In [1]:

```
import datetime as dt
import os
import gc

import pandas as pd
# import dask.dataframe as pd
import numpy as np
import matplotlib.pyplot as plt

import lightgbm as lgb
from sklearn.model_selection import KFold
```

In [2]:

```
pd.set_option('display.max_rows', 1000)
pd.set_option('display.max_columns', 100)
```

In [3]:

```
INPUT_FOLDER = './data/input'
OUTPUT_FOLDER = './data/output'
```

In [4]:

```
os.listdir(INPUT_FOLDER)
```

Out[4]:

```
['geo_params.csv',
 'sample_final.csv',
 'sku_final.csv',
 'test_data.csv',
 'train_final.csv']
```

Зчитаємо дані

In [5]:

```
dateparser = lambda x: dt.datetime.strptime(x, '%Y-%m-%d')
```

In [6]:

```
df_train = pd.read_csv(os.path.join(INPUT_FOLDER, 'train_final.csv'),
    index_col='ID',
    parse_dates = ['date'],
    date_parser=dateparser)
```

In [7]:

```
df_train['sales'].fillna(0, inplace=True)
```

In [8]:

```
df_train.sort_values('date', inplace=True)
```

In [9]:

```
df_train['price'] = df_train.groupby(['geoCluster', 'SKU'], sort=False)['price'].apply(
```

In [14]:

```
df_test = pd.read_csv(
    os.path.join(INPUT_FOLDER, 'test_data.csv'),
    index_col='ID',
    parse_dates = ['date'],
    date_parser=dateparser
)
```

In [20]:

```
df_train = df_train[df_train['date'] >= '2021-05-01']
```

In [21]:

```
submission = pd.read_csv(os.path.join(INPUT_FOLDER, 'sample_final.csv'), index_col='ID'

submission.head()
```

Out[21]:

	sales
ID	
RR1666030	0
RR1666031	0
RR1666032	0
RR1666033	0
RR1666034	0

In [23]:

```
geo_params = pd.read_csv(os.path.join(INPUT_FOLDER, 'geo_params.csv'))
geo_params.head()
```

Out[23]:

	geoCluster	cityId
0	21	1
1	47	1
2	48	1
3	92	1
4	112	1

In [26]:

```
sku = pd.read_csv(os.path.join(INPUT_FOLDER, 'sku_final.csv'))

sku.head()
```

Out[26]:

	SKU	productCategoryId	productCategory_caption_UKR	productCategory_caption_RU	productCategor
0	17	5416.0	Хурма	Хурма	
1	18	5413.0	Фейхоа	Фейхоа	
2	24	5425.0	Гранат	Гранат	
3	25	5431.0	Апельсин	Апельсин	

	SKU	productCategoryId	productCategory_caption_UKR	productCategory_caption_RU	productCategor
4	208	5835.0	Вода України газована	Вода Украины газированная	Water, Uk

Data preprocessing

In [27]:

```
def merge_data(df):
    df = df.copy(deep=True)
    df = df.merge(sku, on = 'SKU', how='left')
    df = df.merge(geo_params, on = 'geoCluster', how='left')
    return df

def preprocess_date(df):
    df['month'] = df['date'].dt.month
#     df['weekofyear'] = df['date'].dt.weekofyear
    df['day'] = df['date'].dt.day
    df['dayofweek'] = df['date'].dt.dayofweek
#     df['dayofyear'] = df['date'].dt.dayofyear
    return df
```

In [28]:

```
df_train = merge_data(df_train)
```

In [29]:

```
df_train = preprocess_date(df_train)
```

In [30]:

```
df_train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 8514733 entries, 0 to 8514732
Data columns (total 28 columns):
 #   Column                       Dtype
---  ------                       -----
 0   geoCluster                   int64
 1   SKU                          int64
 2   date                         datetime64[ns]
 3   price                        float64
 4   sales                        float64
 5   productCategoryId            float64
 6   productCategory_caption_UKR  object
 7   productCategory_caption_RU   object
 8   productCategory_caption_ENG  object
 9   productTypeId                float64
 10  productType_caption_UKR      object
 11  productType_caption_RU       object
 12  productType_caption_ENG      object
 13  brandId                      float64
 14  lagerUnitQuantity            float64
 15  lagerUnitTypeId              int64
 16  lagerUnitType_caption        object
 17  trademark                    float64
 18  countryOfOrigin              float64
 19  countryOfOrigin_caption      object
 20  commodity_group              int64
 21  commodity_group_caption_UKR  object
 22  commodity_group_caption_RU   object
 23  commodity_group_caption_ENG  object
 24  cityId                       int64
 25  month                        int64
 26  day                          int64
 27  dayofweek                    int64
dtypes: datetime64[ns](1), float64(8), int64(8), object(11)
memory usage: 1.8+ GB
```

