Imports and connections

В данном разделе импортируем все библиотеки которые будем использовать, а также обычно подключаемся

Dataset возьмем с Kaggle. К Kaggle можно подключаться через api и эффективно скачивать датасеты, однако тут мы делать этого не будем

Tutorial:https://saturncloud.io/blog/how-to-import-kaggle-datasets-into-jupyter-notebook/

In [182]:
```
from catboost import CatBoostRegressor
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error
from datetime import datetime

import pandas as pd
import numpy as np
import re
```

Data load

https://www.kaggle.com/datasets/felixzhao/productdemandforecasting - ссылка на датасет

Скачиваем csv и считываем при помощи pandas

In [147]:
```
name_dataset_file = 'Historical Product Demand.csv'

df = pd.read_csv(name_dataset_file)
df.head(5)
```

Out[147]:

	Product_Code	Warehouse	Product_Category	Date	Order_Demand
0	Product_0993	Whse_J	Category_028	2012/7/27	100
1	Product_0979	Whse_J	Category_028	2012/1/19	500
2	Product_0979	Whse_J	Category_028	2012/2/3	500
3	Product_0979	Whse_J	Category_028	2012/2/9	500
4	Product_0979	Whse_J	Category_028	2012/3/2	500

DA

Посмотрим сколько у нас уникальных товаров, точек продажи, категорий товаров

In [149]:
```
df['Product_Code'].unique().shape[0]
```

Out[149]:2160

In [150]:
```
df['Warehouse'].unique().shape[0]
```

Out[150]:4

In [151]:
```
df['Product_Category'].unique().shape[0]
```

Out[151]:33

Почистим date

In [152]:
```
df['Date'].isna().sum() / df['Date'].shape[0]
```

Out[152]:0.010718355863910546

In [153]:
```
df = df.dropna(subset = ['Date'])
```

In [154]:
```
df.isna().sum()
```

Out[154]:
```
Product_Code        0
Warehouse           0
Product_Category    0
Date                0
Order_Demand        0
dtype: int64
```

Тип данных object в большинстве своем подразумевает строку Однако target у нас это количество товара в заказе и оно точно не строка. Конвертируем

In [155]:
```
df.dtypes
```

Out[155]:
```
Product_Code        object
Warehouse           object
Product_Category    object
Date                object
Order_Demand        object
dtype: object
```

Прежде чем использовать ниженаписанную функцию попробуйте просто использовать astype(int) и разберитесь в чем тут проблема и зачем тут эта функция

In [156]:
```
# def convert_target(x):
#     try:
#         return int(x)
#     except:
#         return int(re.sub("[^0-9]", "", x))

#df['Order_Demand'] = df['Order_Demand'].apply(lambda x: convert_target(x))
#astype(float)

df['Order_Demand'] = df['Order_Demand'].astype(str)
```

Посмотрим что там у нас с временным промежутком в данных

In [157]:
```
df['Date'] = pd.to_datetime(df['Date']).astype(object)
print(df['Date'].min())
print(df['Date'].max())
```

```
2011-01-08 00:00:00
2017-01-09 00:00:00
```

Feature engineering

Из 4 представленных признаков мало что можно понять

Немного подумав можно понять что количество тех или иных заказываемых товаров могло изменяться от года к году. А также от сезона.

Т.к. модель не поймет datetime если вы его ей отдадите в виде datetime, займемся feature engineering и построим несколько новых фичей

Достанем год и месяц

In [158]:
```
df['year'] = df['Date'].apply(lambda x: x.year)
```

In [159]:
```
df['month'] = df['Date'].apply(lambda x: x.month)
```

Создадим сезон

In [160]:
```
def get_season(x):
    seasons = {1:'winter',
               2:'winter',
```

```
                  3:'spring',
                  4:'spring',
                  5:'spring',
                  6:'summer',
                  7:'summer',
                  8:'summer',
                  9:'fall',
                 10:'fall',
                 11:'fall',
                 12:'winter'}
    return seasons[x]

df['season'] = df['month'].apply(get_season)
df
```

Out[160]:

