Sure! Let's implement a simple example of Gradient Boosting for a regression problem using the popular XGBoost library in Python. First, you'll need to install the XGBoost library if you haven't already:

pip install xgboost

Now, let's create a simple example using a synthetic dataset. In this example, we'll use the Boston housing dataset, which is included in the scikit-learn library. We'll train a Gradient Boosting model to predict the median value of owner-occupied homes in Boston.

import xgboost as xgb

from sklearn.datasets import load\_boston

from sklearn.model\_selection import train\_test\_split

from sklearn.metrics import mean\_squared\_error

# Load the Boston housing dataset

boston = load\_boston()

X, y = boston.data, boston.target

# Split the dataset into training and testing sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

# Create an XGBoost regression model

model = xgb.XGBRegressor(objective ='reg:squarederror', colsample\_bytree = 0.3, learning\_rate = 0.1,

max\_depth = 5, alpha = 10, n\_estimators = 10)

# Train the model

model.fit(X\_train, y\_train)

# Make predictions

y\_pred = model.predict(X\_test)

# Evaluate the model

mse = mean\_squared\_error(y\_test, y\_pred)

print("Mean Squared Error:", mse)

In this example, we first load the Boston housing dataset and split it into training and testing sets. We then create an XGBoost regression model using xgb.XGBRegressor and train it on the training data using the fit method. After training, we make predictions on the test set and evaluate the model's performance using mean squared error.

This is a basic example to get you started with Gradient Boosting in Python using XGBoost. You can further explore tuning hyperparameters, cross-validation, and feature engineering to improve the model's performance.