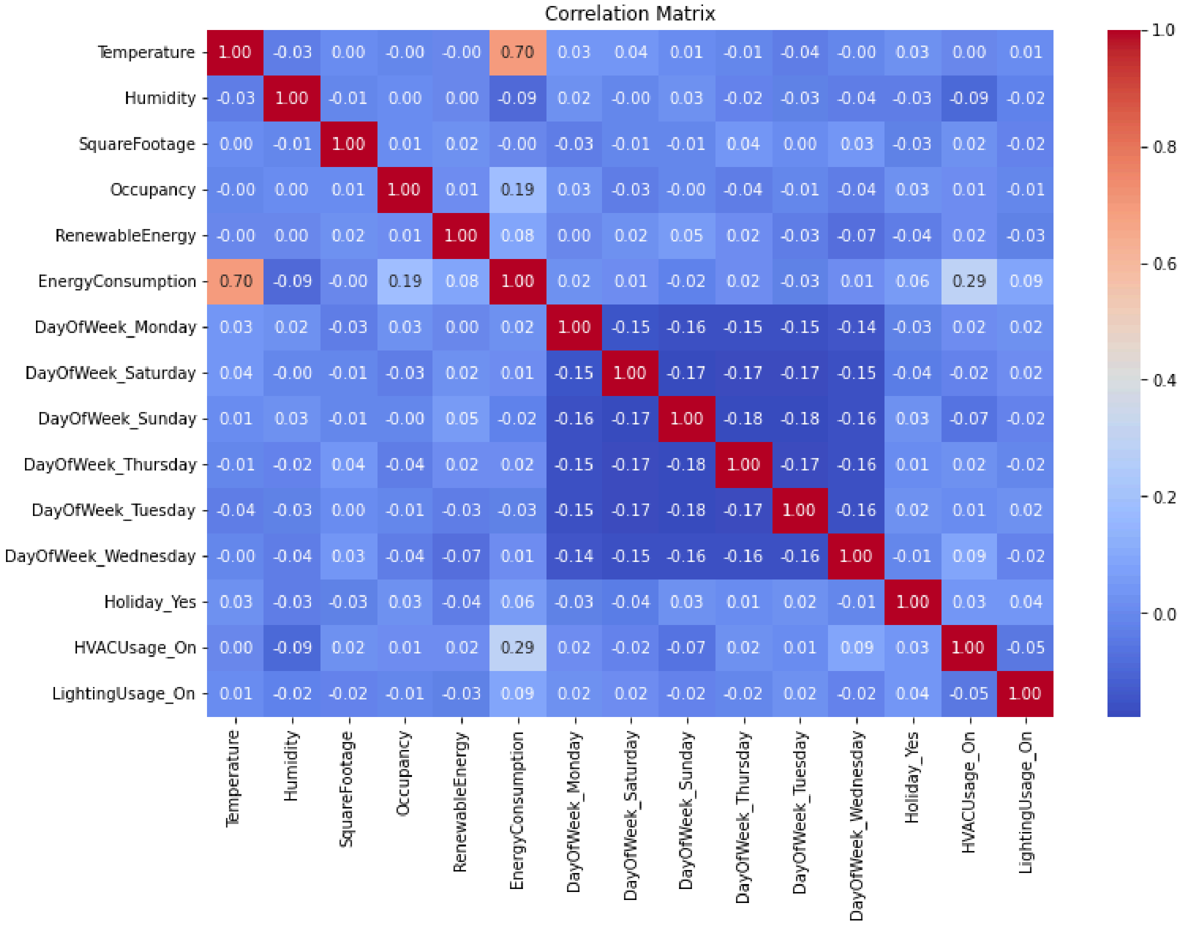
```
Data types after preprocessing:
Temperature              float64
Humidity                 float64
SquareFootage            float64
Occupancy                  int64
RenewableEnergy          float64
EnergyConsumption        float64
DayOfWeek_Monday         float64
DayOfWeek_Saturday       float64
DayOfWeek_Sunday         float64
DayOfWeek_Thursday       float64
DayOfWeek_Tuesday        float64
DayOfWeek_Wednesday      float64
Holiday_Yes              float64
HVACUsage_On             float64
LightingUsage_On         float64
dtype: object
