Ускоряем Pandas и строим правильные графики

Александр Ольферук,
наставник

Яндекс Практикум

Ускорение pandas

Разбираемся с датами

	date_time
0	2013-01-01 00:00:00
1	2013-01-01 01:00:00
2	2013-01-01 02:00:00
3	2013-01-01 03:00:00
4	2013-01-01 04:00:00

Разбираемся с датами

```
def convert(df, column_name):
    return pd.to_datetime(df[column_name])
```

```
Best of 3 trials with 10 function calls per trial:
Function `convert` ran in average of 1.610 seconds.
```

```
def convert(df, column_name):
    return pd.to_datetime(df[column_name],
                          format='%d/%m/%y %H:%M'
                         )
```

```
Best of 3 trials with 100 function calls per trial:
Function `convert_with_format` ran in average of 0.032 seconds.
```

Разбираемся с датами

Если дата не в ISO 8601, то явно указываем ее формат, чтобы не делать лишние операции (которые, к слову, выполняются через **dateutil**)

Примеры:

2020-09-18

2020-09-18T16:29:08+00:00

2020-W38

2020-W38-5

https://en.wikipedia.org/wiki/ISO_8601

Никогда не делайте так!

```
for i in range(len(df)):
    df.iloc[i, 0] = ...
```

```
for index, row in df.iterrows():
    something = row['column_name']
```

Только так

```
df.apply(lambda row: row['column']*123, axis=1)
```

Нормально, в принципе

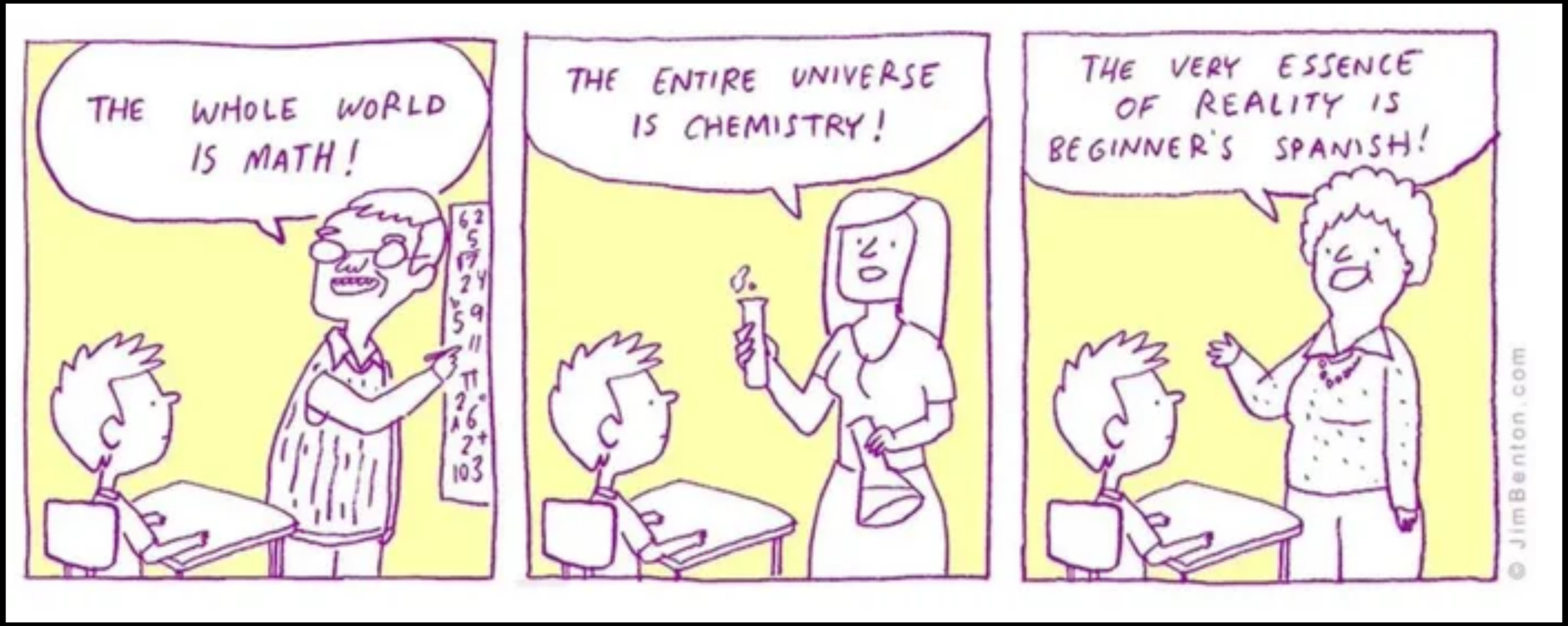
```
peak_hours = df.index.hour.isin(range(17, 24))
shoulder_hours = df.index.hour.isin(range(7, 17))
off_peak_hours = df.index.hour.isin(range(0, 7))

df.loc[peak_hours, 'cost_cents'] = df.loc[peak_hours, 'energy_kwh'] * 28
df.loc[shoulder_hours,'cost_cents'] = df.loc[shoulder_hours, 'energy_kwh'] * 20
df.loc[off_peak_hours,'cost_cents'] = df.loc[off_peak_hours, 'energy_kwh'] * 12
```

Еще лучше

Если задача позволяет, то лучше воспользоваться **pd.cut**

```
cents_per_kwh = pd.cut(x=df.index.hour,
                       bins=[0, 7, 17, 24],
                       include_lowest=True,
                       labels=[12, 20, 28]).astype(int)
df['cost_cents'] = cents_per_kwh * df['energy_kwh']
```

Сначала щютка



Суть

Все, что вы видите в pandas - **векторы** и **матрицы**.

Нужно потратить некоторое количество времени, чтобы научиться мыслить **векторами**, а не отдельными записями, пусть по первому времени и неудобно.

Любая **векторная** функция будет гарантированно быстрее.

Но groupby – не вектор!

Да, и поэтому можно костылить самому с помощью пакета **multiprocessing**

А можно не изобретать, что уже существует: Pandarallel

Можно почитать на Хабре

Numba

Переходим на сайт, смотрим

```
from numba import jit
import numpy as np

x = np.arange(100).reshape(10, 10)

@jit(nopython=True) # Set "nopython" mode for best performance, equivalent to @njit
def go_fast(a): # Function is compiled to machine code when called the first time
    trace = 0.0
    for i in range(a.shape[0]):   # Numba likes loops
        trace += np.tanh(a[i, i]) # Numba likes NumPy functions
    return a + trace              # Numba likes NumPy broadcasting

print(go_fast(x))
```

Numba

```
from numba import jit
import pandas as pd

x = {'a': [1, 2, 3], 'b': [20, 30, 40]}

@jit
def use_pandas(a): # Function will not benefit from Numba jit
    df = pd.DataFrame.from_dict(a) # Numba doesn't know about pd.DataFrame
    df += 1                        # Numba doesn't understand what this is
    return df.cov()                # or this!

print(use_pandas(x))
```

