У нас даны результаты продаж у какого-то товара *itemid* за неделю week. В качестве признаков даются в каким-то виде (непонятно, в каком) продажи за предыдущие недели с номером week - $shift$. Задача - научиться предсказывать продажи, обучившись на трэйновой выборке.
1. Заметим, что из-за того, что тестовая выборка достаточно большая, там могут содержаться признаки не за прошлое, а за настоящее. Конкретнее: если у нас дана строчка *itemid* = a, week = b, мы можем попытаться найти любую строчку с тем же *itemid*, week = b + i, shift = i, и взять признаки f1...f60 оттуда, оставив наш y. Если мы обучимся на этих признаках, наш результат заметно выиграет. К сожалению, это решение имеет мало отношения к реальной жизни.
2. Заметим, что таргет у нас порядка 10^4 и больше. Это плохо, можем переобучиться, и маленькие изменения x могут стать причиной большого изменения y (вообще говоря, у нас градиентный бустинг, а не линейная модель, но все равно). Решение: прологарифмируем таргет, обучимся, предскажем, возведем предсказания в экспоненту.
3. Обучим XGBRegressor из модуля xgboost.
P.S. для тех строк, для которых мы не нашли хороших признаков, обучим и запустим xgboost на обычных. Решением будет комбинация этих двух алгоритмов в зависимости от признаков.

Потом выяснилось, что после проведенной обработке над признаками ответы y почти совпадают со значением признака f30. Я попробовала запустить линейную модель на одном этом признаке, чтобы он сам вычислил коэффициент. SMAPE - 39. Xgboost сработал не лучше. Видимо, остальные признаки тоже несут смысл.

Вообще говоря, я проводила подбор параметров по GridSearch, и параметры из бэйзлайна оказались самыми лучшими. GridSearch из кода был удален, чтобы не мешался. То же случилось и с кодом кросс-валидации
Линейная модель учится слишком долго.
Если обучаться на выборке размером больше 0.2 от train, то результаты хуже (переобучение)
Поскольку SMAPE в sklearn нет, я проводила кросс-валидацию со mean squared error и mean absolute error для моего кода и кода бэйзлайна. Разница всегда была ощутимая.

1. Sample submission - это среднее по всем y из train.

2.В sklearn базовым классом для решающих деревьев является BaseDecisionTree. Внутри инициализируется экземпляр класса BestFirstTreeBuilder или DepthFirstTreeBuilder, которые имеют функцию build иотвечают за построение деревьев. Этот экземпляр принимает на вход экземпляр класса splitter, который и производит разбиение в каждом ноде. В начале фичи сортируются так, чтобы в начале шли те, которые принимают константные значения на всей выборке (чтобы не учитывать их). Затем сплиттер достает случайный признак, выборка сортируется по этому признаку. Для каждой позиции разбиения высчитывается значение критерия функцией cruterion.update, считается его улучшение, выбирается наилучшее разбиение.
В целом, мы в домашнем задании делали то же самое, но мы проходились по всем фичам, а здесь берется их случайное подмнрожество размера max features каим-то хитрым алгоритмом выбора без повторения Фишера-Йейтса (так написано в коментариях к коду файла splitter.pyx), и этим алгоритм построения похож на алгоритм построения

решающего дерева в ходе построения случайного леса.