(800, 14) (200, 14) (800,) (200,)
```

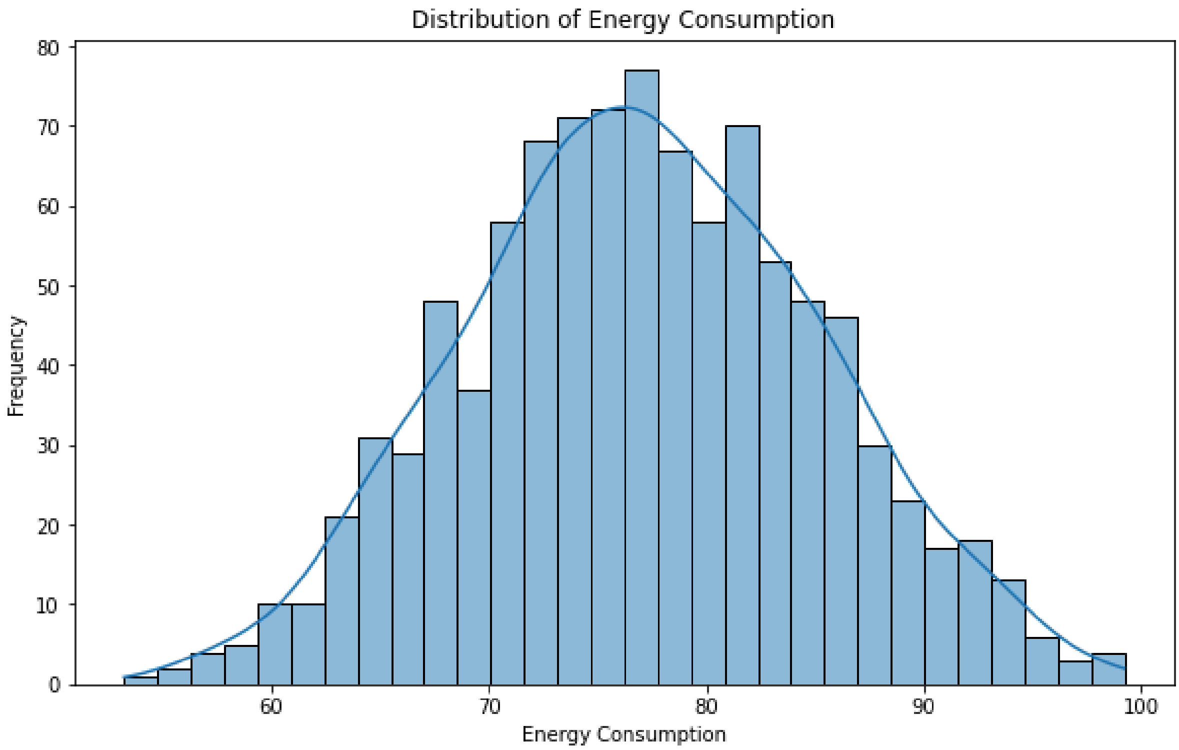
In [9]:

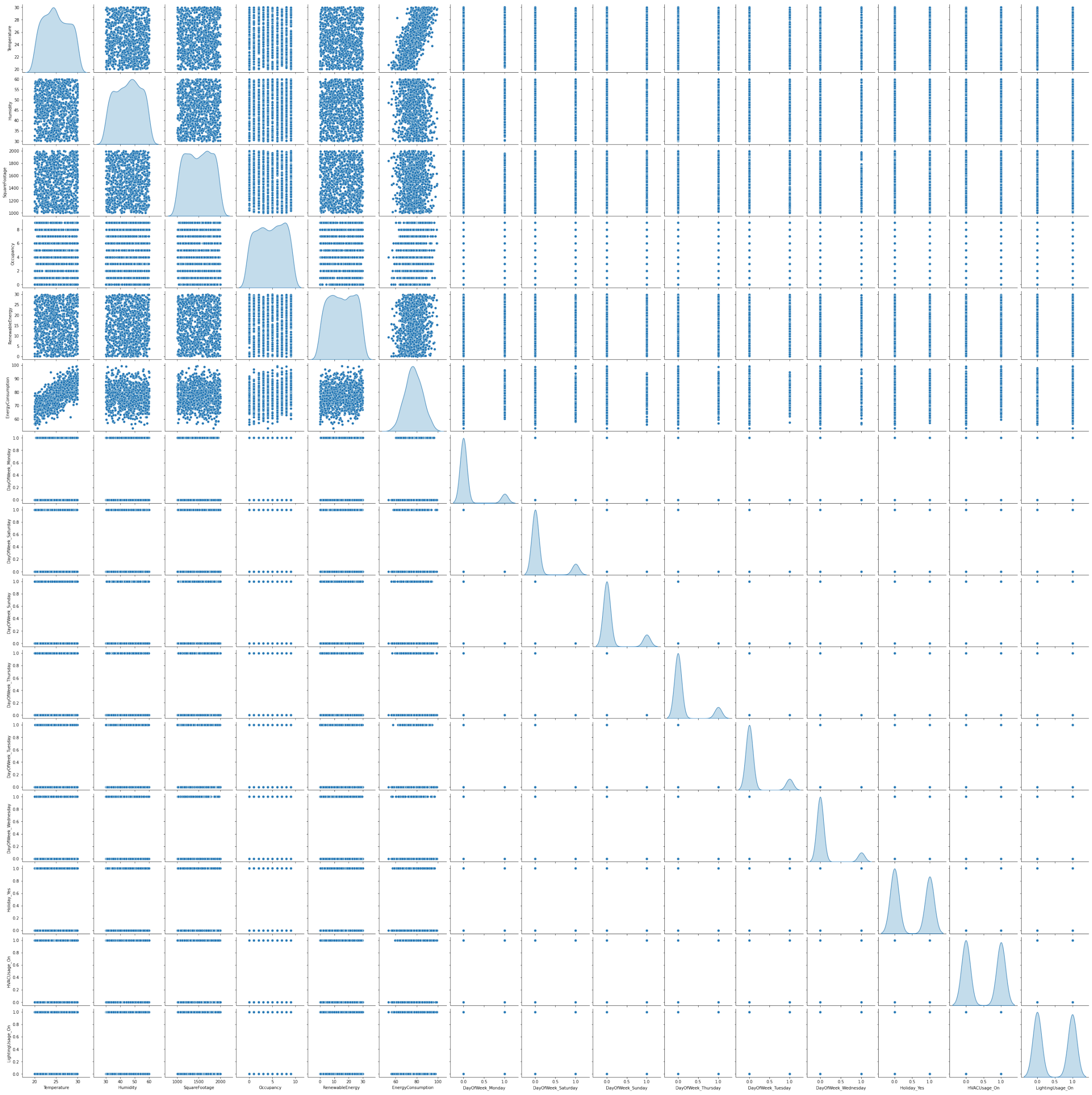
```python
# Step 3: Data Analysis

!pip install xgboost

# 1. Statistical Summary of the Dataset
print("Statistical Summary of the Dataset:")
print(data.describe())

# 2. Correlation Matrix
plt.figure(figsize=(12, 8))
sns.heatmap(data.corr(), annot=True, cmap='coolwarm', fmt='.2f')
plt.title("Correlation Matrix")
plt.show()

# 3. Distribution of Target Variable (Energy Consumption)
plt.figure(figsize=(10, 6))
sns.histplot(y, bins=30, kde=True)
plt.title("Distribution of Energy Consumption")
plt.xlabel("Energy Consumption")
plt.ylabel("Frequency")
plt.show()

# 4. Pairplot of Features
# This can help visualize the relationships between features
sns.pairplot(data, diag_kind='kde')
plt.show()

# 5. Boxplot of Categorical Features vs. Energy Consumption
for col in categorical_cols:
    if col in data.columns:
        plt.figure(figsize=(10, 6))
        sns.boxplot(x=col, y='EnergyConsumption', data=pd.concat([data[col], y], ax
        plt.title(f"Boxplot of {col} vs Energy Consumption")
        plt.show()
    else:
        print(f"Column {col} not found in the data.")