Numba

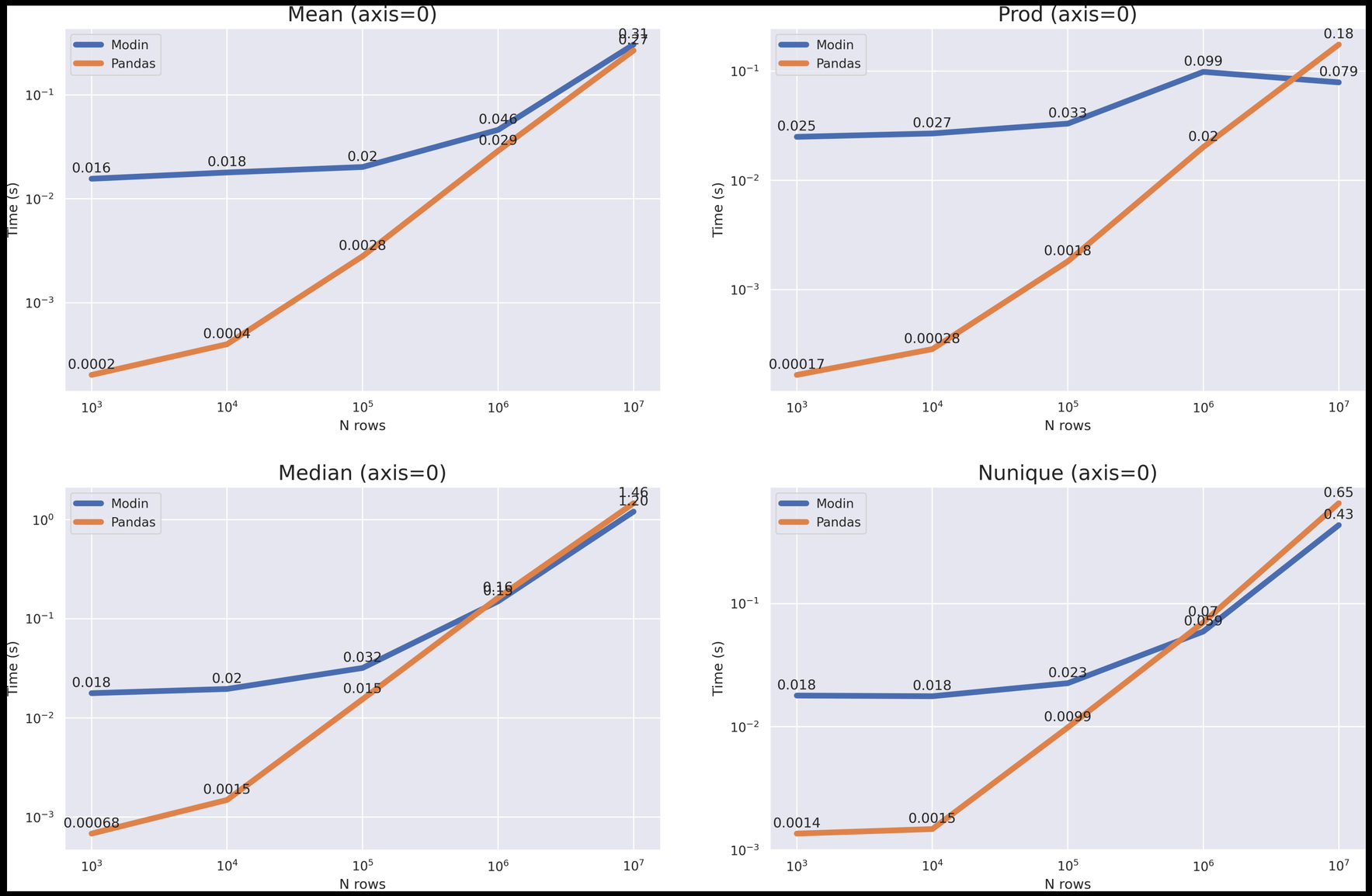
- Возможно добиться ускорения в тысячи раз
- Однако, если ускоряем не чистые математические функции – результат может быть не такой эффектный
- Иногда нужно переделать свой код под Numba

Modin

Ссылка на него

- Аналог pandas, **полностью повторяющий** API pandas
- Эффективен при чтении больших файлов
- Однако с табличками меньшего размера у него проблемы (см. след. слайд)
- Киллер-фича – возможность использовать дисковое пространство, если объема оперативной памяти не хватает
- Норм, если файл 1Gb+

Modin


Mean (axis=0)
Modin
Pandas
Time (s)
N rows
0.016
0.018
0.02
0.046
0.31
0.27
0.029
0.0028
0.0004
0.0002
Prod (axis=0)
Modin
Pandas
Time (s)
N rows
0.025
0.027
0.033
0.099
0.079
0.18
0.02
0.0018
0.00028
0.00017
Median (axis=0)
Modin
Pandas
Time (s)
N rows
0.018
0.02
0.032
0.16
1.46
1.20
0.015
0.0015
0.00068
Nunique (axis=0)
Modin
Pandas
Time (s)
N rows
0.018
0.018
0.023
0.07
0.059
0.65
0.43
0.0099
0.0014
0.0015

Общие советы и что почитать

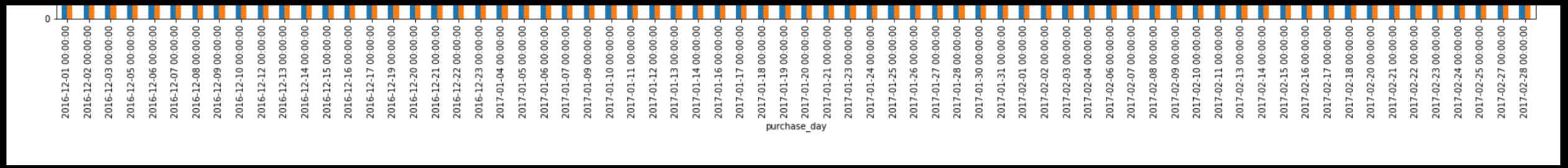
- Статья на Medium, из которой вы узнаете, что numpy работает быстрее pandas, и возможно, вызов **.values** вам не повредит
- Статья по сравнению pandas и Modin
- Тетрадка по измерению query и eval (которые я, к слову, ненавижу)
- Неплохая статья из документации pandas по теме ускорения вычислений
- Какая-то невероятная вандервафля про работе с табличками на GPU! Статья и репозиторий