In [31]:

```
# df_train.sample(15)
```

In [32]:

```
cat_cols = [
    'geoCluster', 'SKU', 'productCategoryId', 'productTypeId', 'brandId', 'lagerUnitTyp
    'trademark', 'countryOfOrigin', 'commodity_group', 'cityId',
    'month',
#     'weekofyear',
    'day',
    'dayofweek',
#     'dayofyear'
]
num_cols = [
    'price',
    'lagerUnitQuantity',
#     'price_change'

]
target = 'sales'
info_cols = ['date']
drop_cols = ['productCategory_caption_UKR', 'productCategory_caption_RU', 'productCateg
    'productType_caption_UKR', 'productType_caption_RU', 'productType_caption_ENG', 'la
    'commodity_group_caption_UKR', 'commodity_group_caption_RU', 'commodity_group_capti

def categorize_columns(df):
    df = df.copy(deep=True)
    df.drop(drop_cols, axis=1, inplace=True)
    df[cat_cols] = df[cat_cols].astype('category')
    return df
```

In [33]:

```
df_train = categorize_columns(df_train)
```

In [34]:

```
X, y = df_train.drop(target, axis=1).copy(), df_train[target].copy()
```

In [35]:

```
del df_train
gc.collect()
```

Out[35]:

```
57
```

In [36]:

```
sku_float_sales = X[y - y.astype(int) != 0]['SKU'].unique()
```

Model training

In [37]:

```
params = {
    'random_state':42,
    'n_estimators':500,
    'objective': 'regression',
    'metric': 'mae',
    "early_stopping_rounds":50,
    'learning_rate':0.1,
    'subsample':0.8,
    'subsample_freq':10,
    'feature_fraction':0.5
}
```

In [38]:

```
def custom_mae(y_true, y_pred, dates):
    df = pd.DataFrame(data={
        'y_true': np.asarray(y_true),
        'y_pred': np.asarray(y_pred),
        'date': np.asarray(dates)})
    df['error'] = np.abs(df['y_pred'] - df['y_true'])
    df_sum = df.groupby('date').sum().reset_index()
    df_sum.dropna(subset=['error', 'y_true'], how='any', inplace=True)
    df_sum = df_sum[df_sum['y_true'] > 0]
    if df_sum.shape[0] != df['date'].drop_duplicates().shape[0]:
        print(f"{df['date'].drop_duplicates().shape[0] - df_sum.shape[0]} rows were dro
    return (df_sum['error'] / df_sum['y_true']).mean()
```

In [39]:

```
n_splits = 5
folds = np.array_split(X['date'].drop_duplicates().values, n_splits)
folds = [(fold[0], fold[-1]) for fold in folds]
for i, fold in enumerate(folds):
    print(f'Fold {i}: {fold}')
```

```
Fold 0: (numpy.datetime64('2021-05-01T00:00:00.000000000'), numpy.datetime64('2021-05-16
T00:00:00.000000000'))
Fold 1: (numpy.datetime64('2021-05-17T00:00:00.000000000'), numpy.datetime64('2021-06-01
T00:00:00.000000000'))
Fold 2: (numpy.datetime64('2021-06-02T00:00:00.000000000'), numpy.datetime64('2021-06-17
T00:00:00.000000000'))
Fold 3: (numpy.datetime64('2021-06-18T00:00:00.000000000'), numpy.datetime64('2021-07-03
T00:00:00.000000000'))
Fold 4: (numpy.datetime64('2021-07-04T00:00:00.000000000'), numpy.datetime64('2021-07-19
T00:00:00.000000000'))
```