# 6. Check for Missing Values
print("Missing Values in the Dataset:")
print(data.isnull().sum())

# 7. Feature Importance (Optional but recommended before model creation)
# This can be done using an initial model, such as a Random Forest
from sklearn.ensemble import RandomForestRegressor
```

```python
model = RandomForestRegressor(random_state=42)
model.fit(X_train, y_train)

# Get feature importances
importances = model.feature_importances_
feature_names = X.columns

# Create a DataFrame for visualization
importance_df = pd.DataFrame({'Feature': feature_names, 'Importance': importances})
importance_df = importance_df.sort_values(by='Importance', ascending=False)

# Plot the feature importances
plt.figure(figsize=(12, 8))
sns.barplot(x='Importance', y='Feature', data=importance_df)
plt.title("Feature Importance")
plt.show()

import xgboost as xgb
from sklearn.metrics import mean_squared_error, r2_score
```

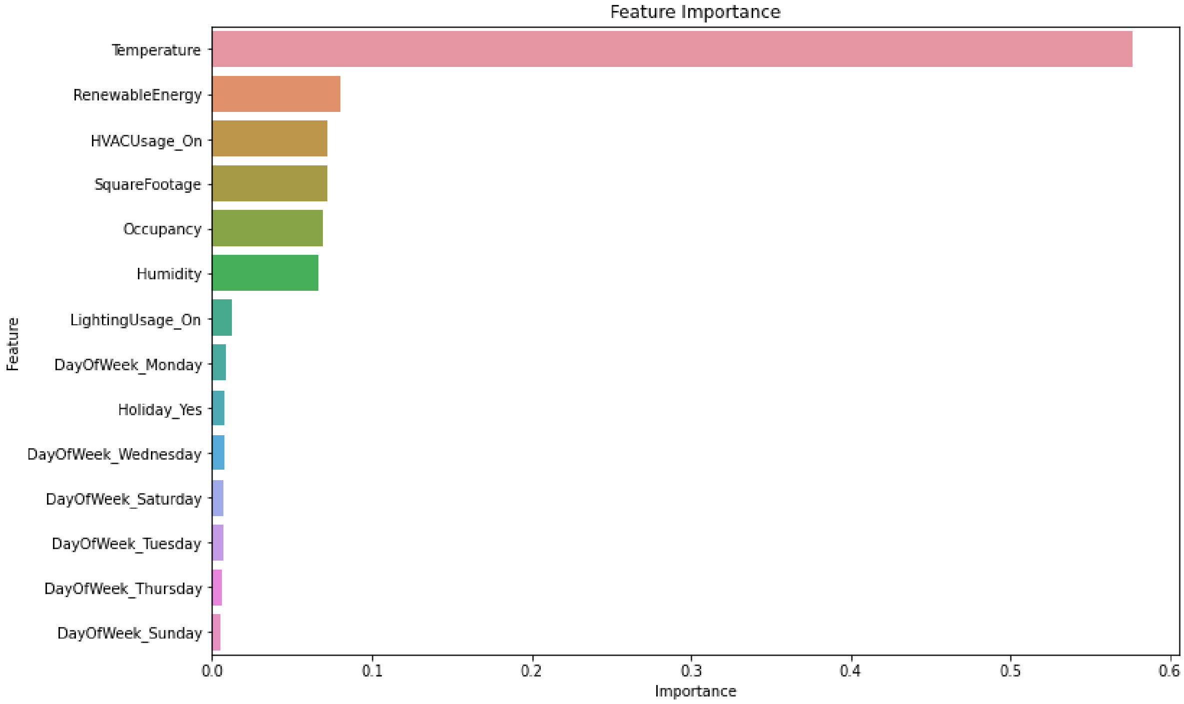
```
Requirement already satisfied: xgboost in c:\users\aditya\anaconda3\lib\site-packa
ges (2.1.1)
Requirement already satisfied: numpy in c:\users\aditya\anaconda3\lib\site-package
s (from xgboost) (1.21.5)
Requirement already satisfied: scipy in c:\users\aditya\anaconda3\lib\site-package
s (from xgboost) (1.7.3)
Statistical Summary of the Dataset:
       Temperature     Humidity  SquareFootage    Occupancy  RenewableEnergy  \
count  1000.000000  1000.000000    1000.000000  1000.000000      1000.000000   
mean     24.982026    45.395412    1500.052488     4.581000        15.132813   
std       2.836850     8.518905     288.418873     2.865598         8.745917   
min      20.007565    30.015975    1000.512661     0.000000         0.006642   
25%      22.645070    38.297722    1247.108548     2.000000         7.628385   
50%      24.751637    45.972116    1507.967426     5.000000        15.072296   
75%      27.418174    52.420066    1740.340165     7.000000        22.884064   