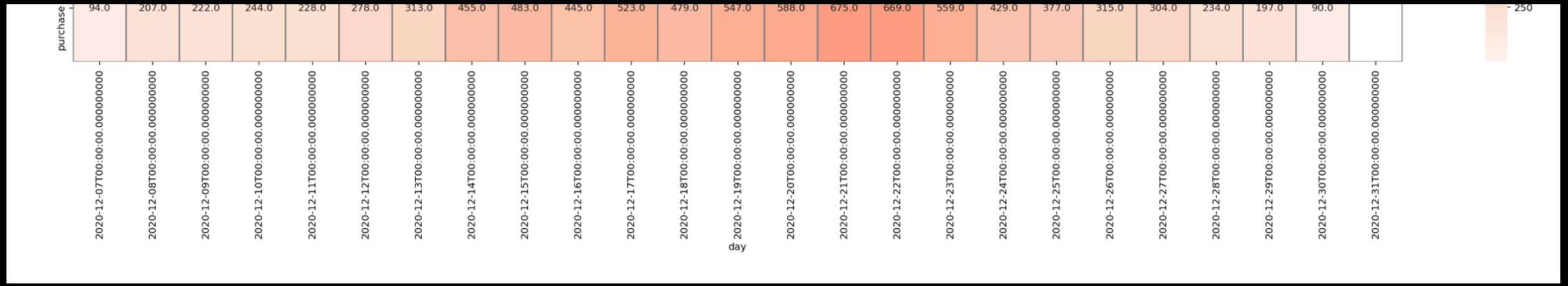
Перерыв

5 минут

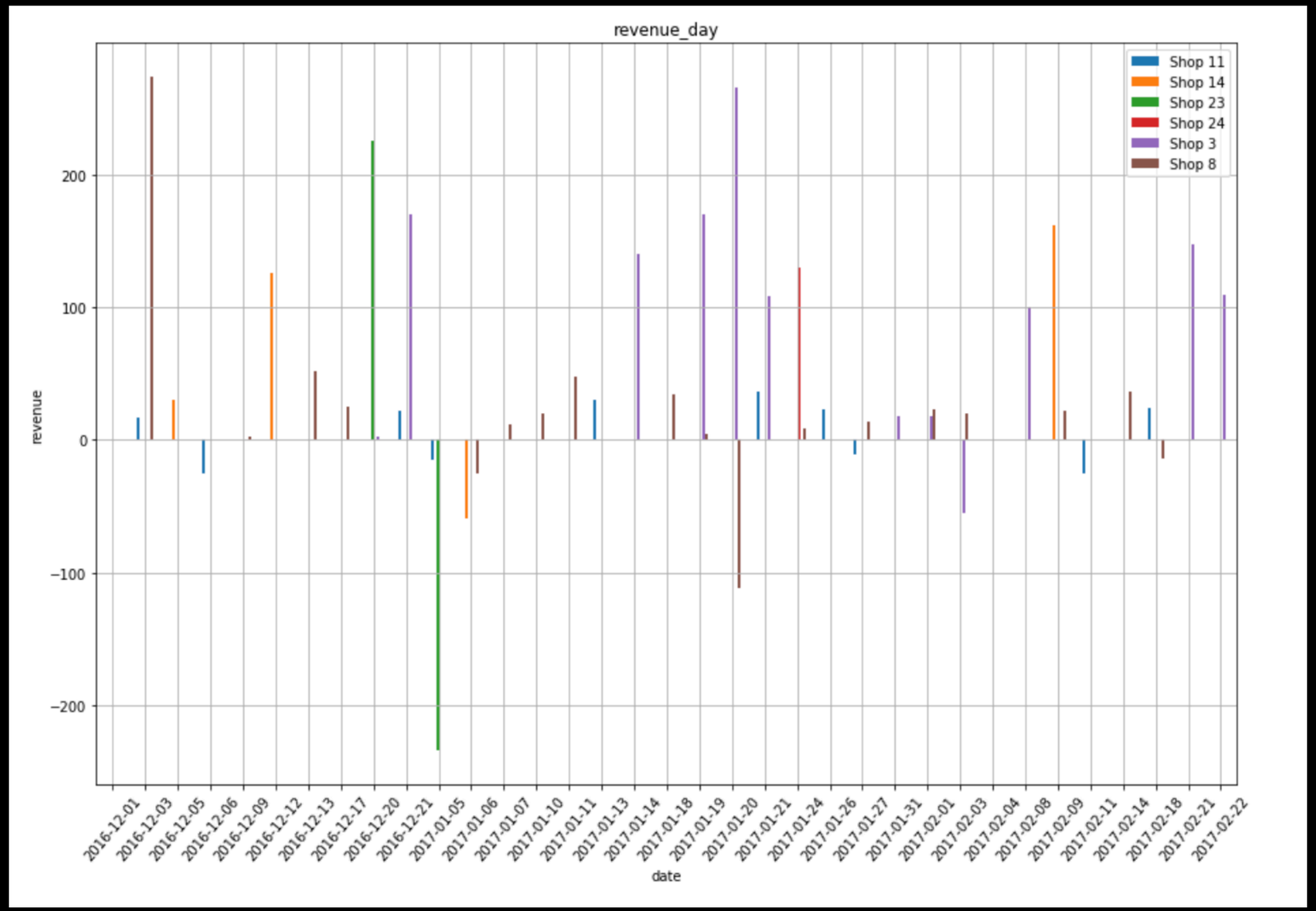
Ошибки визуализации

Форматирование дат и времени

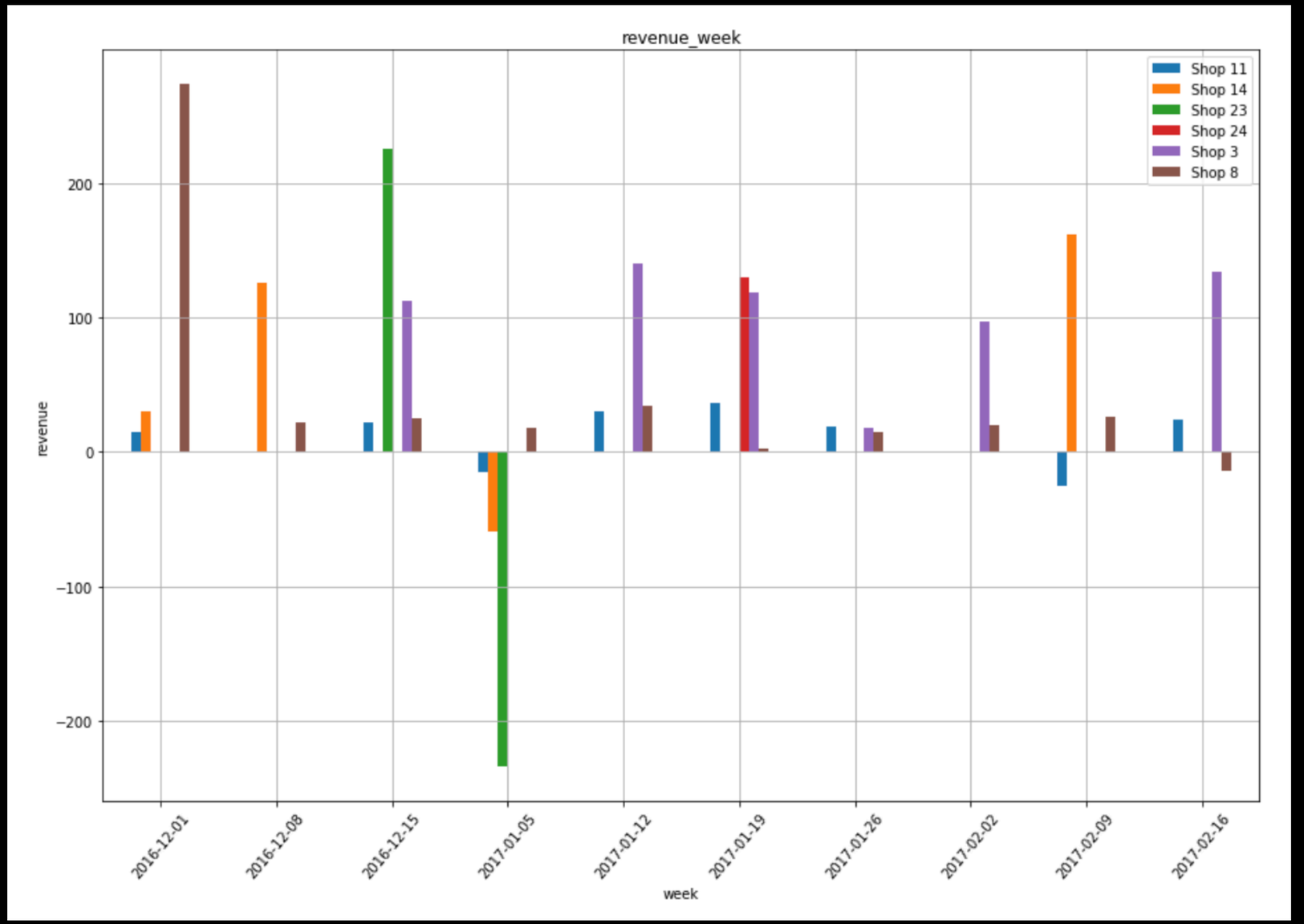




