CatBoost

September 13, 2025

0.0.1 CatBoost is like XGBoost but more advanced. It has tuning in itslef and it can tun the parameteres byself. Another advantage of CatBoost is that use GPU fpr processing data, instead of anothers that use CPU.

We should install by running: pip install -U catboost

CatBoost has regression and classification operations.

[1]:
```
import numpy as np
import pandas as pd
```

[2]:
```
df = pd.read_csv('train.csv')
df
```

[2]:

	id	cat1	cat2	cat3	cat4	cat5	cat6	cat7	cat8	cat9	...	cont6	\
0	1	A	B	A	B	A	A	A	A	B	...	0.718367	
1	2	A	B	A	A	A	A	A	A	B	...	0.438917	
2	5	A	B	A	A	B	A	A	A	B	...	0.289648	
3	10	B	B	A	B	A	A	A	A	B	...	0.440945	
4	11	A	B	A	B	A	A	A	A	B	...	0.178193	
...	...	...	...	...	...	...	...	...	...	...	...	...	
188313	587620	A	B	A	A	A	A	A	A	B	...	0.242437	
188314	587624	A	A	A	A	A	B	A	A	A	...	0.334270	
188315	587630	A	B	A	A	A	A	A	B	B	...	0.345883	
188316	587632	A	B	A	A	A	A	A	A	B	...	0.704364	
188317	587633	B	A	A	B	A	A	A	A	A	...	0.844563	

	cont7	cont8	cont9	cont10	cont11	cont12	cont13	\
0	0.335060	0.30260	0.67135	0.83510	0.569745	0.594646	0.822493	
1	0.436585	0.60087	0.35127	0.43919	0.338312	0.366307	0.611431	
2	0.315545	0.27320	0.26076	0.32446	0.381398	0.373424	0.195709	
3	0.391128	0.31796	0.32128	0.44467	0.327915	0.321570	0.605077	
4	0.247408	0.24564	0.22089	0.21230	0.204687	0.202213	0.246011	
...	...	...	...	...	...	...	...	
188313	0.289949	0.24564	0.30859	0.32935	0.223038	0.220003	0.333292	
188314	0.382000	0.63475	0.40455	0.47779	0.307628	0.301921	0.318646	
188315	0.370534	0.24564	0.45808	0.47779	0.445614	0.443374	0.339244	
188316	0.562866	0.34987	0.44767	0.53881	0.863052	0.852865	0.654753	
188317	0.533048	0.97123	0.93383	0.83814	0.932195	0.946432	0.810511	

```
          cont14     loss
0       0.714843  2213.18
1       0.304496  1283.60
2       0.774425  3005.09
3       0.602642   939.85
4       0.432606  2763.85
...          ...      ...
188313  0.208216  1198.62
188314  0.305872  1108.34
188315  0.503888  5762.64
188316  0.721707  1562.87
188317  0.721460  4751.72

[188318 rows x 132 columns]
```

Above is insurance dataset with unknown datas. at the end is loss that insurance paid for each person.

As above, cats are categorical datas and conts is continous datas. Thats better to say that each data of conts are min_max scaled and it is not their real value.

[3]:
```
df.isnull().sum()
```

[3]:
```
id        0
cat1      0
cat2      0
cat3      0
cat4      0
         ..
cont11    0
cont12    0
cont13    0
cont14    0
loss      0
Length: 132, dtype: int64
```