max      29.998671    59.969085    1999.982252     9.000000        29.965327   

       EnergyConsumption  DayOfWeek_Monday  DayOfWeek_Saturday  \
count        1000.000000       1000.000000         1000.000000   
mean           77.055873          0.123000            0.143000   
std             8.144112          0.328602            0.350248   
min            53.263278          0.000000            0.000000   
25%            71.544690          0.000000            0.000000   
50%            76.943696          0.000000            0.000000   
75%            82.921742          0.000000            0.000000   
max            99.201120          1.000000            1.000000   

       DayOfWeek_Sunday  DayOfWeek_Thursday  DayOfWeek_Tuesday  \
count       1000.000000         1000.000000        1000.000000   
mean           0.154000            0.146000           0.146000   
std            0.361129            0.353283           0.353283   
min            0.000000            0.000000           0.000000   
25%            0.000000            0.000000           0.000000   
50%            0.000000            0.000000           0.000000   
75%            0.000000            0.000000           0.000000   
max            1.000000            1.000000           1.000000   

       DayOfWeek_Wednesday  Holiday_Yes  HVACUsage_On  LightingUsage_On  
count          1000.000000  1000.000000   1000.000000       1000.000000  
mean              0.124000     0.467000      0.492000          0.491000  
std               0.329746     0.499159      0.500186          0.500169  
min               0.000000     0.000000      0.000000          0.000000  
25%               0.000000     0.000000      0.000000          0.000000  
50%               0.000000     0.000000      0.000000          0.000000  
75%               0.000000     1.000000      1.000000          1.000000  
max               1.000000     1.000000      1.000000          1.000000  
```


Correlation Matrix