	Product_Code	Warehouse	Product_Category	Date	Order_Demand	year	month	season
0	Product_0993	Whse_J	Category_028	2012-07-27 00:00:00	100	2012	7	summer
1	Product_0979	Whse_J	Category_028	2012-01-19 00:00:00	500	2012	1	winter
2	Product_0979	Whse_J	Category_028	2012-02-03 00:00:00	500	2012	2	winter
3	Product_0979	Whse_J	Category_028	2012-02-09 00:00:00	500	2012	2	winter
4	Product_0979	Whse_J	Category_028	2012-03-02 00:00:00	500	2012	3	spring
...	...	...	...	...	...	...	...	...
1048570	Product_1791	Whse_J	Category_006	2016-04-27 00:00:00	1000	2016	4	spring
1048571	Product_1974	Whse_J	Category_006	2016-04-27 00:00:00	1	2016	4	spring
1048572	Product_1787	Whse_J	Category_006	2016-04-28 00:00:00	2500	2016	4	spring
1048573	Product_0901	Whse_J	Category_023	2016-10-07 00:00:00	50	2016	10	fall
1048574	Product_0704	Whse_J	Category_001	2016-06-27 00:00:00	4	2016	6	summer

1037336 rows × 8 columns

Достанем

день с начала месяца

день с начала года

и конвертируем дату в числовой тип

In [161]:
```
df['day_from_month'] = df['Date'].apply(lambda x: x.day)
df['day_from_year'] = df['Date'].apply(lambda x: x.dayofyear)
df['date_int_convert'] = df['Date'].apply(lambda x: int(x.timestamp()))
```

Однако наши дни в месяце а также дни в году цикличны

А мы с Вами знаем прекрасные математические функции которые имеют такую же цикличность

Сделаем цикличность некоторых временных признаков

Вопрос на погуглить: а зачем это делают?

In [162]:
```
df['sin_day_month'] = np.sin(df['day_from_month']*(2.*np.pi/30))
df['sin_day_year'] = np.sin(df['day_from_month']*(2.*np.pi/365))
```

In [163]:
```
df
```

Out[163]:

	Product_Code	Warehouse	Product_Category	Date	Order_Demand	year	month	season	day_from_mo
0	Product_0993	Whse_J	Category_028	2012-07-27 00:00:00	100	2012	7	summer	
1	Product_0979	Whse_J	Category_028	2012-01-19 00:00:00	500	2012	1	winter	
2	Product_0979	Whse_J	Category_028	2012-02-03 00:00:00	500	2012	2	winter	
3	Product_0979	Whse_J	Category_028	2012-02-09 00:00:00	500	2012	2	winter	
4	Product_0979	Whse_J	Category_028	2012-03-02 00:00:00	500	2012	3	spring	
...	...	...	...	...	...	...	...	...	
1048570	Product_1791	Whse_J	Category_006	2016-04-27 00:00:00	1000	2016	4	spring	
1048571	Product_1974	Whse_J	Category_006	2016-04-27 00:00:00	1	2016	4	spring	
1048572	Product_1787	Whse_J	Category_006	2016-04-28 00:00:00	2500	2016	4	spring	
1048573	Product_0901	Whse_J	Category_023	2016-10-07 00:00:00	50	2016	10	fall	
1048574	Product_0704	Whse_J	Category_001	2016-06-27 00:00:00	4	2016	6	summer	

1037336 rows × 13 columns

Catboost

Закончив вертеть данные перейдем наконец-то к модели

Название CatBoost произошло (нет не от котиков) а от Cat-Categorical. Данная библиотека позволяет конвертировать и обрабатывать самостоятельно categorical features что является ее основным преимуществом

Создадим три списка,

1. Список категориальных фичей
2. Список все фичей которые будут задействованы в обучении

3. Список target-ов (то что мы предсказываем)

In [164]:
```
CAT_FEATURES = ['Product_Code',
                'Warehouse',
                'Product_Category',
                'season',
                'month',
                'year']

FEATURES = ['Product_Code',
                'Warehouse',
                'Product_Category',
                'season',
                'month',
                'year',
                'sin_day_month',
                'sin_day_year',
                'date_int_convert']


TARGET = ['Order_Demand']
```

Произведем разбиение на обучающую и тестовую выборку при помощи метода train test split

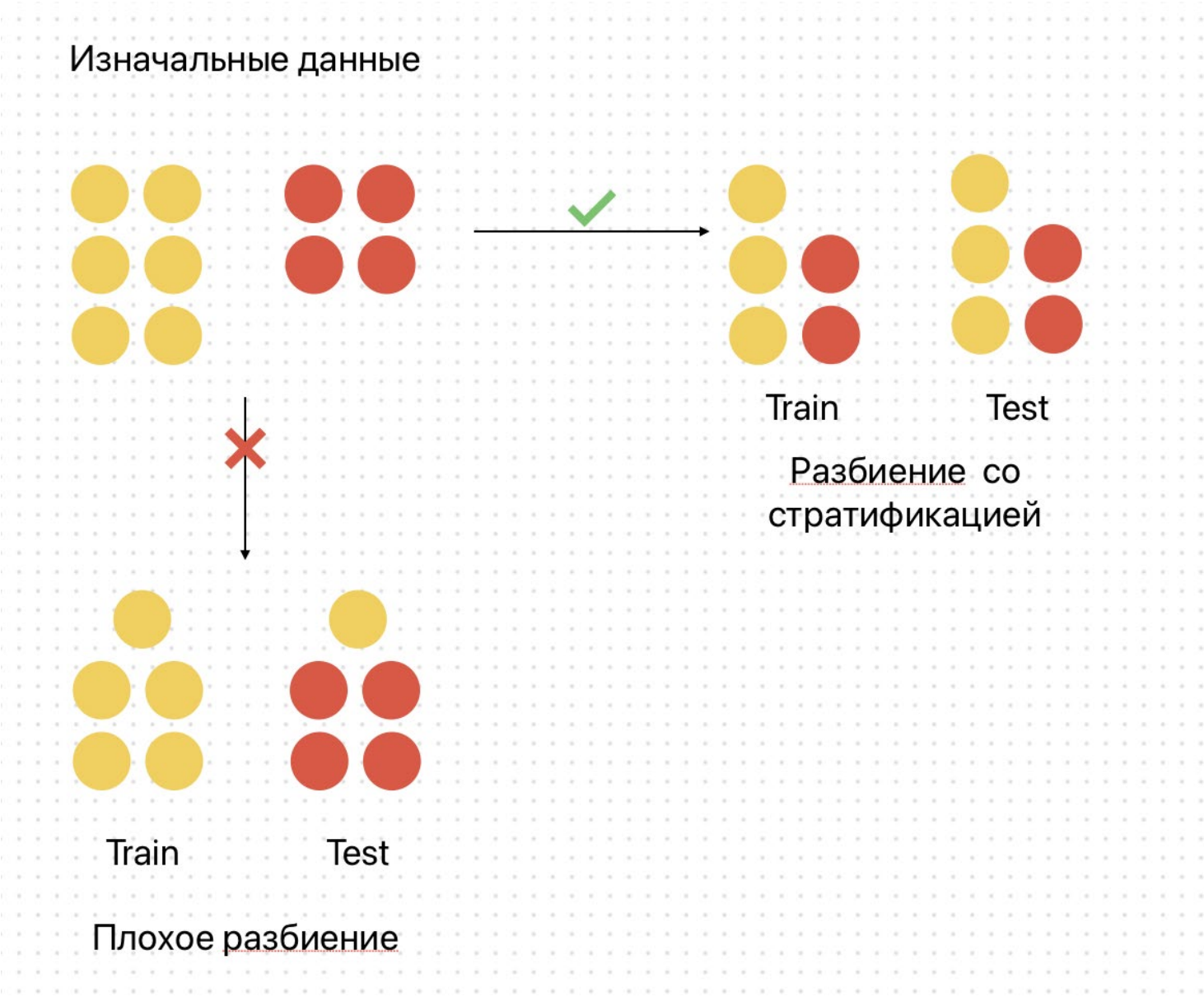
test_size - параметр задающий долю значений попадающую в test часть. Если она 0.33 то в train соответственно попадет 77 оставшихся

Но важнее здесь другое: а именно **стратификация**

Процедура стратификации позволяет разделить наши данные так, чтобы и в обучающую и в тестовую выборку данные попали распределившись согласно test_size по стратифицирующей колонке.

(Представим себе ситуацию что у нас есть 4 красных шарика и 6 желтых). Всего 10 шариков. И мы хотим чтобы наши шарики распределились в соотношении 50 / 50. Стратификация по цвету в данном случае разобьет каждую группу цвета в соотношении 50/50 и предотвратит ситуацию когда в одну из выборок попадут шары одного цвета.

Вопрос на погуглить: а что делают с непрерывными признаками?



Подготовимся к стратификации и создадим колонку из года + месяца

In [165]:
```
df['stratify_column'] = df['year'].astype(str) + '-' + df['month'].astype(str)
```

Разберитесь с ошибкой

In [166]:
```
X_train, X_test, y_train, y_test = train_test_split(df[FEATURES],
                                                    df[TARGET],
                                                    test_size=0.33,
                                                    random_state=42,
                                                    stratify =
      df['stratify_column'])
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Cell In[166], line 1
----> 1 X_train, X_test, y_train, y_test = train_test_split(df[FEATURES],
      2                                                     df[TARGET],
      3                                                     test_size=0.33,
      4                                                     random_state=42,
      5                                                     stratify = df['stratify_colum
n'])

File ~/anaconda3/lib/python3.11/site-packages/sklearn/utils/_param_validation.py:211, in
validate_params.<locals>.decorator.<locals>.wrapper(*args, **kwargs)
    205 try:
```

```
    206     with config_context(
    207         skip_parameter_validation=(
    208             prefer_skip_nested_validation or global_skip_validation
    209         )
    210     ):
--> 211         return func(*args, **kwargs)
    212 except InvalidParameterError as e:
    213     # When the function is just a wrapper around an estimator, we allow
    214     # the function to delegate validation to the estimator, but we replace
    215     # the name of the estimator by the name of the function in the error
    216     # message to avoid confusion.
    217     msg = re.sub(
    218         r"parameter of \w+ must be",
    219         f"parameter of {func.__qualname__} must be",
    220         str(e),
    221     )

File ~/anaconda3/lib/python3.11/site-packages/sklearn/model_selection/_split.py:2638, in train_test_split(test_size, train_size, random_state, shuffle, stratify, *arrays)
   2634         CVClass = ShuffleSplit
   2636     cv = CVClass(test_size=n_test, train_size=n_train, random_state=random_state)
-> 2638     train, test = next(cv.split(X=arrays[0], y=stratify))
   2640 return list(
   2641     chain.from_iterable(
   2642         (_safe_indexing(a, train), _safe_indexing(a, test)) for a in arrays
   2643     )
   2644 )

File ~/anaconda3/lib/python3.11/site-packages/sklearn/model_selection/_split.py:1726, in BaseShuffleSplit.split(self, X, y, groups)
   1696 """Generate indices to split data into training and test set.
   1697
   1698 Parameters
   (...)
   1723 to an integer.
   1724 """
   1725 X, y, groups = indexable(X, y, groups)
-> 1726 for train, test in self._iter_indices(X, y, groups):
   1727     yield train, test

File ~/anaconda3/lib/python3.11/site-packages/sklearn/model_selection/_split.py:2115, in StratifiedShuffleSplit._iter_indices(self, X, y, groups)
   2113 class_counts = np.bincount(y_indices)
   2114 if np.min(class_counts) < 2:
-> 2115     raise ValueError(
   2116         "The least populated class in y has only 1"
   2117         " member, which is too few. The minimum"
   2118         " number of groups for any class cannot"
   2119         " be less than 2."
   2120     )
   2122 if n_train < n_classes:
   2123     raise ValueError(
   2124         "The train_size = %d should be greater or "
   2125         "equal to the number of classes = %d" % (n_train, n_classes)
   2126     )

ValueError: The least populated class in y has only 1 member, which is too few. The minimum number of groups for any class cannot be less than 2.
```

Решение ошибки:

In [167]:
```
check_count_stratify = df['stratify_column'].value_counts().to_dict()
```

```
check_count_stratify
```

Out[167]:
```
{'2013-10': 24546,
 '2014-3': 19839,
 '2014-10': 19579,
 '2015-7': 19127,
 '2015-3': 19122,
 '2013-7': 19085,
 '2014-9': 18970,
 '2013-9': 18946,
 '2014-7': 18867,
 '2013-11': 18617,
 '2012-3': 18604,
 '2012-10': 18515,
 '2015-6': 18413,
 '2015-1': 18245,
 '2014-2': 18214,
 '2012-2': 18123,
 '2015-2': 18116,
 '2014-4': 18077,
 '2014-1': 18013,
 '2013-5': 17786,
 '2015-10': 17785,
 '2014-6': 17762,
 '2012-7': 17718,
 '2013-4': 17685,
 '2015-4': 17603,
 '2014-11': 17486,
 '2012-11': 17416,
 '2013-12': 17398,
 '2013-3': 17397,
 '2016-11': 17335,
 '2015-11': 17335,
 '2016-3': 17282,
 '2013-2': 17119,
 '2012-5': 17014,
 '2014-12': 16937,
 '2012-6': 16911,
 '2013-8': 16660,
 '2015-9': 16656,
 '2014-5': 16639,
 '2013-1': 16638,
 '2012-4': 16590,
 '2012-8': 16492,
 '2013-6': 16421,
 '2016-6': 16418,
 '2016-10': 16191,
 '2015-12': 16180,
 '2016-2': 16130,
 '2014-8': 16021,
 '2012-1': 15614,
 '2012-9': 15613,
 '2015-5': 15572,
 '2015-8': 15507,
 '2016-9': 15376,
 '2016-8': 15333,
 '2016-7': 15319,
 '2016-4': 15223,
 '2016-12': 15036,
 '2012-12': 15025,
 '2016-1': 14515,
```

```
 '2016-5': 14487,
 '2011-12': 598,
 '2017-1': 53,
 '2011-11': 31,
 '2011-9': 4,
 '2011-10': 3,
 '2011-6': 2,
 '2011-1': 1,
 '2011-5': 1}
```

Обьединим сезоны в которых мало записей в "другие"

In [168]:
```
df['stratify_column'] = df['stratify_column'].apply(lambda x: 'other' if
check_count_stratify[x] < 5 else x)
```

Теперь разобьем наши данные на train и validation

Test же возьмем из последних лет (а именно тех данных за последние годы (Ведь мы хотим предсказывать будущее а не прошлое)

In [175]:
```
X_train, X_eval, y_train, y_eval = train_test_split(df[df['year'] < 2016]
[FEATURES],
                                                    df[df['year'] < 2016][TARGET],
                                                    test_size=0.33,
                                                    random_state=42,
                                                    stratify = df[df['year'] <
2016]['stratify_column'])
```

In [187]:
```
test = df[df['year'] >= 2016]
X_test = test[FEATURES]
y_test = test[TARGET]
```

При объявлении модели зададим несколько параметров (их называют гиперпараметры)

Почитать за что они отвечают можно в офиц. документации

https://catboost.ai/en/docs/concepts/python-reference_catboostregressor

https://catboost.ai/en/docs/concepts/python-reference_catboostregressor_fit#eval_set

In [180]:
```
model = CatBoostRegressor(iterations=10000,
                          learning_rate=1**(-2),
                          loss_function='RMSE',
                          cat_features=CAT_FEATURES,
                          depth=4)
# Fit model
model.fit(X_train,
          y_train,
          eval_set = (X_eval, y_eval),
          early_stopping_rounds=1000,
          verbose=100,
          plot=True)

# Get predictions
preds = model.predict(X_test)
```

```
MetricVisualizer(layout=Layout(align_self='stretch', height='500px'))
0:      learn: 27889.8588917    test: 26873.3253298     best: 26873.3253298 (0) total: 50
.6ms    remaining: 8m 26s
100:    learn: 27295.6888425    test: 26389.2662688     best: 26379.8940997 (97)        t
otal: 3.7s      remaining: 6m 2s
200:    learn: 27078.7702388    test: 26312.7477841     best: 26312.7477841 (200)       t
otal: 7.59s     remaining: 6m 9s
300:    learn: 26951.4304776    test: 26301.9382215     best: 26281.7882486 (253)       t
otal: 11.7s     remaining: 6m 16s
```

```
400:    learn: 26856.3941540    test: 26286.0286200     best: 26275.7632406 (383)       t
otal: 16s       remaining: 6m 21s
500:    learn: 26747.9101838    test: 26271.1590427     best: 26253.3902214 (490)       t
otal: 20.3s     remaining: 6m 25s
600:    learn: 26600.5398207    test: 26279.2625116     best: 26253.3902214 (490)       t
otal: 24.6s     remaining: 6m 25s
700:    learn: 26517.4586232    test: 26297.1657548     best: 26253.3902214 (490)       t
otal: 28.8s     remaining: 6m 22s
800:    learn: 26424.0774583    test: 26287.6329709     best: 26253.3902214 (490)       t
otal: 33.2s     remaining: 6m 21s
900:    learn: 26319.8453762    test: 26387.9941211     best: 26253.3902214 (490)       t
otal: 37.3s     remaining: 6m 16s
1000:   learn: 26252.0674315    test: 26401.6694172     best: 26253.3902214 (490)       t
otal: 41.8s     remaining: 6m 15s
1100:   learn: 26196.8964445    test: 26357.8361294     best: 26253.3902214 (490)       t
otal: 45.8s     remaining: 6m 9s
1200:   learn: 26150.8353400    test: 26356.7209917     best: 26253.3902214 (490)       t
otal: 50s       remaining: 6m 6s
1300:   learn: 26110.4107961    test: 26367.1391040     best: 26253.3902214 (490)       t
otal: 54.2s     remaining: 6m 2s
1400:   learn: 26076.0492253    test: 26378.0481770     best: 26253.3902214 (490)       t
otal: 58.7s     remaining: 6m
Stopped by overfitting detector  (1000 iterations wait)

bestTest = 26253.39022
bestIteration = 490

Shrink model to first 491 iterations.
```

In [188]:
```
test['pred'] = model.predict(X_test)
test
```

```
/var/folders/nf/rd32kdyn0j9fz___z5xdz_q40000gn/T/ipykernel_11446/1552603700.py:1: Setting
WithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_g
uide/indexing.html#returning-a-view-versus-a-copy
  test['pred'] = model.predict(X_test)
```

Out[188]:

	Product_Code	Warehouse	Product_Category	Date	Order_Demand	year	month	season	day_from_mo
690943	Product_1178	Whse_A	Category_024	2016-01-28 00:00:00	10	2016	1	winter	
699219	Product_1502	Whse_C	Category_019	2016-01-04 00:00:00	100000	2016	1	winter	
768552	Product_0190	Whse_A	Category_007	2016-01-06 00:00:00	320	2016	1	winter	
768635	Product_0337	Whse_A	Category_021	2016-01-06 00:00:00	2	2016	1	winter	
768656	Product_1053	Whse_A	Category_024	2016-01-06 00:00:00	10	2016	1	winter	
...	...	...	...	...	...	...	...	...	
				2016-					

1048570	Product_1791	Whse_J	Category_006	04-27 00:00:00	1000	2016	4	spring
1048571	Product_1974	Whse_J	Category_006	2016-04-27 00:00:00	1	2016	4	spring
1048572	Product_1787	Whse_J	Category_006	2016-04-28 00:00:00	2500	2016	4	spring
1048573	Product_0901	Whse_J	Category_023	2016-10-07 00:00:00	50	2016	10	fall
1048574	Product_0704	Whse_J	Category_001	2016-06-27 00:00:00	4	2016	6	summer

188698 rows × 15 columns

Ошибка на тестовом множестве

In [190]:
```
mean_squared_error(test['Order_Demand'],test['pred'])**(0.5)
```

Out[190]:71976.4125056439
В результате нашего обучения получили какую-то модель которая может предсказывать нам наши будущие заказы

Попытайтесь ответить на вопросы:

1. Что видно на графике? Что произошло на 890 итерации?
2. Почему мы не добрались до 10.000 итерации?
3. Как было бы лучше задать гиперпараметры учитывая количество категориальных фичей?
4. Попытайтесь оценить насколько результаты модели хороши? Как их интерпретировать?
5. Как можно было улучшить разбиение на трейн и тест чтобы получить лучшие результаты?

Попытайтесь сделать:

1. Напишите код который будет подбирать хорошие гиперпараметры модели (gridsearch, optuna)
2. Постройте лес на этих же данных
3. Постройте другие бустинги на этих же данных
4. Сравните результаты. Какие модели лучше?

spoiler: этой задачей мы еще займемся когда будем заниматься временными рядами