In [40]:

```
models = []

train_mae_scores = []
val_mae_scores = []
```

In [41]:

```
for fold, fold_dates in enumerate(folds):

    print(f'\nFold {fold+1}')

    start_date, end_date = pd.Timestamp(fold_dates[0]), pd.Timestamp(fold_dates[1])
    print(start_date, end_date)

    train_mask = (X['date'] < start_date) | (X['date']>end_date)
    val_mask = (X['date'] >= start_date) & (X['date'] <= end_date)


    X_train, y_train = X.loc[train_mask], y.loc[train_mask]
    X_val, y_val = X.loc[val_mask], y.loc[val_mask]

    train_dates, val_dates = X_train['date'], X_val['date']

    X_train, X_val = X_train.drop('date', axis=1), X_val.drop('date', axis=1)

    lgb_reg = lgb.LGBMRegressor(**params)
    lgb_reg.fit(X_train, y_train, eval_set=[(X_train, y_train), (X_val, y_val)], verbos

    y_train_pred = lgb_reg.predict(X_train)
    y_val_pred = lgb_reg.predict(X_val)

#     y_train_pred = np.where(X_train['SKU'].isin(sku_float_sales), y_train_pred, np.ro
#     y_val_pred = np.where(X_val['SKU'].isin(sku_float_sales), y_val_pred, np.round(y_

    train_mae_scores.append(custom_mae(y_train, np.round(y_train_pred), train_dates))
    val_mae_scores.append(custom_mae(y_val, np.round(y_val_pred), val_dates))

    print('Train mean daily MAE:', train_mae_scores[-1])
    print('Valid mean daily MAE:', val_mae_scores[-1])

    models.append(lgb_reg)
```

```
Fold 1
2021-05-01 00:00:00 2021-05-16 00:00:00
/home/eddie/miniconda3/envs/hack4retail/lib/python3.9/site-packages/lightgbm/sklearn.py:
736: UserWarning: 'verbose' argument is deprecated and will be removed in a future relea
se of LightGBM. Pass 'log_evaluation()' callback via 'callbacks' argument instead.
  _log_warning("'verbose' argument is deprecated and will be removed in a future release
of LightGBM. "
[LightGBM] [Warning] early_stopping_round is set=50, early_stopping_rounds=50 will be ig
nored. Current value: early_stopping_round=50
[LightGBM] [Warning] feature_fraction is set=0.5, colsample_bytree=1.0 will be ignored.
Current value: feature_fraction=0.5
/home/eddie/miniconda3/envs/hack4retail/lib/python3.9/site-packages/lightgbm/basic.py:17
80: UserWarning: Overriding the parameters from Reference Dataset.
  _log_warning('Overriding the parameters from Reference Dataset.')
/home/eddie/miniconda3/envs/hack4retail/lib/python3.9/site-packages/lightgbm/basic.py:15
13: UserWarning: categorical_column in param dict is overridden.
  _log_warning(f'{cat_alias} in param dict is overridden.')
[50]    training's l1: 0.286592 valid_1's l1: 0.291598
Train mean daily MAE: 1.0370767630681577
Valid mean daily MAE: 1.0568761683609842

Fold 2
2021-05-17 00:00:00 2021-06-01 00:00:00
[LightGBM] [Warning] early_stopping_round is set=50, early_stopping_rounds=50 will be ig
nored. Current value: early_stopping_round=50
[LightGBM] [Warning] feature_fraction is set=0.5, colsample_bytree=1.0 will be ignored.
Current value: feature_fraction=0.5
[50]    training's l1: 0.283891 valid_1's l1: 0.299177
[100]   training's l1: 0.281303 valid_1's l1: 0.299156
Train mean daily MAE: 1.0378611854219386
```

```
Valid mean daily MAE: 1.0314736089208234

Fold 3
2021-06-02 00:00:00 2021-06-17 00:00:00
[LightGBM] [Warning] early_stopping_round is set=50, early_stopping_rounds=50 will be ig
nored. Current value: early_stopping_round=50
[LightGBM] [Warning] feature_fraction is set=0.5, colsample_bytree=1.0 will be ignored.
Current value: feature_fraction=0.5
[50]	training's l1: 0.289658	valid_1's l1: 0.282616
[100]	training's l1: 0.286938	valid_1's l1: 0.279262
[150]	training's l1: 0.284991	valid_1's l1: 0.276907
[200]	training's l1: 0.283635	valid_1's l1: 0.275909
[250]	training's l1: 0.283071	valid_1's l1: 0.275302
[300]	training's l1: 0.282165	valid_1's l1: 0.274934
[350]	training's l1: 0.281606	valid_1's l1: 0.274417
[400]	training's l1: 0.281006	valid_1's l1: 0.273919
[450]	training's l1: 0.280188	valid_1's l1: 0.273133
[500]	training's l1: 0.279418	valid_1's l1: 0.272655
Train mean daily MAE: 1.0211847411949384
Valid mean daily MAE: 1.045528178126395

Fold 4
2021-06-18 00:00:00 2021-07-03 00:00:00
[LightGBM] [Warning] early_stopping_round is set=50, early_stopping_rounds=50 will be ig
nored. Current value: early_stopping_round=50
[LightGBM] [Warning] feature_fraction is set=0.5, colsample_bytree=1.0 will be ignored.
Current value: feature_fraction=0.5
[50]	training's l1: 0.288168	valid_1's l1: 0.292
[100]	training's l1: 0.285024	valid_1's l1: 0.291847
[150]	training's l1: 0.283335	valid_1's l1: 0.291722
Train mean daily MAE: 1.0346770870699171
Valid mean daily MAE: 1.0828506211480438

Fold 5
2021-07-04 00:00:00 2021-07-19 00:00:00
[LightGBM] [Warning] early_stopping_round is set=50, early_stopping_rounds=50 will be ig
nored. Current value: early_stopping_round=50
[LightGBM] [Warning] feature_fraction is set=0.5, colsample_bytree=1.0 will be ignored.
Current value: feature_fraction=0.5
[50]	training's l1: 0.289072	valid_1's l1: 0.293897
[100]	training's l1: 0.285566	valid_1's l1: 0.291556
[150]	training's l1: 0.283666	valid_1's l1: 0.290518
[200]	training's l1: 0.282289	valid_1's l1: 0.289072
[250]	training's l1: 0.281734	valid_1's l1: 0.289713
Train mean daily MAE: 1.0311821878115128
Valid mean daily MAE: 1.097401858715899
```

