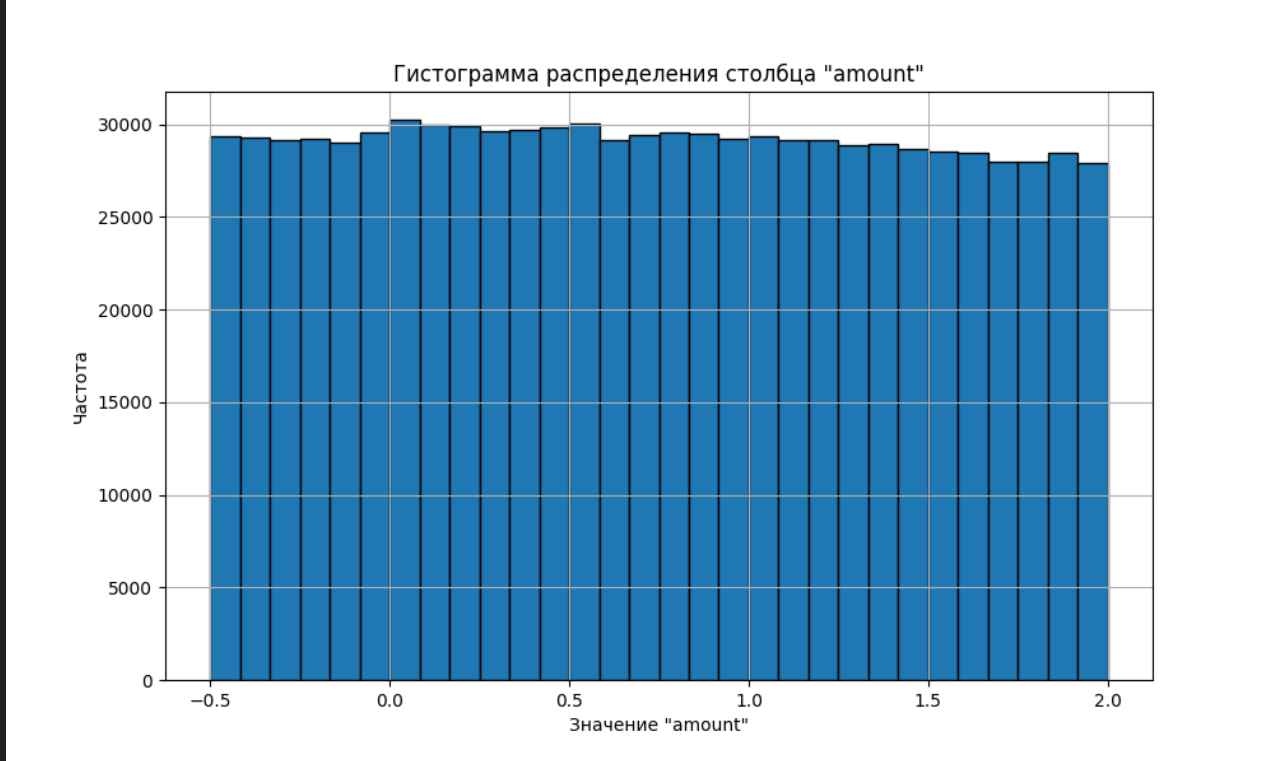
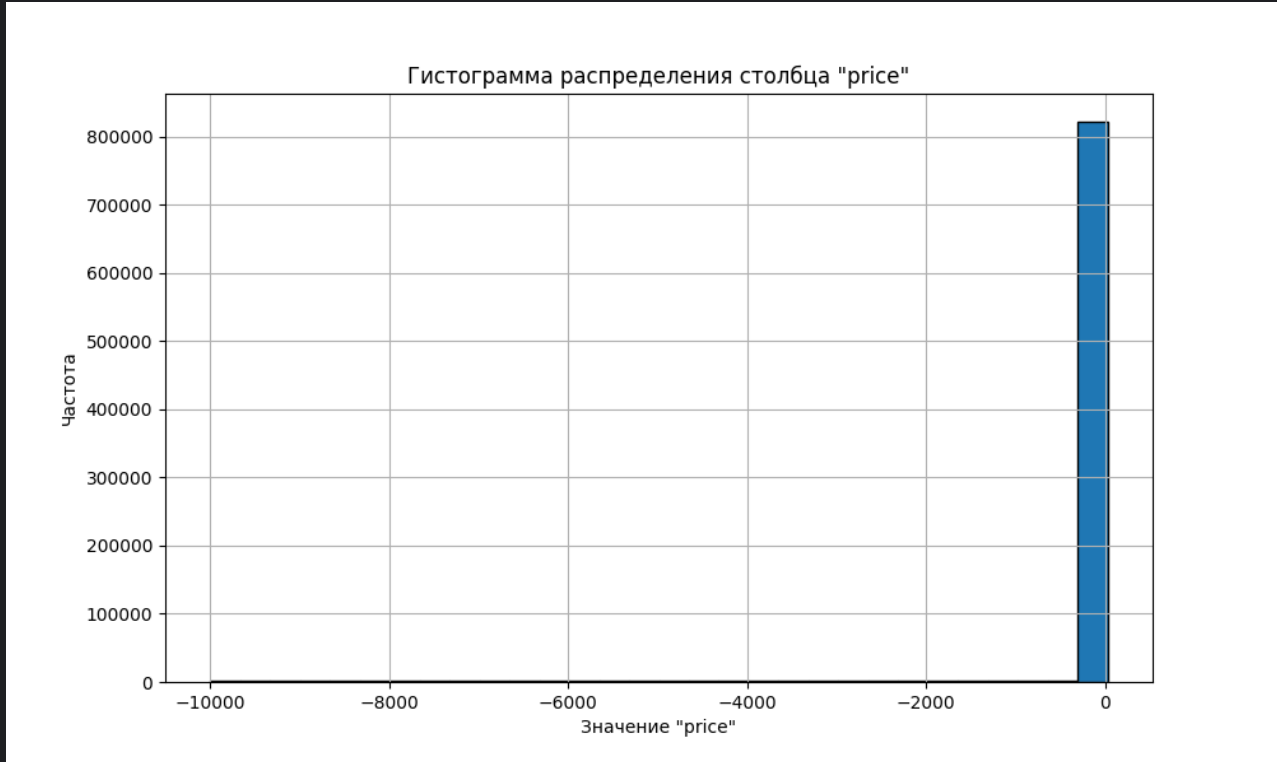
После первого прочтения задачи было много мыслей как это решать. Сначала были следующие вопросы: как максимизировать прибыль, не зная спроса и предложения, зачем нам дан файл с себестоимостью продукта, хотя нам надо спрогнозировать на 90 следующих дней. В итоге я пришел в следующим идеям

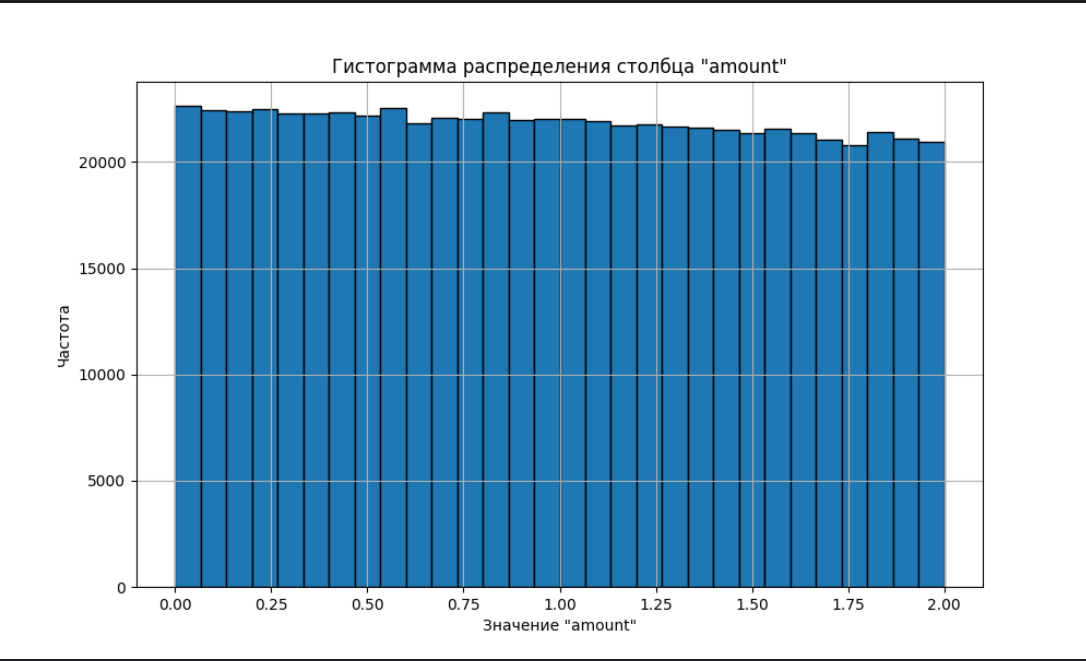
Насколько я понял, суть задачи предсказать изменения цены товара отдельного конкурента в отдельном городе. Я начал думать, как это разрабатывать.

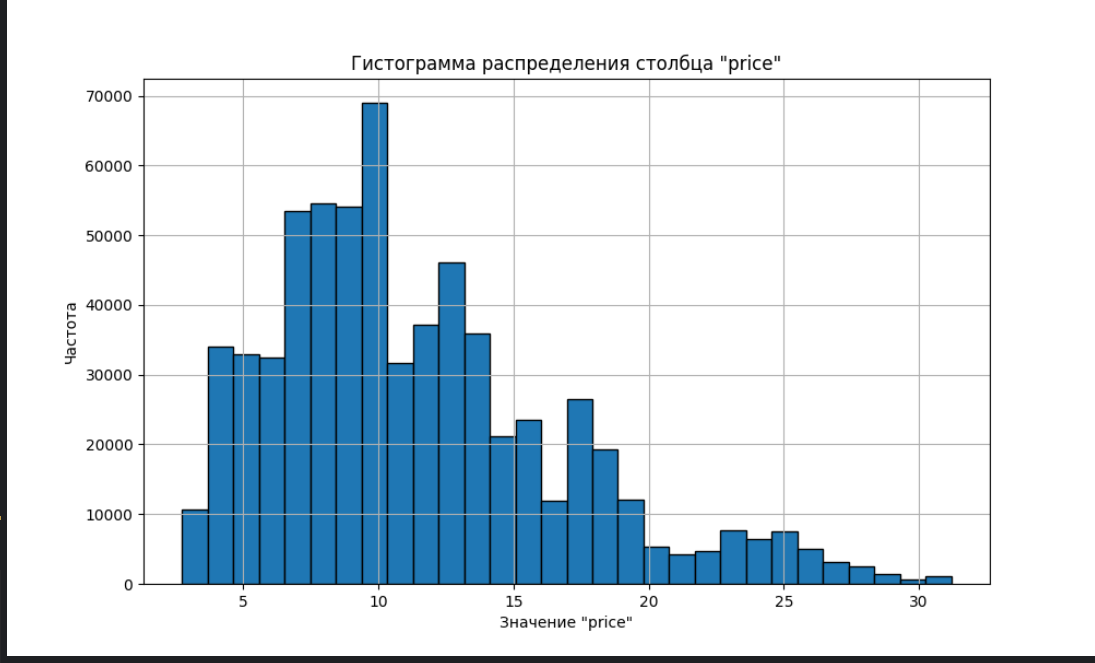
Поначалу брались дефолтные модели – линейная регрессия, полином, градиентный бустинг, случайный лес, но все не показывали нужного результаты в независимости от обработки данных. В итоге я решил, что это задача deep лернинга и начал использовать нейронные сети. Начала с РНН моделей, и уже было очевидно, что эти модели показывают себя намного лучше, банально потому, что они дольше обучаются и видят связи, что не видели вышеуказанные модели. Лучшие результаты теста показала ЛСТМ, поэтому в последствии я решил использовать ее.

Сначала я решил заняться изучением аномалий и увидел большое их количество в файле transaction\_df  




После исправления графи стали выглядеть следующим образом



  
  
за основу удалений аномалий бралось очевидное – цены и количество не должно быть меньше 0 и все же добавил пороговое значение z-теста, чтобы отсечь явно выбивающиеся данные. Но по итогу мне это абсолютно не пригодилось, так как данные датафрейм вообще не использовал в предикте.  
  
Далее идет основная часть – сама модель ЛСТМ

На этапе обучения я наблюдал большое количество проблем: сначала я пытался обучать модель на всех данных, но это оказалось большой ошибкой. Далее я попробовал обучать на каждой составляющей, то есть для каждого товара каждого конкурента в каждом городе, но данных были слишком мало, чтобы составить правильные корреляции, поэтому я остановился на следующем варианте

for place in places:  
 for product in products:  
 print(f"Predicting for: Place={place}, Product={product}")  
 # Фильтрация данных для конкретного конкурента, города и товара  
 data\_filter = data\_with\_weather[  
 (data\_with\_weather['place'] == place) &  
 (data\_with\_weather['product'] == product)]

Обучение шло в зависимости от товара и города, но конкуренты не участвовали, что дало расширить диапазон значений для обучения, но не сделать его слишком непредсказуемым

Следующая проблема, с которой я столкнулся, что модель не видела основной закономерности – возрастания цены, поэтому были введены следующие метрики:

Скользящее среднее – для наблюдения среднего за последние n значений

Look back – цена за прошлое количество дней

look\_back = 10  
for i in range(1, look\_back + 1):  
 data\_with\_weather[f'price\_lag\_{i}'] = data\_with\_weather.groupby(['competitor', 'product'])['price'].shift(i)  
  
# Добавление скользящих средних за последние 25, 50, 100 дней  
data\_with\_weather['rolling\_mean\_25'] = data\_with\_weather.groupby(['competitor', 'product'])['price'].shift(1).rolling(  
 window=25).mean()  
data\_with\_weather['rolling\_mean\_50'] = data\_with\_weather.groupby(['competitor', 'product'])['price'].shift(1).rolling(  
 window=50).mean()

Это позволило модели увидеть основную тенденцию возрастания.

Третей проблемой стала точность предиктов – ее решил при помощи скейлера и хорошей обработке исходных данных.

Четвертая проблема – подбор гиперпараметров нейронной сети. Муторная задача оказалась, но в итоге лучшими значениями оказались: epochs=25, batch\_size=64

И сама модель:

model = Sequential()  
model.add(  
 LSTM(240, input\_shape=(X\_train\_scaled.shape[1], X\_train\_scaled.shape[2]), return\_sequences=True))  
model.add(LSTM(120, return\_sequences=True))  
model.add(LSTM(60))  
model.add(Dense(1))  
model.compile(loss='mean\_squared\_error', optimizer='adam')

При этих значениях не было переобучения, а обучение было порядка 99.4 процентов тестовых данных

Далее осталось понять как предсказывать значения не имея явной информации по конкуренту. Решил ее взятием данных по конкуренту с соответствующим лук бэком.

X\_pred = X\_test\_scaled[-1].reshape(1, X\_test\_scaled.shape[1], 1)  
for competitor in competitors:  
 # Найти последние доступные цены для текущего конкурента  
 last\_prices = data\_filter[data\_filter['competitor'] == competitor]['price'].values[-look\_back:]  
  
 # Использовать последние цены для начала прогнозирования  
 X\_pred[:, :, 0] = scaler\_X.transform(np.array([[  
 data\_filter['price\_trend'].values[-1],  
 data\_filter['rolling\_mean\_25'].values[-1],  
 data\_filter['rolling\_mean\_50'].values[-1],  
 data\_filter['hot'].values[-1],  
 data\_filter['rain'].values[-1],  
 data\_filter['snow'].values[-1],  
 \*last\_prices  
 ]]))

Далее оставалось лишь использовать pivot table, чтобы придать таблице нужный вид и применить наши правила.

В задаче не сказано, должна ли отличаться цена от меньшей цены среди конкурентов или большей, но, проанализировав данные, стало понятно, что все-таки смотрятся по большему значению(transaction\_df все-таки пригодилась). Я придумал следующий алгоритм - берем блоки по 3 строки, в каждой строке находим максимум, выбираем максимум с наименьшим значением и репитим его на три строки. Максимальное rmse, что было у меня при обучении модели = 1.5, поэтому наше значением мы умножим на 1.18 (с запасом)

Осталось две нерешённые проблемы – город Нокрон не обучаем, какие я бы параметры не пробовал, ничего не выходит. Вторая проблема это сравнение с соседних значений на разницы в +-1 и последующее изменение значения. Много пробовал вариантов:

# for i in range(1, len(result\_df)):  
# # Проверка условия: разница больше чем 1 и номер строки кратен 30  
# if abs(result\_df['my\_price'].iloc[i] - result\_df['my\_price'].iloc[i-1]) > 1 and (i) % 30 == 0:  
# # Ничего не делаем, так как условие выполняется  
# pass  
# else:  
# # Проверка разницы больше чем 1 в блоке else  
# if abs(result\_df['my\_price'].iloc[i] - result\_df['my\_price'].iloc[i-1]) > 1:  
# # Изменяем значение на my\_price[i] = my\_price[i-1] + 1 или -1  
# result\_df['my\_price'].iloc[i] = result\_df['my\_price'].iloc[i-1] + 1 if result\_df['my\_price'].iloc[i] > result\_df['my\_price'].iloc[i-1] else result\_df['my\_price'].iloc[i-1] - 1  
#

Это был основное вариант, но он требует доработки

Для города Нокрон я пытался переобучить модель, использовать другие модели, но все равно лстм показала лучшие результаты, хотя по сравнению с другими городами они были не так хороши

Но на доработку двух названных задач уже не хватило сил, маны и времени(