```
import matplotlib.pyplot as plt
import numpy as np

from sklearn.tree import DecisionTreeRegressor
from sklearn import model_selection

from sklearn.datasets import load_diabetes

X, y = load_diabetes(return_X_y=True)
X_train, X_test, y_train, y_test = model_selection.train_test_split(X, y, test_size=0.
25)



def gb_predict(X, trees_list, coef_list, eta):
    # Реализуемый алгоритм градиентного бустинга будет инициализироваться нулевыми
значениями,
    # поэтому все деревья из списка trees_list уже являются дополнительными и при
предсказании прибавляются с шагом eta
    return np.array([sum([eta * coef * alg.predict([x])[0] for alg, coef in zip(
trees_list, coef_list)]) for x in X])



def mean_squared_error(y_real, prediction):
    return (sum((y_real - prediction) ** 2)) / len(y_real)



def bias(y, z):
    return (y - z)
```

```
def gb_fit(n_trees, max_depth, X_train, X_test, y_train, y_test, coefs, eta):
    # Деревья будем записывать в список
    trees = []

    # Будем записывать ошибки на обучающей и тестовой выборке на каждой итерации в
список
    train_errors = []
    test_errors = []

    for i in range(n_trees):
        tree = DecisionTreeRegressor(max_depth=max_depth, random_state=42)

        # инициализируем бустинг начальным алгоритмом, возвращающим ноль,
        # поэтому первый алгоритм просто обучаем на выборке и добавляем в список
        if len(trees) == 0:
            # обучаем первое дерево на обучающей выборке
            tree.fit(X_train, y_train)

            train_errors.append(mean_squared_error(y_train, gb_predict(X_train, trees,
coefs, eta)))
            test_errors.append(mean_squared_error(y_test, gb_predict(X_test, trees,
coefs, eta)))
        else:
            # Получим ответы на текущей композиции
            target = gb_predict(X_train, trees, coefs, eta)

            # алгоритмы начиная со второго обучаем на сдвиг
            tree.fit(X_train, bias(y_train, target))
```

```
            train_errors.append(mean_squared_error(y_train, gb_predict(X_train, trees,
coefs, eta)))
            test_errors.append(mean_squared_error(y_test, gb_predict(X_test, trees,
coefs, eta)))

        trees.append(tree)

    return trees, train_errors, test_errors


def evaluate_alg(X_train, X_test, y_train, y_test, trees, coefs, eta):
    # train_prediction = gb_predict(X_train, trees, coefs, eta)
    #
    # print(f'Ошибка алгоритма из {n_trees} деревьев глубиной {max_depth} \
    # с шагом {eta} на тренировочной выборке: {mean_squared_error(y_train,
train_prediction)}')
    #
    test_prediction = gb_predict(X_test, trees, coefs, eta)

    print(f'Ошибка алгоритма из {n_trees} деревьев глубиной {max_depth} \
    с шагом {eta} на тестовой выборке: {mean_squared_error(y_test, test_prediction)}')


def get_error_plot(n_trees, train_err, test_err):
    plt.xlabel('Iteration number')
    plt.ylabel('MSE')
    plt.xlim(0, n_trees)
    plt.plot(list(range(n_trees)), train_err, label='train error')
```

```
    plt.plot(list(range(n_trees)), test_err, label='test error')
    plt.legend(loc='upper right')
    plt.show()


# Число деревьев в ансамбле
n_trees = 10
# для простоты примем коэффициенты равными 1
coefs = [1] * n_trees
# Максимальная глубина деревьев
max_depth = 3
# Шаг
eta = 1