Temperature 1.00 -0.03 0.00 -0.00 -0.00 0.70 0.03 0.04 0.01 -0.01 -0.04 -0.00 0.03 0.00 0.01
Humidity -0.03 1.00 -0.01 0.00 0.00 -0.09 0.02 -0.00 0.03 -0.02 -0.03 -0.04 -0.03 -0.09 -0.02
SquareFootage 0.00 -0.01 1.00 0.01 0.02 -0.00 -0.03 -0.01 -0.01 0.04 0.00 0.03 -0.03 0.02 -0.02
Occupancy -0.00 0.00 0.01 1.00 0.01 0.19 0.03 -0.03 -0.00 -0.04 -0.01 -0.04 0.03 0.01 -0.01
RenewableEnergy -0.00 0.00 0.02 0.01 1.00 0.08 0.00 0.02 0.05 0.02 -0.03 -0.07 -0.04 0.02 -0.03
EnergyConsumption 0.70 -0.09 -0.00 0.19 0.08 1.00 0.02 0.01 -0.02 0.02 -0.03 0.01 0.06 0.29 0.09
DayOfWeek_Monday 0.03 0.02 -0.03 0.03 0.00 0.02 1.00 -0.15 -0.16 -0.15 -0.15 -0.14 -0.03 0.02 0.02
DayOfWeek_Saturday 0.04 -0.00 -0.01 -0.03 0.02 0.01 -0.15 1.00 -0.17 -0.17 -0.17 -0.15 -0.04 -0.02 0.02
DayOfWeek_Sunday 0.01 0.03 -0.01 -0.00 0.05 -0.02 -0.16 -0.17 1.00 -0.18 -0.18 -0.16 0.03 -0.07 -0.02
DayOfWeek_Thursday -0.01 -0.02 0.04 -0.04 0.02 0.02 -0.15 -0.17 -0.18 1.00 -0.17 -0.16 0.01 0.02 -0.02
DayOfWeek_Tuesday -0.04 -0.03 0.00 -0.01 -0.03 -0.03 -0.15 -0.17 -0.18 -0.17 1.00 -0.16 0.02 0.01 0.02
DayOfWeek_Wednesday -0.00 -0.04 0.03 -0.04 -0.07 0.01 -0.14 -0.15 -0.16 -0.16 -0.16 1.00 -0.01 0.09 -0.02
Holiday_Yes 0.03 -0.03 -0.03 0.03 -0.04 0.06 -0.03 -0.04 0.03 0.01 0.02 -0.01 1.00 0.03 0.04
HVACUsage_On 0.00 -0.09 0.02 0.01 0.02 0.29 0.02 -0.02 -0.07 0.02 0.01 0.09 0.03 1.00 -0.05
LightingUsage_On 0.01 -0.02 -0.02 -0.01 -0.03 0.09 0.02 0.02 -0.02 -0.02 0.02 -0.02 0.04 -0.05 1.00
Temperature
Humidity
SquareFootage
Occupancy
RenewableEnergy
EnergyConsumption
DayOfWeek_Monday
DayOfWeek_Saturday
DayOfWeek_Sunday
DayOfWeek_Thursday
DayOfWeek_Tuesday
DayOfWeek_Wednesday
Holiday_Yes
HVACUsage_On
LightingUsage_On
1.0
0.8
0.6
0.4
0.2
0.0


Distribution of Energy Consumption
Frequency
0
10
20
30
40
50
60
70
80
60
70
80
90
100
Energy Consumption



```
Column DayOfWeek not found in the data.
Column Holiday not found in the data.
Column HVACUsage not found in the data.
Column LightingUsage not found in the data.