In [42]:

```
train_mae_scores, val_mae_scores = np.array(train_mae_scores), np.array(val_mae_scores)

print(f'Train score: {train_mae_scores.mean()}+-{train_mae_scores.std()}\n{train_mae_sc
print(f'Val score: {val_mae_scores.mean()}+-{val_mae_scores.std()}\n{val_mae_scores}\n'
```

```
Train score: 1.0323963929132929+-0.00606872006386897
[1.03707676 1.03786119 1.02118474 1.03467709 1.03118219]

Val score: 1.062826087054429+-0.024141512059155802
[1.05687617 1.03147361 1.04552818 1.08285062 1.09740186]
```

In [61]:

```
pd.Series(index=X_train.columns, data=lgb_reg.feature_importances_).sort_values()
```

Out[61]:

```
lagerUnitTypeId        5
commodity_group        7
countryOfOrigin       15
cityId                22
brandId               99
dayofweek            179
lagerUnitQuantity    185
trademark            220
month                290
productCategoryId    297
productTypeId        315
day                  703
price                867
geoCluster          1371
SKU                 1455
dtype: int32
```

In [45]:

```
X_train.columns
```

Out[45]:

```
Index(['geoCluster', 'SKU', 'price', 'productCategoryId', 'productTypeId',
       'brandId', 'lagerUnitQuantity', 'lagerUnitTypeId', 'trademark',
       'countryOfOrigin', 'commodity_group', 'cityId', 'month', 'day',
       'dayofweek'],
      dtype='object')
```

In [46]:

```
gc.collect()
```

Out[46]:

```
14
```

Прогноз

In [47]:

```
df_test = merge_data(df_test)
df_test = preprocess_date(df_test)
df_test = categorize_columns(df_test)
```

In [48]:

```
X_sku_prices = X.groupby(['SKU'])['price'].agg(['min', 'max'])
```

In [49]:

```
def adjust_price(price, sku):
    price_min_max = X_sku_prices.loc[sku]
    if price > price_min_max['max']:
        return price_min_max['max']
    if price < price_min_max['min']:
        return price_min_max['min']
    return price

train_skus = X_sku_prices.index.tolist()
df_test.loc[df_test['SKU'].isin(train_skus), 'price'] = df_test.loc[df_test['SKU'].isin
```

In [50]:

```
df_test.drop(info_cols, axis=1, inplace=True)
```

In [51]:

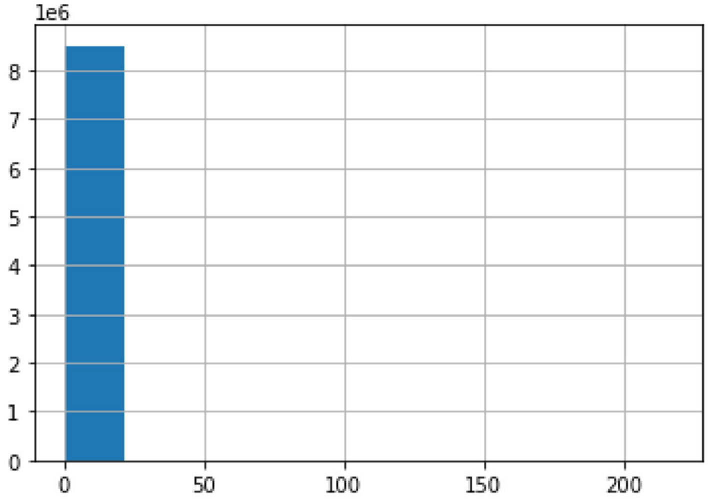
```
predictions = []
for model in models:
    predictions.append(model.predict(df_test, num_iteration=lgb_reg.best_iteration_))
```

In [52]:

```
y.hist()
```

Out[52]:

```
<AxesSubplot:>
```

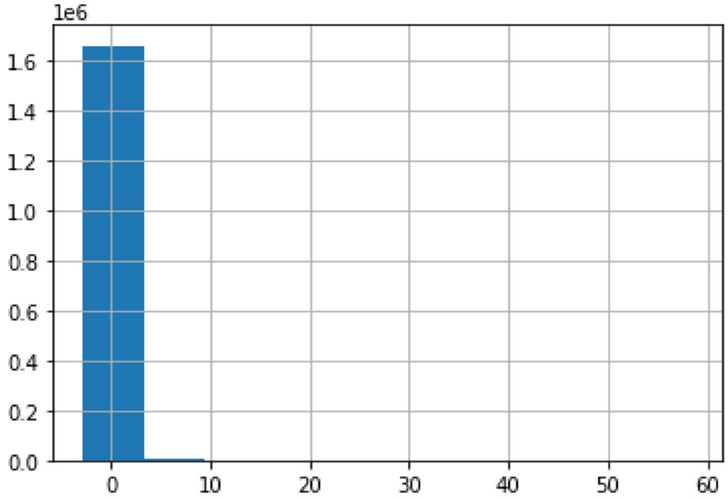


In [53]:

```
submission['sales'] = np.array(predictions).mean(axis=0)
submission['sales'].hist()
```

Out[53]:

```
<AxesSubplot:>
```



In [54]:

```
submission['sales'].min()
```

Out[54]:

```
-2.7819200747371204
```

In [55]:

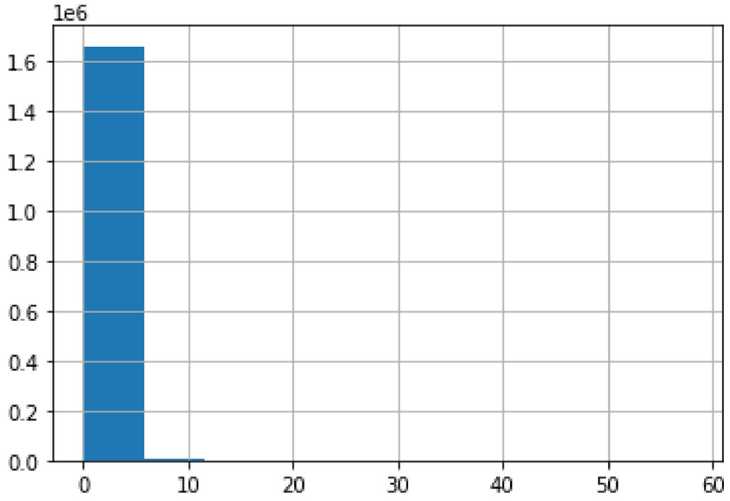
```
submission.loc[submission['sales']<0, 'sales'] = 0

submission['sales'] = np.round(submission['sales'])

submission['sales'].hist()
```

Out[55]:

```
<AxesSubplot:>
```



In [57]:

```
ts = dt.datetime.now().strftime('%Y%m%d_%H_%M_%S')
submission.to_csv(
    os.path.join(
        OUTPUT_FOLDER,
        f'{ts}.csv'
    )
)
```