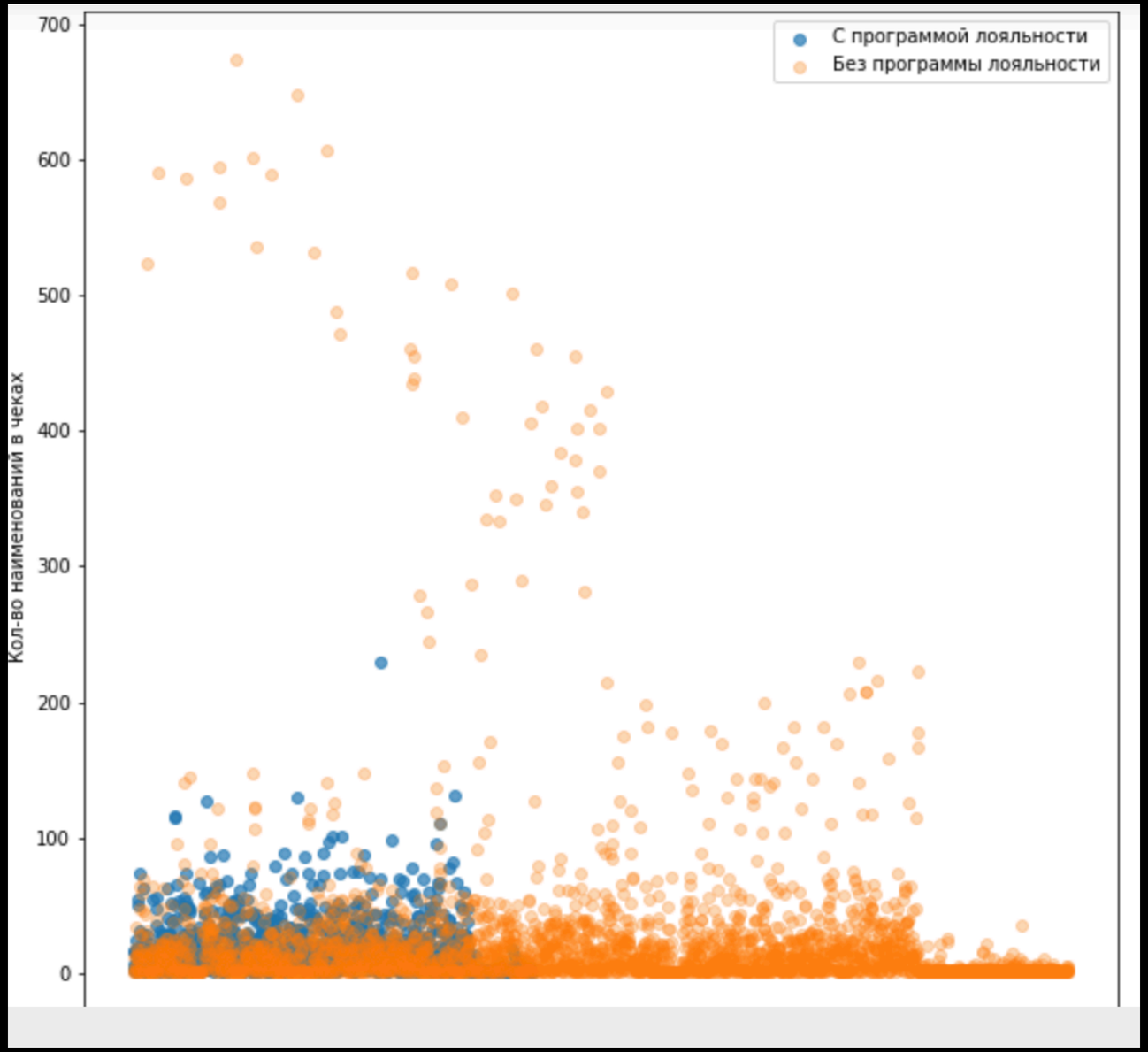
Неверный выбор графика



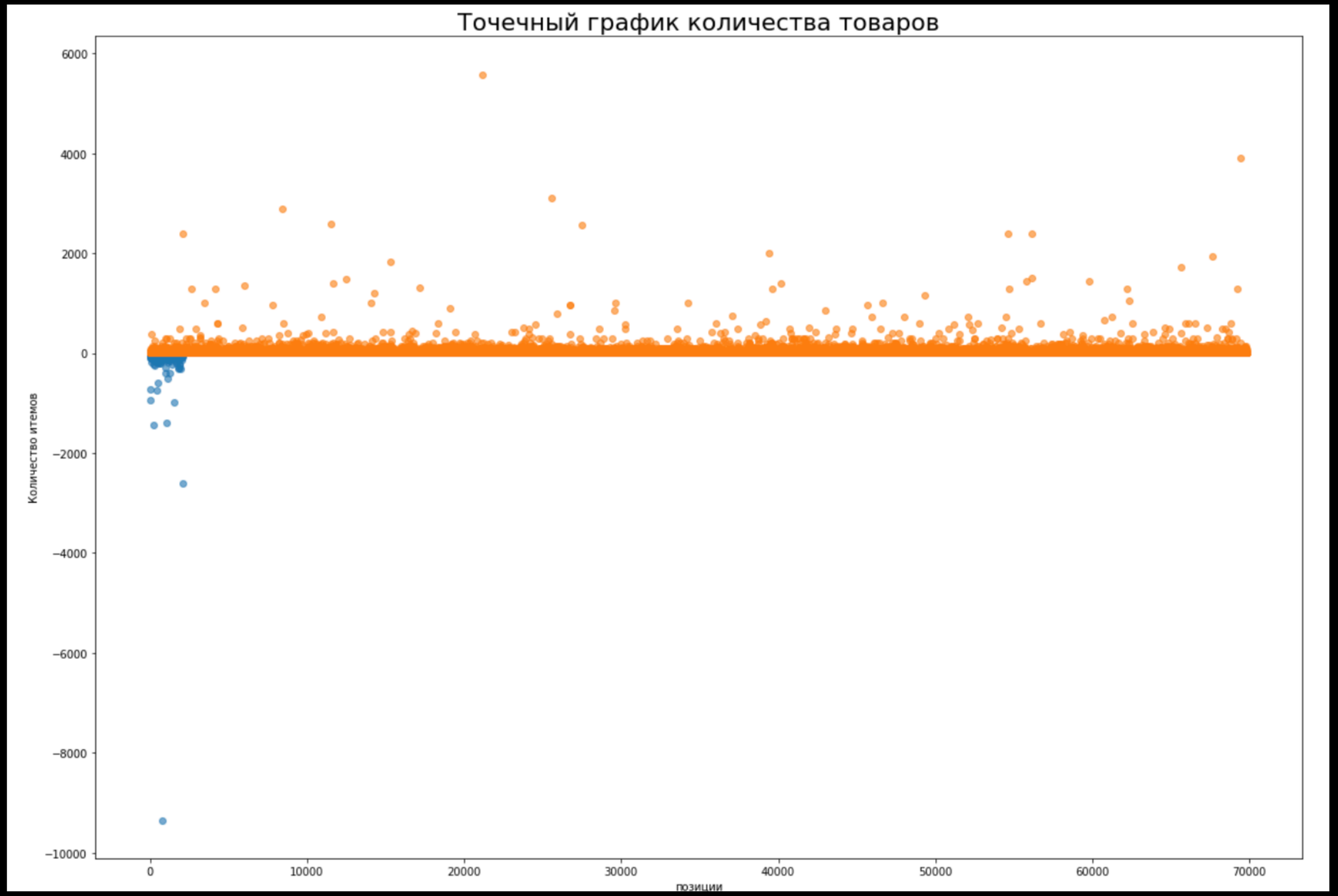
Неверный выбор графика



Неверный выбор графика №2

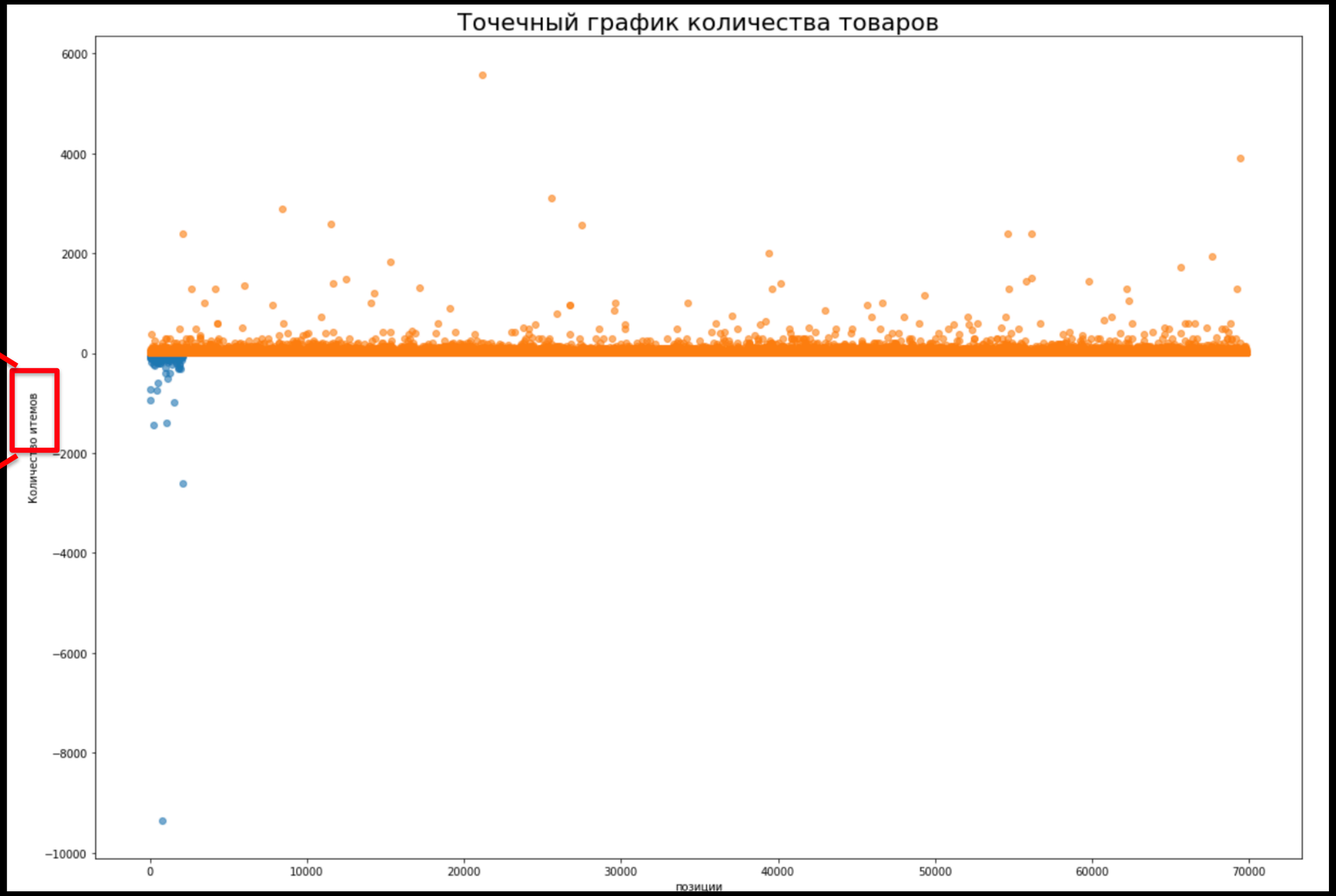


Неверный выбор графика №2