Missing Values in the Dataset:
Temperature            0
Humidity               0
SquareFootage          0
Occupancy              0
RenewableEnergy        0
EnergyConsumption      0
DayOfWeek_Monday       0
DayOfWeek_Saturday     0
DayOfWeek_Sunday       0
DayOfWeek_Thursday     0
DayOfWeek_Tuesday      0
DayOfWeek_Wednesday    0
Holiday_Yes            0
HVACUsage_On           0
LightingUsage_On       0
dtype: int64
```

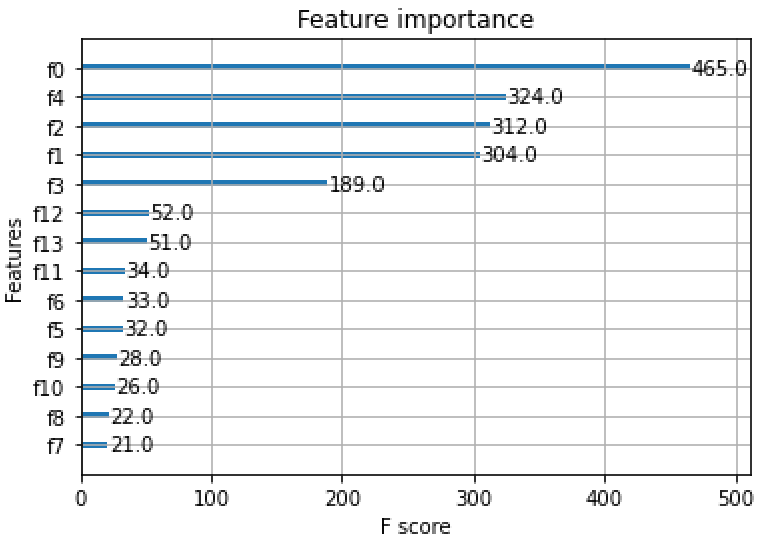


In [10]:

```
# Step 4: Model Creation


dtrain = xgb.DMatrix(X_train, label=y_train)
dtest = xgb.DMatrix(X_test, label=y_test)

params = {
    'objective': 'reg:squarederror',
    'max_depth': 6,
    'eta': 0.1,
    'subsample': 0.8,
    'colsample_bytree': 0.8,
    'eval_metric': 'rmse'
}

model = xgb.train(params, dtrain, num_boost_round=100, evals=[(dtest, "Test")], ear

y_pred = model.predict(dtest)

rmse = mean_squared_error(y_test, y_pred, squared=False)
r2 = r2_score(y_test, y_pred)

print(f"Root Mean Squared Error (RMSE): {rmse}")
print(f"R-squared (R2): {r2}")

# Feature Importance (optional)
xgb.plot_importance(model)
plt.show()

import xgboost as xgb
from sklearn.metrics import mean_squared_error

# Step 4: Model Creation
dtrain = xgb.DMatrix(X_train, label=y_train)
dtest = xgb.DMatrix(X_test, label=y_test)

# Define XGBoost parameters
params = {
    'max_depth': 6,
    'eta': 0.1,
    'objective': 'reg:squarederror'
```

```python
}

# Train the model
model = xgb.train(params, dtrain, num_boost_round=100)

# Predict on the test set
y_pred = model.predict(dtest)

# Evaluate the model performance
rmse = mean_squared_error(y_test, y_pred, squared=False)
print(f"Step 4: Model Creation - Root Mean Squared Error (RMSE): {rmse}")

from sklearn.model_selection import GridSearchCV
from sklearn.metrics import mean_absolute_error, r2_score
```

```
[0]	Test-rmse:7.76426
[1]	Test-rmse:7.76873
[2]	Test-rmse:7.48527
[3]	Test-rmse:7.13718
[4]	Test-rmse:6.92934
[5]	Test-rmse:6.67520
[6]	Test-rmse:6.47182
[7]	Test-rmse:6.30439
[8]	Test-rmse:6.16041
[9]	Test-rmse:6.01678
[10]	Test-rmse:5.92096
[11]	Test-rmse:5.82810
[12]	Test-rmse:5.74675
[13]	Test-rmse:5.71332
[14]	Test-rmse:5.66514
[15]	Test-rmse:5.64722
[16]	Test-rmse:5.61516
[17]	Test-rmse:5.57527
[18]	Test-rmse:5.55373
[19]	Test-rmse:5.52590
[20]	Test-rmse:5.53440
[21]	Test-rmse:5.54101
[22]	Test-rmse:5.55179
[23]	Test-rmse:5.53525
[24]	Test-rmse:5.53118
[25]	Test-rmse:5.51204
[26]	Test-rmse:5.50587
[27]	Test-rmse:5.49723
[28]	Test-rmse:5.50805
[29]	Test-rmse:5.50302
[30]	Test-rmse:5.51594
[31]	Test-rmse:5.51786
[32]	Test-rmse:5.50548
[33]	Test-rmse:5.50733
[34]	Test-rmse:5.50591
[35]	Test-rmse:5.51631
[36]	Test-rmse:5.51091
[37]	Test-rmse:5.50634
Root Mean Squared Error (RMSE): 5.50634416260444
R-squared (R2): 0.5371010129636163