# 1) Для реализованной модели градиентного бустинга построить графики зависимости
ошибки
# от количества деревьев в ансамбле и от максимальной глубины деревьев.
# Сделать выводы о зависимости ошибки от этих параметров.


max_n_trees = 14
np_data = np.zeros((max_n_trees,max_depth))

for depth in range(max_depth):
    n_trees = 1
    while n_trees <= max_n_trees:
        coefs = [1] * n_trees
        trees, train_errors, test_errors = gb_fit(n_trees, depth+1, X_train, X_test,
y_train, y_test, coefs, eta)
```

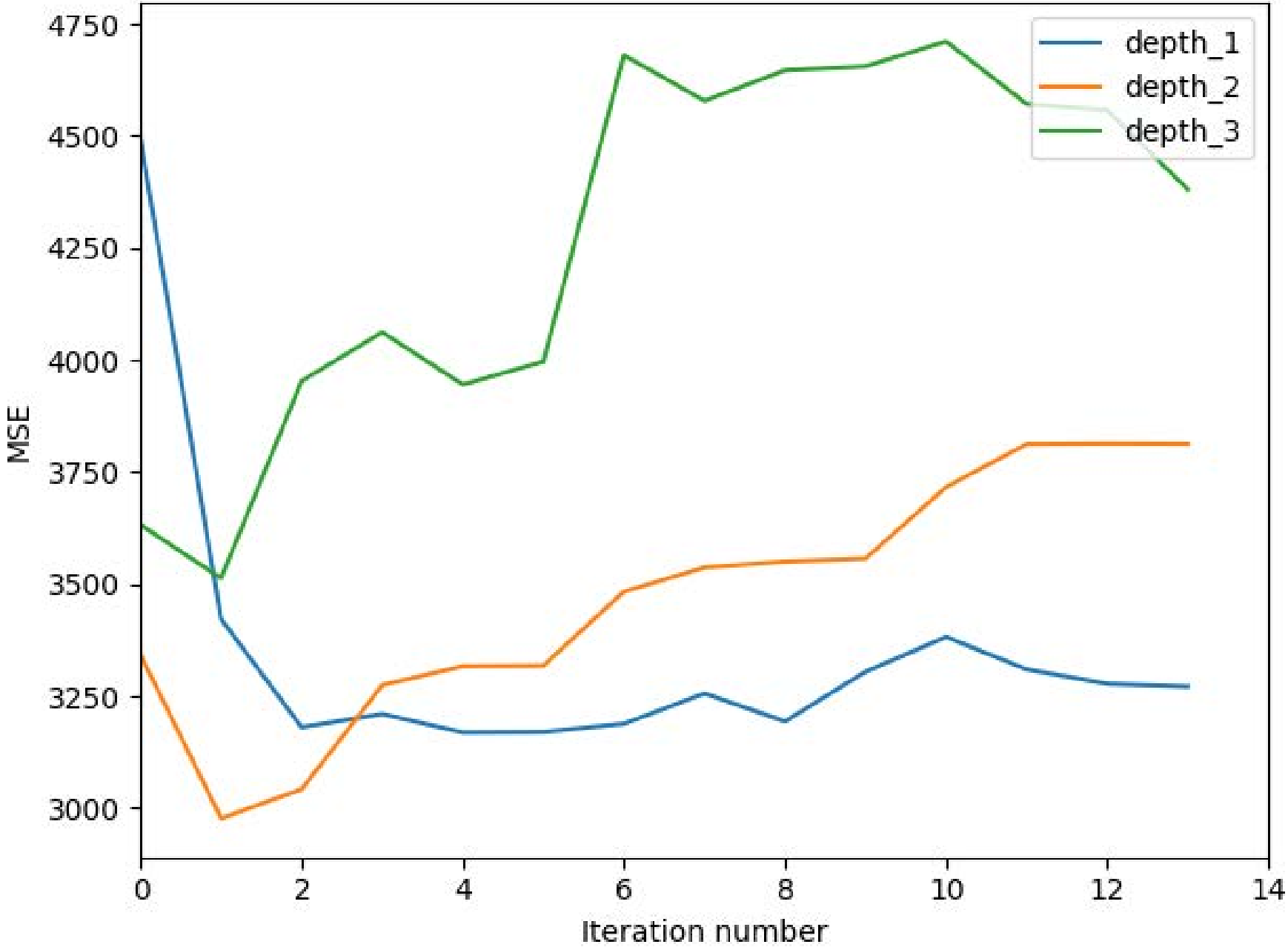
```
        np_data[n_trees-1,depth] = mean_squared_error(y_test, gb_predict(X_test, trees
, coefs, eta))

        n_trees+=1
    print(f'DEPTH: {depth+1}')

plt.xlabel('Iteration number')
plt.ylabel('MSE')
plt.xlim(0, max_n_trees)
for depth in range(max_depth):
    plt.plot(list(range(max_n_trees)), np_data[:,depth], label=f'depth_{depth+1}')

plt.legend(loc='upper right')
plt.show()


# Ответ(философский вывод): Судя по графику ошибки MSE, от 2х до 4х дереревьев
достаточно для получения минимальной ошибки
# Если рассматривать глубину деревьев, то для данного алгоритма пеньки лучьше чем
ветвистые деревья. Это легко проследить по синему графику depth_1
```


depth_1
depth_2
depth_3
MSE
Iteration number
4750
4500
4250
4000
3750
3500
3250
3000
0
2
4
6
8
10
12
14