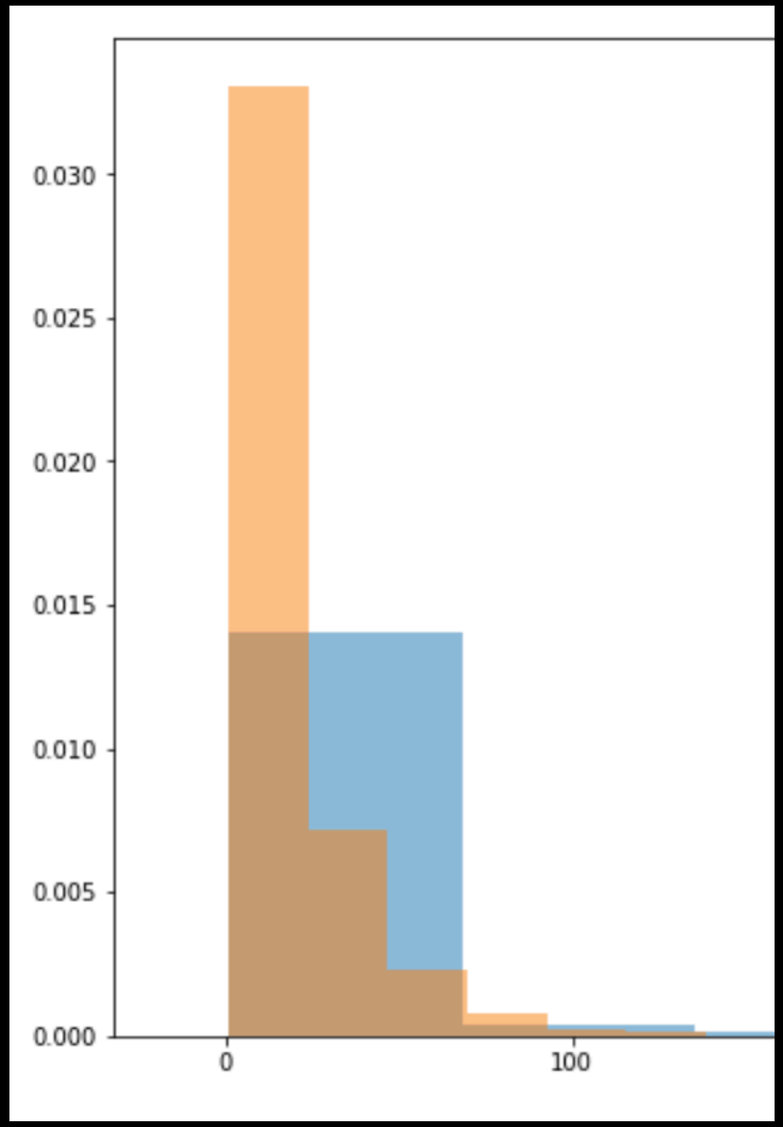


Неверный выбор графика №2

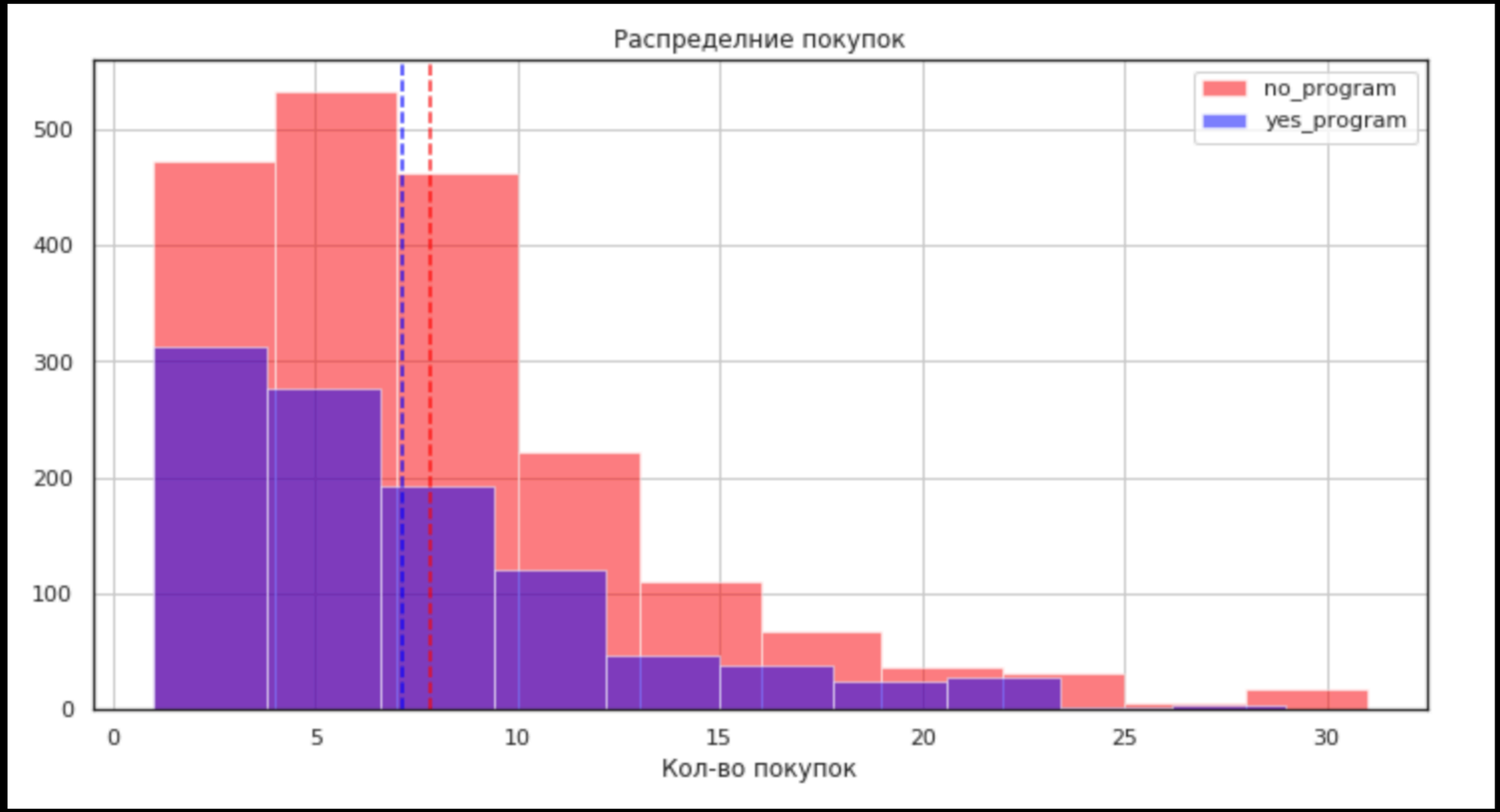
тво итемов



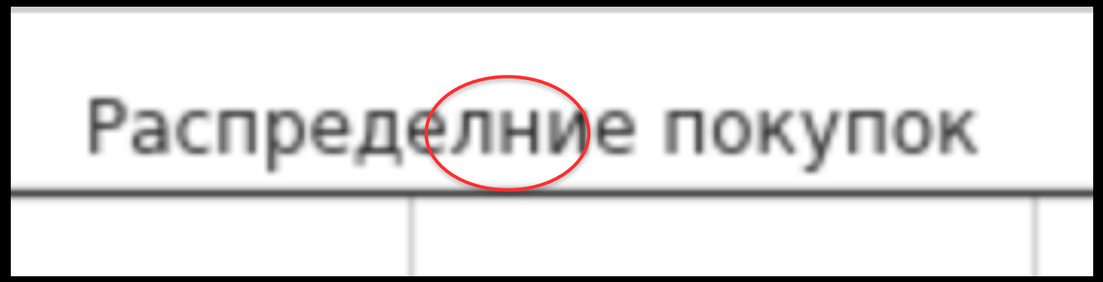
Неравномерные бины гистограммы



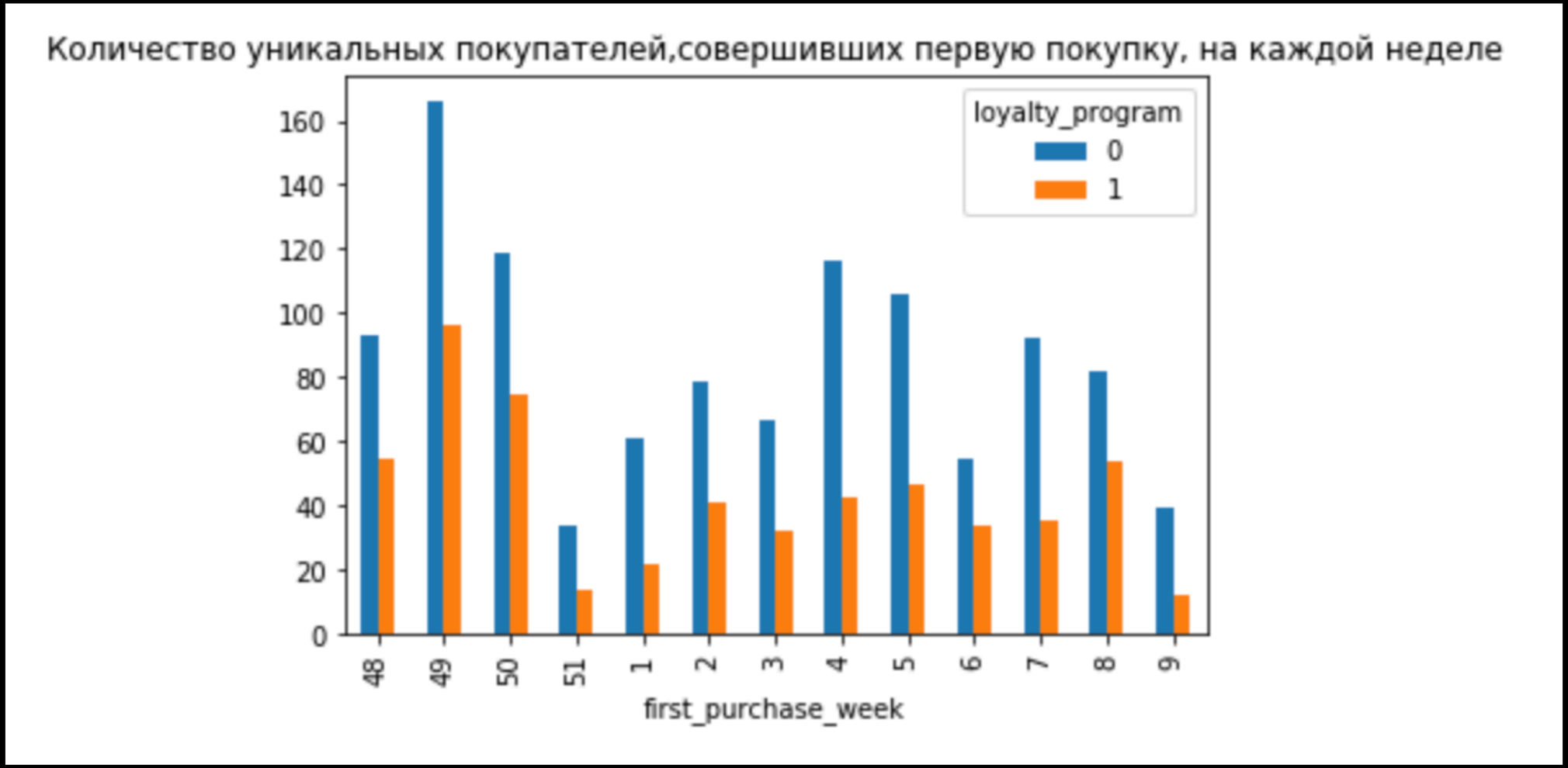
Неравномерные бины гистограммы



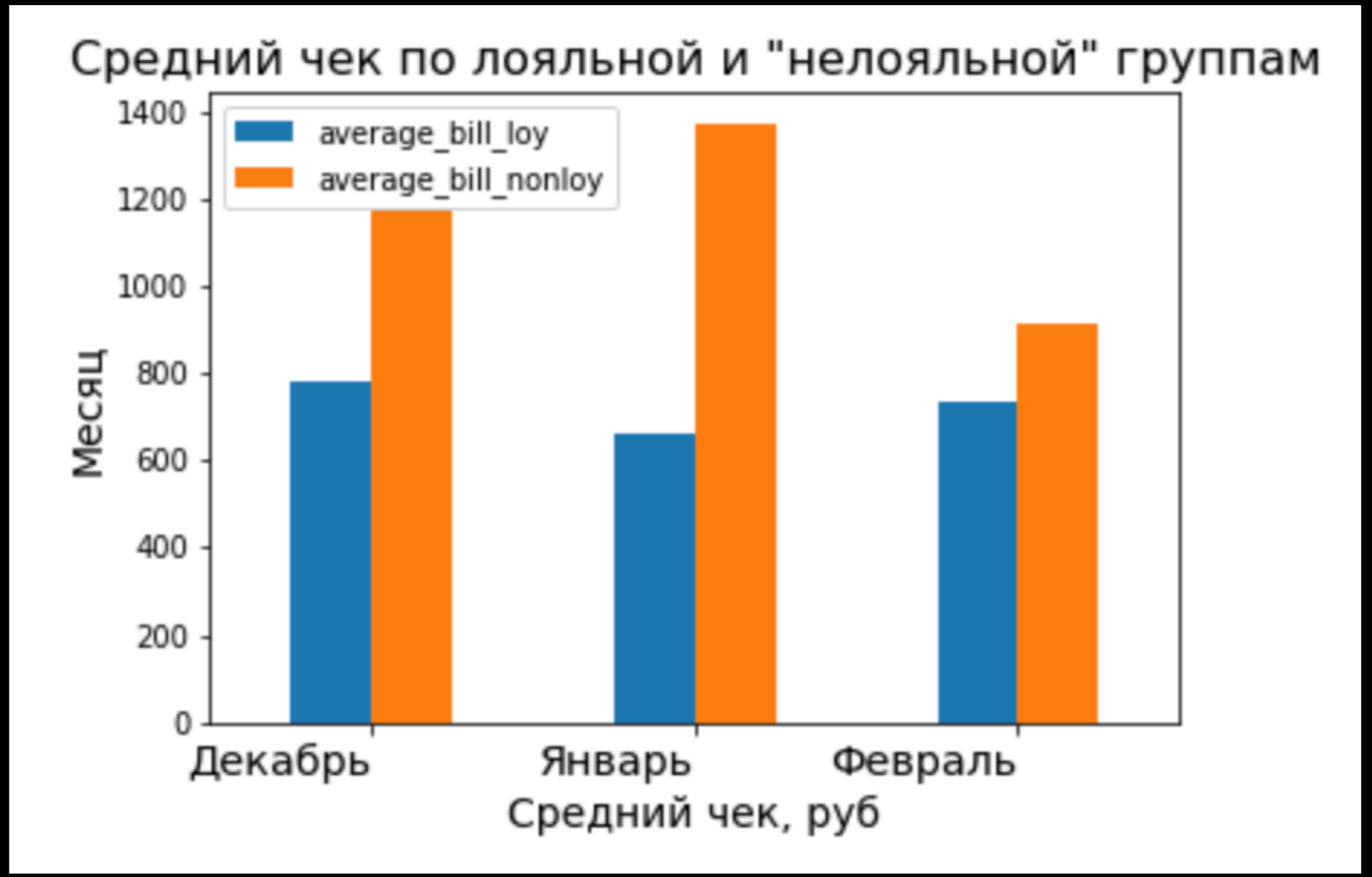
Грамматика и правописание



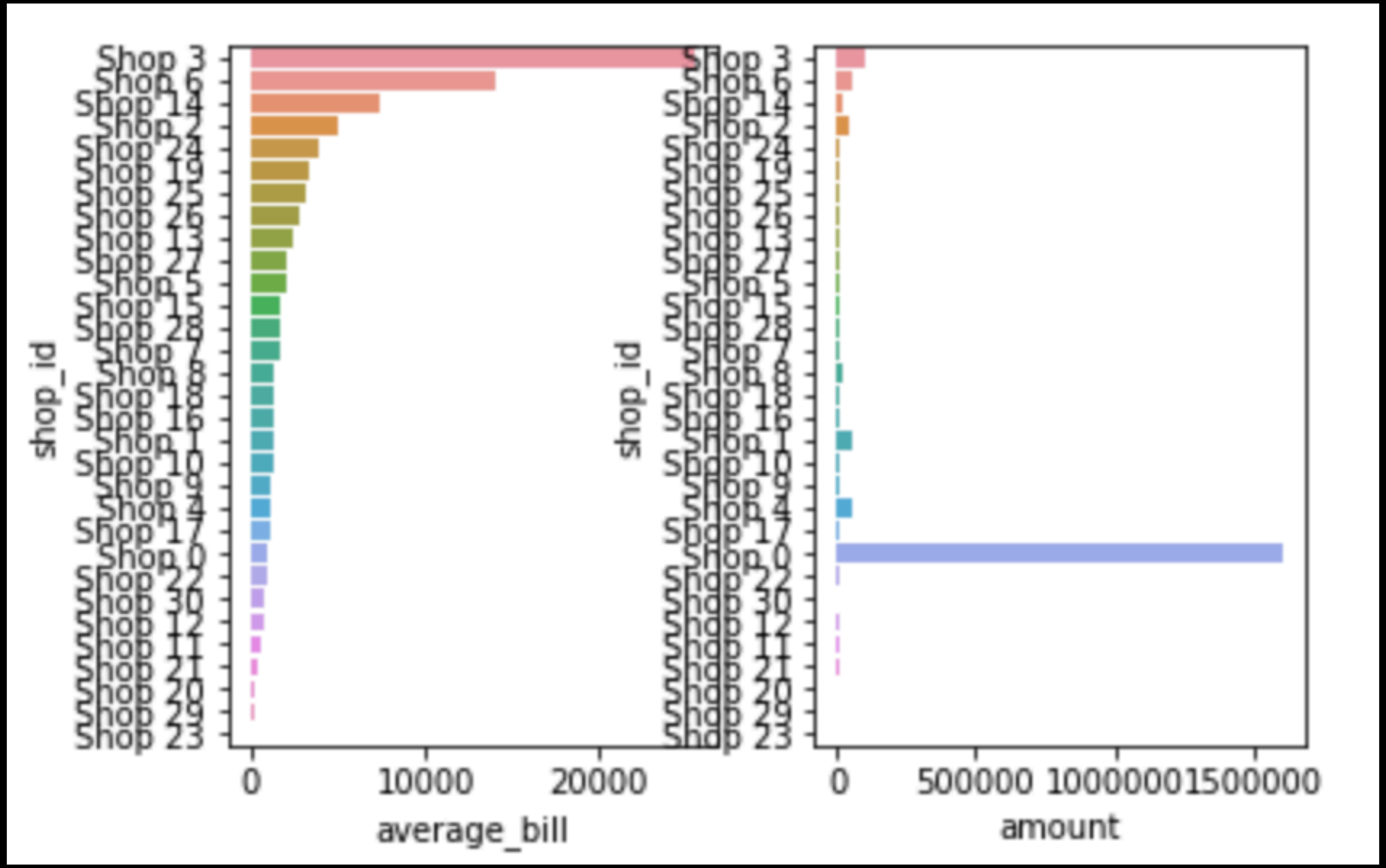
Смешивание языков



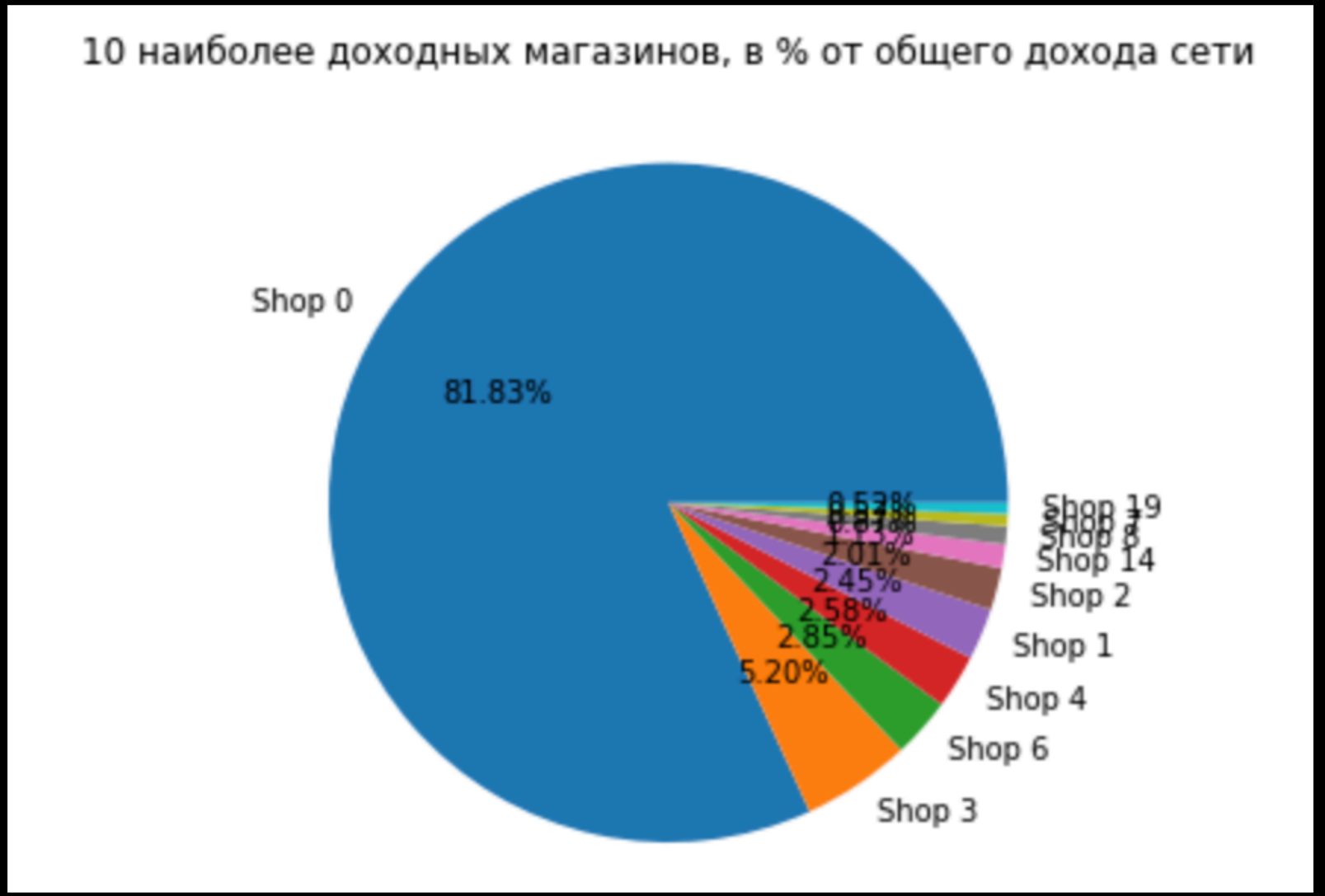
Перекрытия


Средний чек по лояльной и "нелояльной" группам
average_bill_loy
average_bill_nonloy
1400
1200
1000
800
600
400
200
0
Месяц
Декабрь
Январь
Февраль
Средний чек, руб

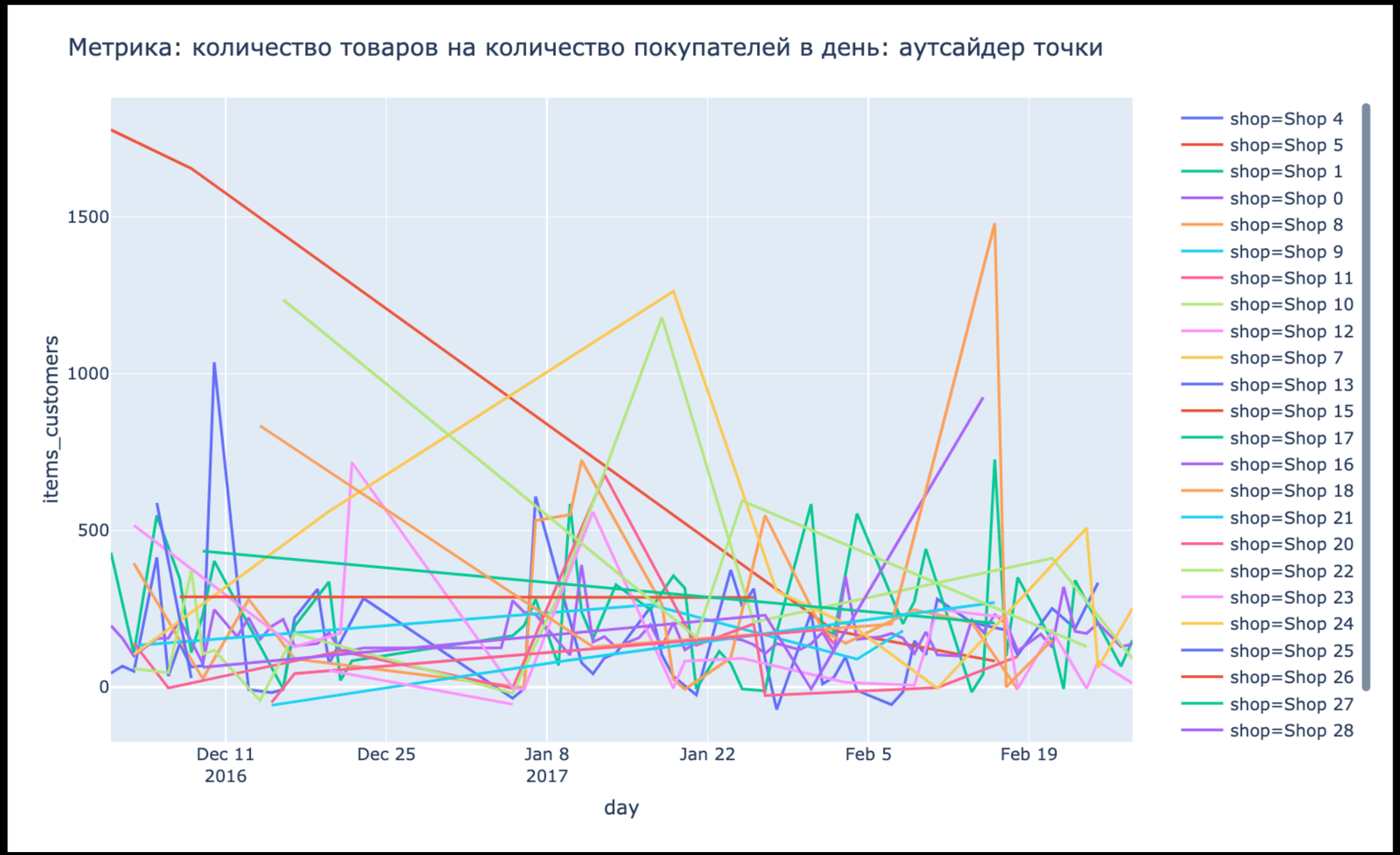
Слишком мелко



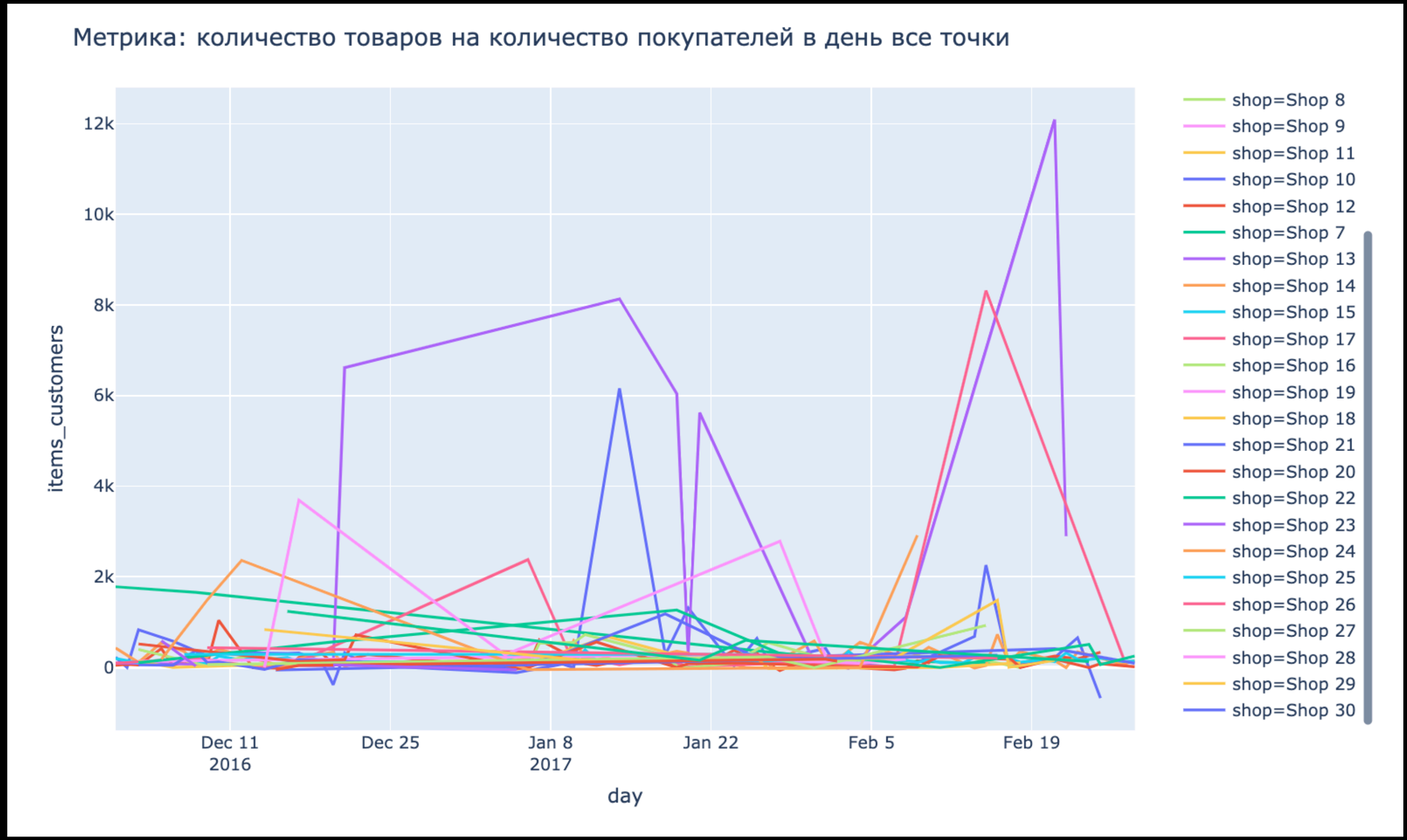
Слишком мелко



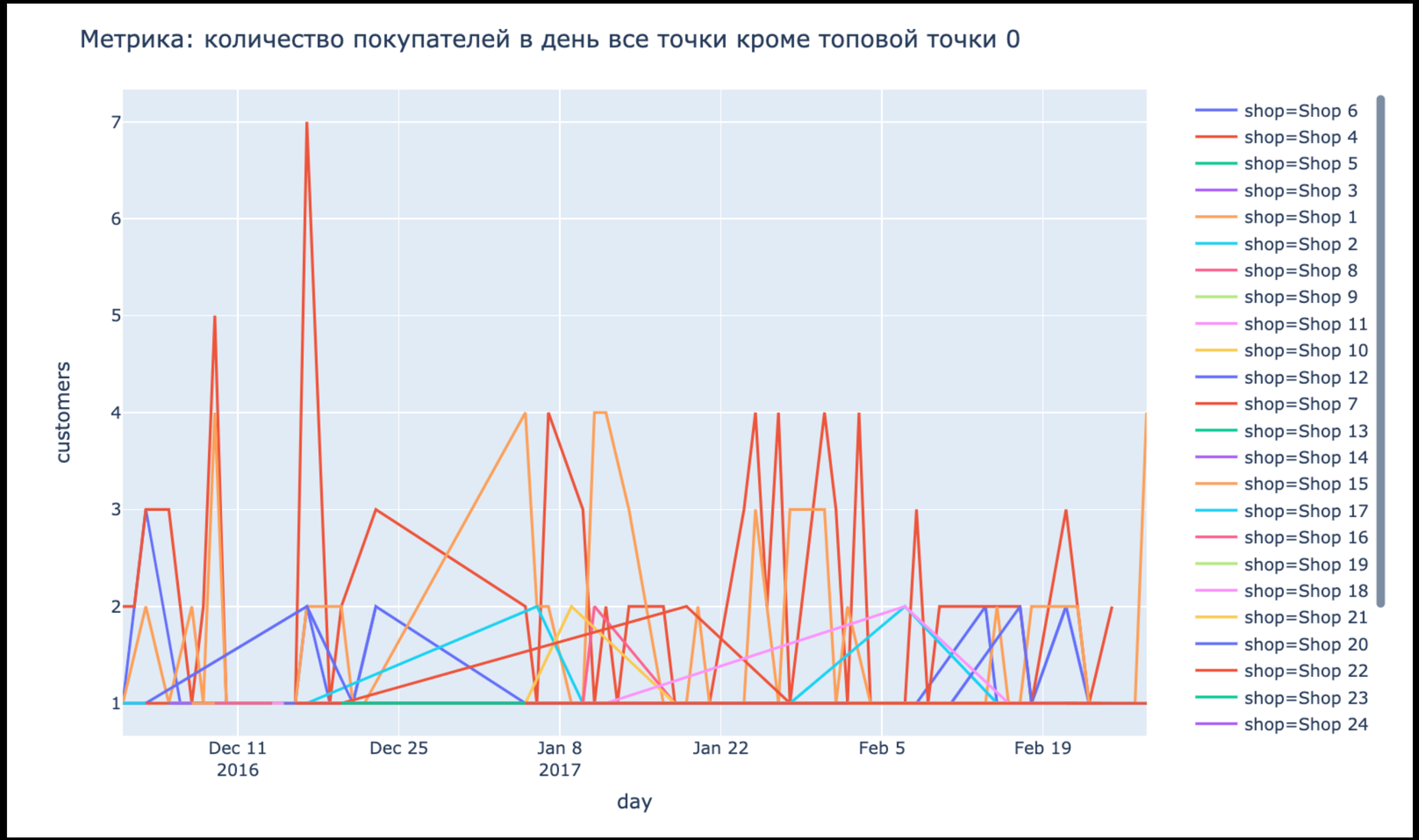
Современное искусство



Современное искусство



Современное искусство



Ускоряем Pandas и строим правильные графики

Александр Ольферук,
наставник

Яндекс Практикум