```




```
Step 4: Model Creation - Root Mean Squared Error (RMSE): 5.639691046364907
```

In [11]:
```python
# Step 5: Model Validation

# Evaluate the model performance on the test set
mae = mean_absolute_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print(f"Step 5: Model Validation - Mean Absolute Error (MAE): {mae}")
print(f"Step 5: Model Validation - R-squared (R2): {r2}")

# Hyperparameter Tuning using GridSearchCV
param_grid = {
    'max_depth': [3, 6, 9],
    'eta': [0.01, 0.1, 0.3],
    'subsample': [0.6, 0.8, 1.0],
    'colsample_bytree': [0.6, 0.8, 1.0],
    'n_estimators': [50, 100, 200]
}

xgb_reg = xgb.XGBRegressor(objective='reg:squarederror')
grid_search = GridSearchCV(estimator=xgb_reg, param_grid=param_grid, cv=3, scoring=

grid_search.fit(X_train, y_train)
print("Best Hyperparameters found by GridSearchCV:", grid_search.best_params_)

# Train the model with the best hyperparameters
best_model = grid_search.best_estimator_

# Predict on the test set with the best model
y_best_pred = best_model.predict(X_test)

# Re-evaluate the model with the best hyperparameters
best_rmse = mean_squared_error(y_test, y_best_pred, squared=False)
best_mae = mean_absolute_error(y_test, y_best_pred)
best_r2 = r2_score(y_test, y_best_pred)

print(f"Best Model - Root Mean Squared Error (RMSE): {best_rmse}")
print(f"Best Model - Mean Absolute Error (MAE): {best_mae}")
print(f"Best Model - R-squared (R2): {best_r2}")
```

```
Step 5: Model Validation - Mean Absolute Error (MAE): 4.537023204324606
Step 5: Model Validation - R-squared (R2): 0.5144095337933573
Fitting 3 folds for each of 243 candidates, totalling 729 fits
Best Hyperparameters found by GridSearchCV: {'colsample_bytree': 1.0, 'eta': 0.1,
'max_depth': 3, 'n_estimators': 50, 'subsample': 0.6}
Best Model - Root Mean Squared Error (RMSE): 5.4743394151057165
Best Model - Mean Absolute Error (MAE): 4.406607500941804
Best Model - R-squared (R2): 0.5424664278195561
```

In [12]:

```python
# Step 6: Prediction

# Output the first few predictions and corresponding true values
print("Step 6: Prediction")
for i in range(10):
    print(f"Predicted: {y_best_pred[i]}, Actual: {y_test.iloc[i]}")
```

```
Step 6: Prediction
Predicted: 83.96011352539062, Actual: 86.92061127676786
Predicted: 80.1465835571289, Actual: 88.35160574829843
Predicted: 75.21458435058594, Actual: 79.43136341116085
Predicted: 88.04579162597656, Actual: 90.00918791745602
Predicted: 75.00990295410156, Actual: 83.89109970828049
Predicted: 84.37055969238281, Actual: 87.54904071367156
Predicted: 76.75141906738281, Actual: 79.69723688154224
Predicted: 72.79784393310547, Actual: 80.91405726325533
Predicted: 76.39122009277344, Actual: 85.13385609164183
Predicted: 67.87235260009766, Actual: 71.01713960978869
```

In [ ]: