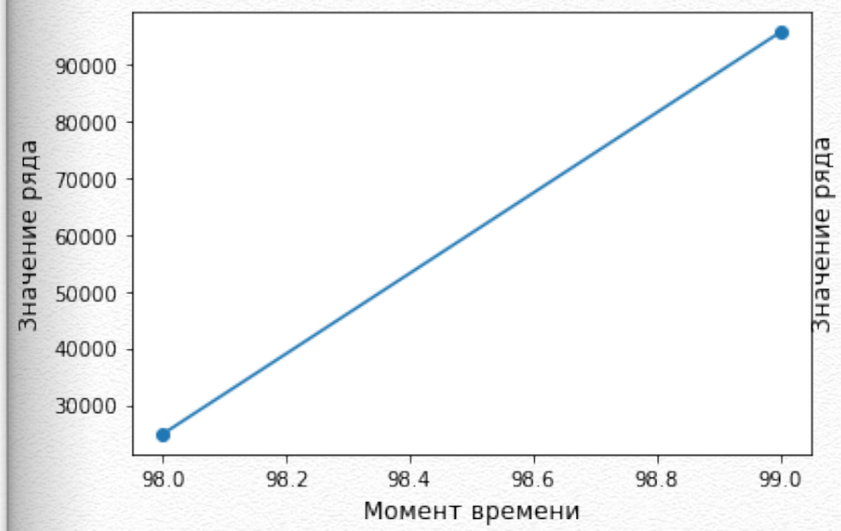
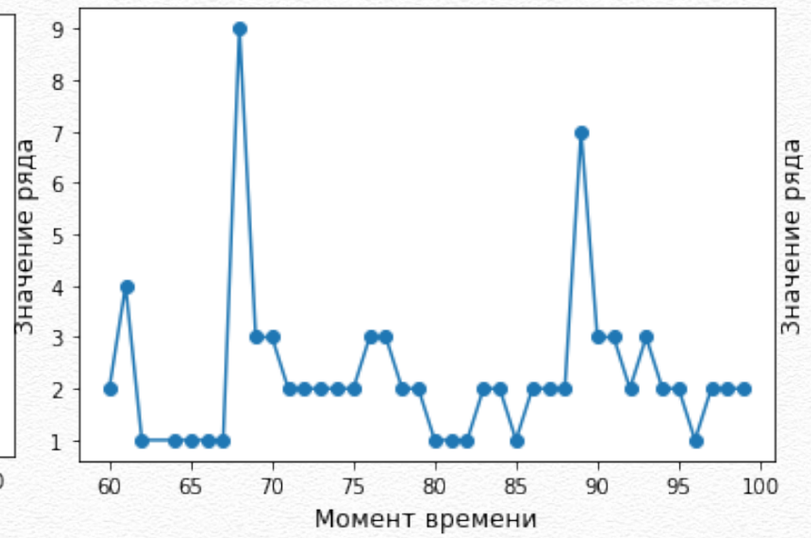
Предсказание временных рядов: стадии от отрицания до принятия

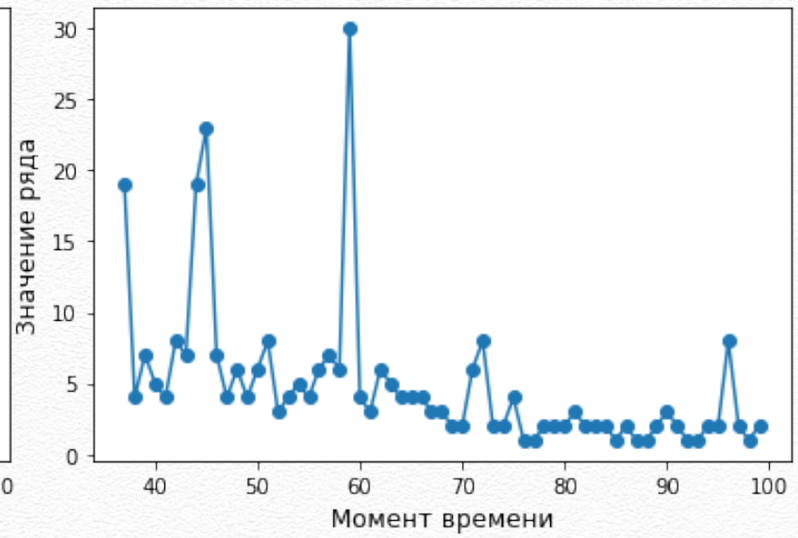
Милютина Лилия

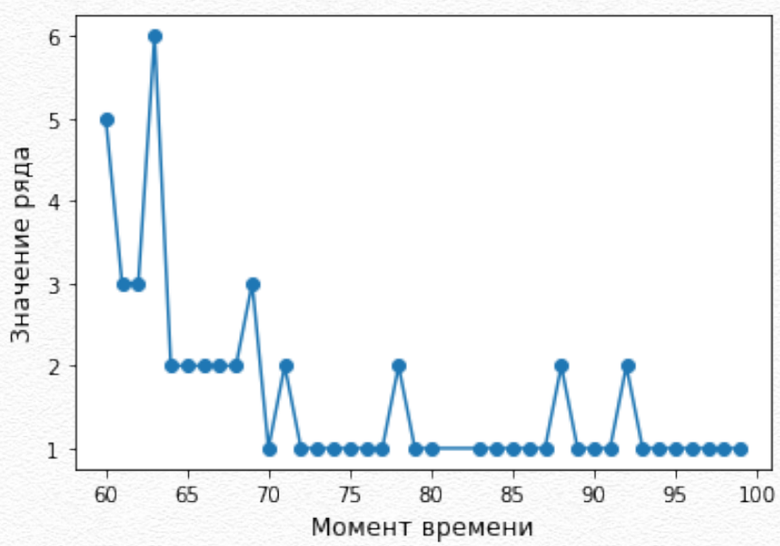
Анализ данных о временных рядах

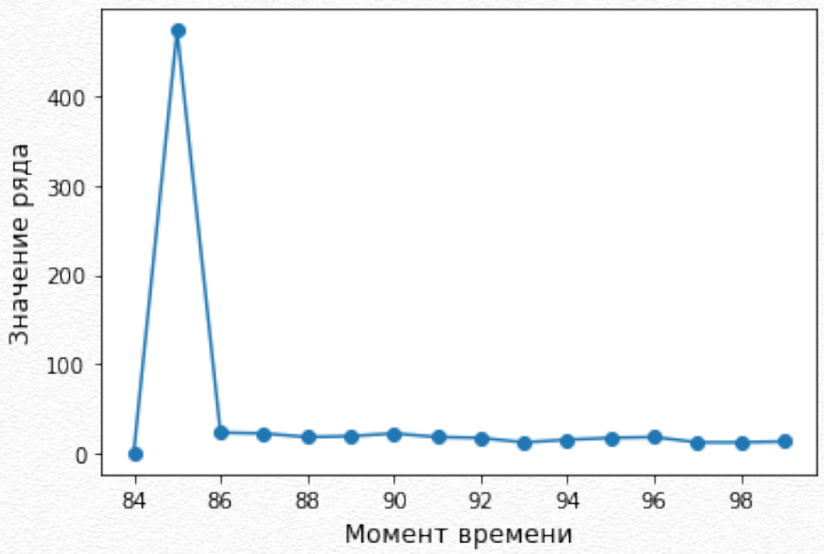
Посмотрим на несколько временных рядов





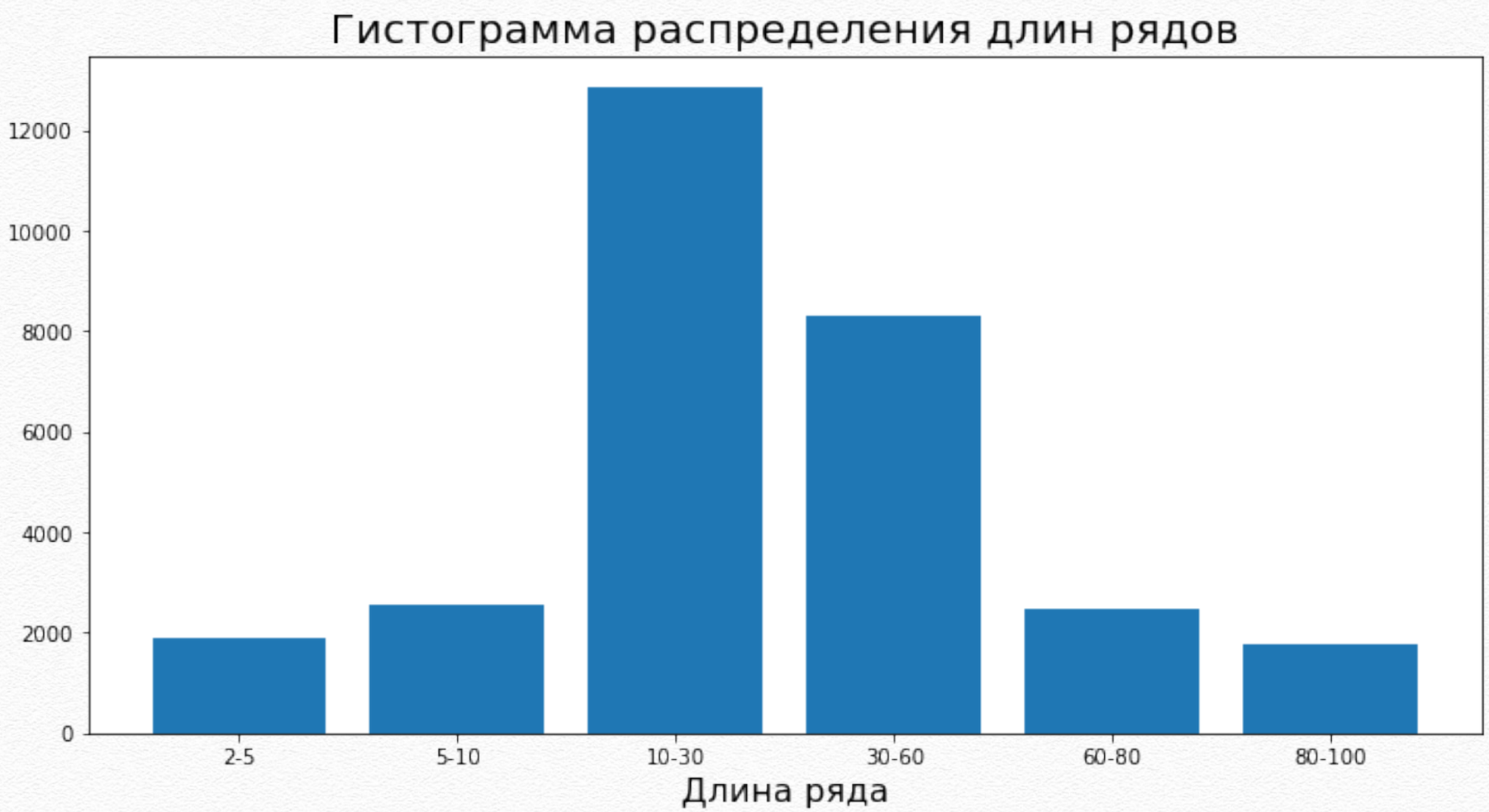






Посмотрев на первые 5 экземпляров, немного пугаешься…
Есть ряды с двумя точками, есть — с огромными выбросами.

Анализ данных о временных рядах



Однако гистограмма распределения внушает надежды: маленьких рядов довольно немного (рядов с длиной меньше 10 не более 15 %)

Построение алгоритма. Часть 1.

Увидев количество рядов, захотелось сразу построить какой-то сложный алгоритм, учитывающий все закономерности в данных. А это большая ошибка, до конца не хотелось в верить в то, что простые алгоритмы способны что-либо правильно предсказать.

Однако первый бейзлайн удалось побить скользящим средним по последним n-м элементам, где n подбиралось по валидации.

Построение алгоритма. Часть 1.

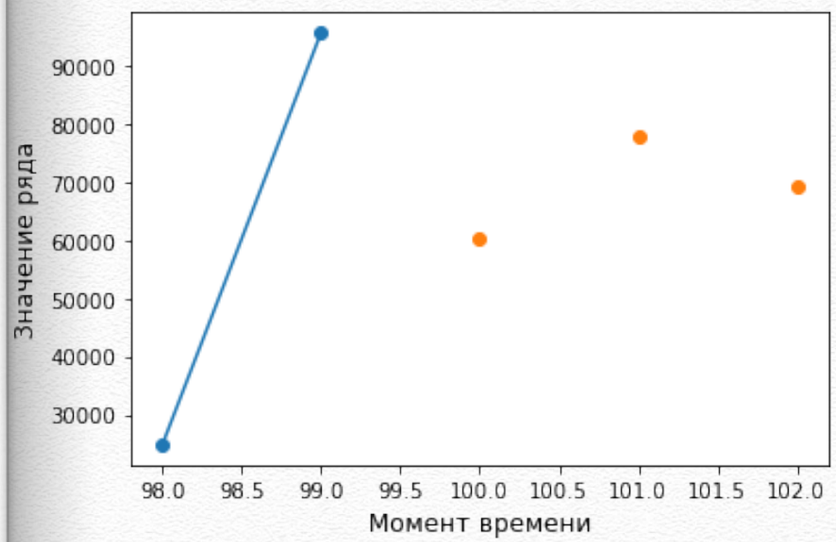
```python
# predict average

def pred_av(vec, win_size):
    vec1 = vec.copy()

    v1 = moving_average(vec1, win_size)
    vec1 = vec1 + [v1]

    v2 = moving_average(vec1, win_size)
    vec1 = vec1 + [v2]

    v3 = moving_average(vec1, win_size)
    vec1 = vec1 + [v3]

    el_temp = [v1, v2, v3]

    el_temp = [round(el) for el in el_temp]
    return el_temp
```

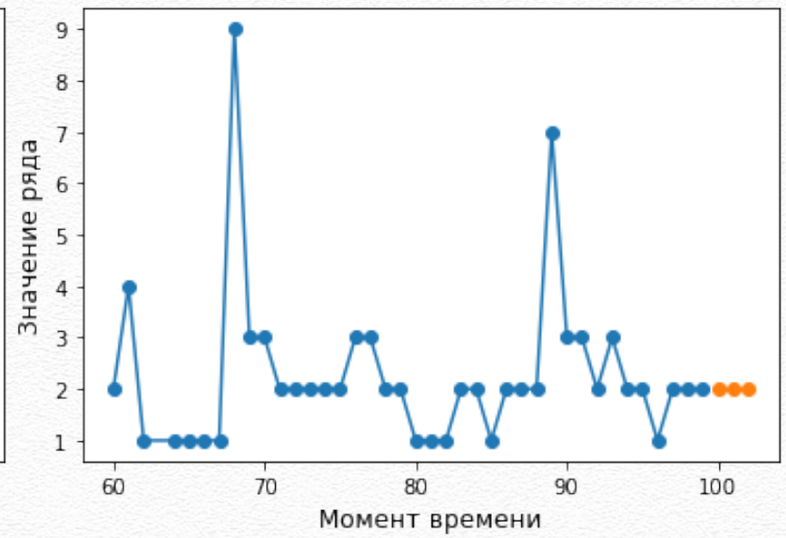
```python
# validation and choose optimal window size

win_size_vec = []

win_s = range(2, 11)
errors_best = []

for el in values:
    if (len(el) > 5):
        errors = []
        for win in win_s:
            res = pred_av(el[-3:], win)
            errors.append(err(el, res))
        errors_best.append(min(errors))
        error_min_index = errors.index(min(errors))
        win_size_vec.append(win_s[error_min_index])
    else:
        win_size_vec.append(2)
```

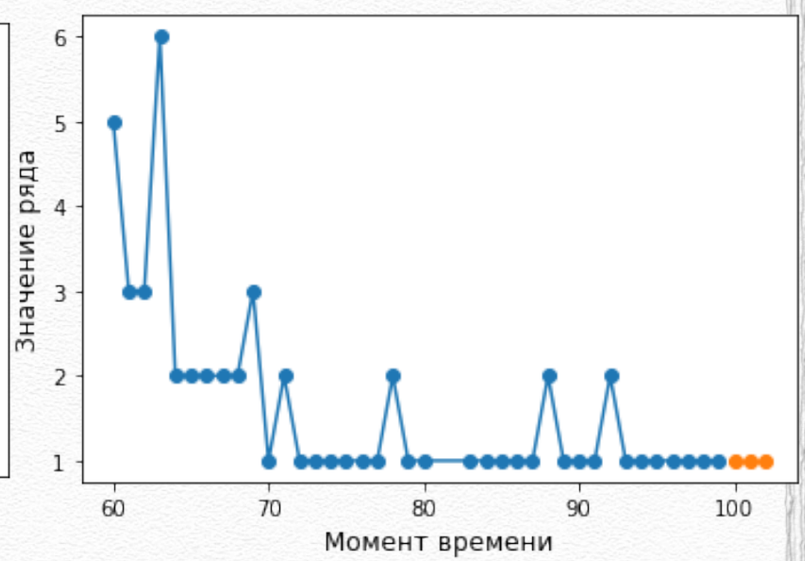
Валидация производилась по предсказаниям последних трех точек, для рядов длиной меньше 5, окно по умолчанию выбиралось равным 2

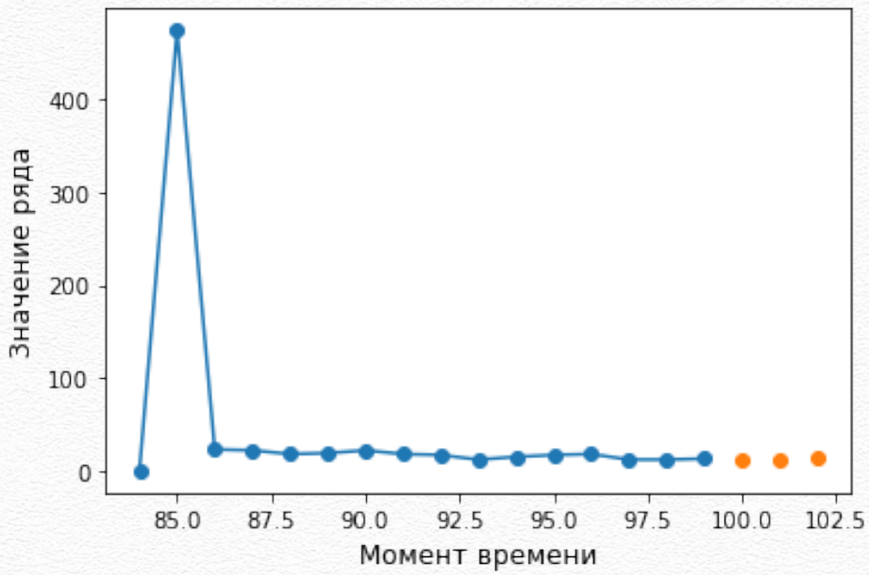
Удивило, что среднее показало себя лучше, нежели, чем медиана, хотя вроде бы она робастна к выбросам и ошибка более настроена на нее, чем на среднее

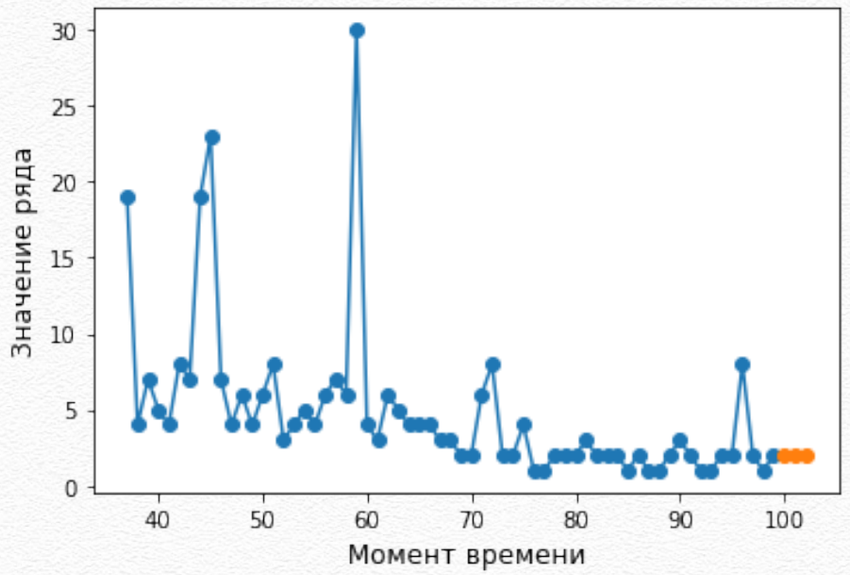
Построение алгоритма. Часть 1.









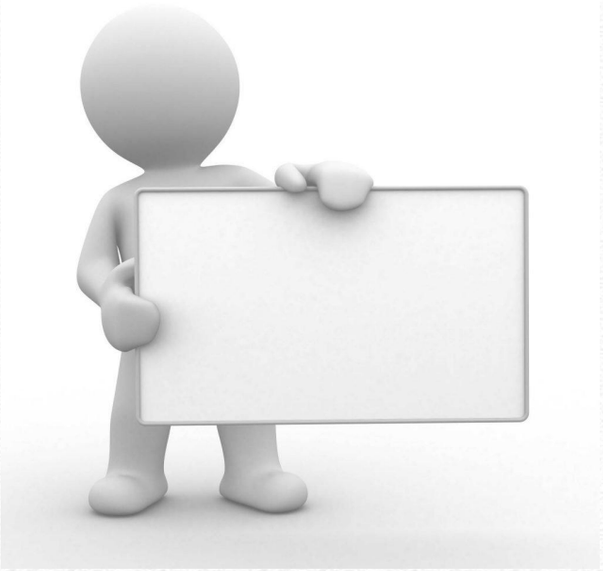


Из графиков судить что-либо очень сложно, но чисто зрительно предсказания для больших рядыов выглядят приемлемо, а вот на малых, на взгляд, ошибка велика.

Первоначальные итоги

Данное решение дало скор 0.54922, что значительно лучше, чем я пыталась вначале накрутить ансамбли из разных алгоритмов: медианы разных окон, линейная регрессия. Ведь казалось, что именно линейная регрессия сможет уловить тренд на рядах с малым количеством точек, однако, для линейной регрессии, нужны хорошие фичи, которых у меня на тот момент еще не было. Стадия отрицания простых алгоритмов сменилась на стадию их принятия.

Построение алгоритма. Часть 2.



Но все же хотелось повторить модель машинного обучения. Так как рядов очень много и некоторые из них очень немногочисленны, то обучать модель под каждый ряд очень долго и неэффективно, поэтому я построила единый датафрейм и настраивала модель сразу на всех рядах.

Основные особенности:

1. Добавление лагов, количество подбиралось по валидации.
2. Статистические данные о ряде по окну разной ширины (среднее, медиана, максимум, минимум, дисперсия, квантили, длина ряда).
3. Добавление информации о рядах.

Построение алгоритма. Часть 2.

Часть полученного датафрейма.

		y	lag_1	lag_2	lag_3	lag_4	lag_5	lag_6	lag_7	lag_8	lag_9	lag_10	lag_11	lag_12	lag_13	lag_14	lag_15
num	time																
0	99	95644.0	24850.0	24850.0	24850.0	24850.0	24850.0	24850.0	24850.0	24850.0	24850.0	24850.0	24850.0	24850.0	24850.0	24850.0	24850.0
1	70	3.0	3.0	9.0	1.0	1.0	1.0	1.0	1.0	4.0	2.0	1.0	1.0	1.0	1.0	1.0	1.0
	71	2.0	3.0	3.0	9.0	1.0	1.0	1.0	1.0	1.0	4.0	2.0	1.5	1.5	1.5	1.5	1.5
	72	2.0	2.0	3.0	3.0	9.0	1.0	1.0	1.0	1.0	1.0	4.0	2.0	2.0	2.0	2.0	2.0
	73	2.0	2.0	2.0	3.0	3.0	9.0	1.0	1.0	1.0	1.0	1.0	4.0	2.0	2.0	2.0	2.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
29859	95	1.0	0.0	1.0	1.0	0.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
	96	0.0	1.0	0.0	1.0	1.0	0.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
	97	1.0	0.0	1.0	0.0	1.0	1.0	0.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
	98	1.0	1.0	0.0	1.0	0.0	1.0	1.0	0.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
	99	1.0	1.0	1.0	0.0	1.0	0.0	1.0	1.0	0.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0

Пропущенные значения заполнялись последним валидным значением.

Построение алгоритма. Часть 2.

Таким образом, получался огромный датафрейм, но благодаря валидации были отобраны наиболее значимые фичи, а также взяты последние 30 наблюдений.
В число значимых фичей попали все лаги (их было 10 в модели), а также статистические характеристики по окнам и некоторые категории, так что дополнительная информация дала пользу.

Построение алгоритма. Часть 2.

Валидация

Валидация производилась на один шаг вперед, причем учитывались веса объектов, чтобы лучше подстроиться под метрику MASE.

```
parameters = {'learning_rate': [0.01, 0.05, 0.1, 0.15, 0.2],
              'n_estimators': [300, 500, 800, 1000, 1500, 2000],
              'max_depth' : [5, 7, 9, 11, 13],
              'metric' : ["mae"],
              'reg_lambda': [0, 0.5, 1, 1.5, 2],
             }

model = GridSearchCV(LGBMRegressor(silent=False), param_grid=parameters, cv=4, n_jobs=-1, verbose=3)

model.fit(X_train, y_train, sample_weight = weights_all)

y_pred = model.predict(X_test)
y_pred_ = [int(round(el)) for el in y_pred]
```

Построение алгоритма. Часть 2.

Перепробовав разные модели, лучше всего получилось пофитить градиентный бустинг. Однако очень важно было подобрать параметры, что не сразу получилось, и скор совсем был не тот, что хотелось. Основная ошибка заключалась в том, что настройка производилась не под нормированный MAE, что, конечно, сильно искажало картину.

В итоге данная модель дала скор 0.53490

tsfresh

В ходе поиска полезных библиотек для анализа временных рядов была обнаружена tsfresh. Хоть она и не была использована в последней версии моего решения, однако с ней я тоже познакомилась.
Она оказалось удобной для генерации дополнительных фичей.

Summary

Без преувеличения эта задача была настоящей школой жизни: после апробирования огромного количества наворотов (ансамбли с разными алгоритмами, причем решением бралось сначала среднее, потом медиана), экспоненциального среднего — и не преодоления даже первого бейзлайна, руки опускались. Но начав, как и следовало ранее, с самого простого: со скользящего среднего с адаптивным окном и получив первые результаты, дело сдвинулось с мертвой точки.

Пришлось принять, что наивные алгоритмы работают порою даже лучше, чем изощренные, и начинать следует именно с них.