[4]:
```
df.isnull().sum().sum()
```

[4]:
```
0
```

[5]:
```
df.describe()
```

[5]:
```
                  id          cont1          cont2          cont3  \
count  188318.000000  188318.000000  188318.000000  188318.000000
mean   294135.982561       0.493861       0.507188       0.498918
std    169336.084867       0.187640       0.207202       0.202105
min         1.000000       0.000016       0.001149       0.002634
```

```
25%    147748.250000       0.346090       0.358319       0.336963
50%    294539.500000       0.475784       0.555782       0.527991
75%    440680.500000       0.623912       0.681761       0.634224
max    587633.000000       0.984975       0.862654       0.944251

              cont4          cont5          cont6          cont7  \
count  188318.000000  188318.000000  188318.000000  188318.000000
mean        0.491812       0.487428       0.490945       0.484970
std         0.211292       0.209027       0.205273       0.178450
min         0.176921       0.281143       0.012683       0.069503
25%         0.327354       0.281143       0.336105       0.350175
50%         0.452887       0.422268       0.440945       0.438285
75%         0.652072       0.643315       0.655021       0.591045
max         0.954297       0.983674       0.997162       1.000000

              cont8          cont9         cont10         cont11  \
count  188318.000000  188318.000000  188318.000000  188318.000000
mean        0.486437       0.485506       0.498066       0.493511
std         0.199370       0.181660       0.185877       0.209737
min         0.236880       0.000080       0.000000       0.035321
25%         0.312800       0.358970       0.364580       0.310961
50%         0.441060       0.441450       0.461190       0.457203
75%         0.623580       0.566820       0.614590       0.678924
max         0.980200       0.995400       0.994980       0.998742

             cont12         cont13         cont14           loss
count  188318.000000  188318.000000  188318.000000  188318.000000
mean        0.493150       0.493138       0.495717    3037.337686
std         0.209427       0.212777       0.222488    2904.086186
min         0.036232       0.000228       0.179722       0.670000
25%         0.311661       0.315758       0.294610    1204.460000
50%         0.462286       0.363547       0.407403    2115.570000
75%         0.675759       0.689974       0.724623    3864.045000
max         0.998484       0.988494       0.844848  121012.250000
```

However we write df.describe(), the result is not helpful.

[6]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 188318 entries, 0 to 188317
Columns: 132 entries, id to loss
dtypes: float64(15), int64(1), object(116)
memory usage: 189.7+ MB
```

[7]:
```
traindf = pd.read_csv('train.csv')
testdf = pd.read_csv('test.csv')
```

We have two types of dataset. we will train the traindf and predict the testdf.

[8]:
```
testdf
```

[8]:
```
            id cat1 cat2 cat3 cat4 cat5 cat6 cat7 cat8 cat9  ...     cont5  \
0            4    A    B    A    A    A    A    A    A    B  ...  0.281143
1            6    A    B    A    B    A    A    A    A    B  ...  0.836443
2            9    A    B    A    B    B    A    B    A    B  ...  0.718531
3           12    A    A    A    A    B    A    A    A    A  ...  0.397069
4           15    B    A    A    A    A    B    A    A    A  ...  0.302678
...        ...  ...  ...  ...  ...  ...  ...  ...  ...  ...  ...       ...
125541  587617    A    A    A    B    A    A    A    A    A  ...  0.281143
125542  587621    A    A    A    A    B    B    A    B    A  ...  0.674529
125543  587627    B    B    A    A    B    A    A    A    B  ...  0.794794
125544  587629    A    A    A    A    A    B    A    B    A  ...  0.302678
125545  587634    A    B    A    A    A    A    A    A    B  ...  0.413817

           cont6     cont7    cont8    cont9   cont10    cont11    cont12  \
0       0.466591  0.317681  0.61229  0.34365  0.38016  0.377724  0.369858
1       0.482425  0.443760  0.71330  0.51890  0.60401  0.689039  0.675759
2       0.212308  0.325779  0.29758  0.34365  0.30529  0.245410  0.241676
3       0.369930  0.342355  0.40028  0.33237  0.31480  0.348867  0.341872
4       0.398862  0.391833  0.23688  0.43731  0.50556  0.359572  0.352251
...          ...       ...      ...      ...      ...       ...       ...
125541  0.438917  0.815941  0.39455  0.48740  0.40666  0.550529  0.538473
125542  0.346948  0.424968  0.47669  0.25753  0.26894  0.324486  0.352251
125543  0.808958  0.511502  0.72299  0.94438  0.83510  0.933174  0.926619
125544  0.372125  0.388545  0.31796  0.32128  0.36974  0.307628  0.301921
125545  0.221699  0.242044  0.25461  0.31399  0.25183  0.245410  0.241676

          cont13    cont14
0       0.704052  0.392562
1       0.453468  0.208045
2       0.258586  0.297232
3       0.592264  0.555955
4       0.301535  0.825823
...          ...       ...
125541  0.298734  0.345946
125542  0.490001  0.290576
125543  0.848129  0.808125
125544  0.608259  0.361542
125545  0.287682  0.220323

[125546 rows x 131 columns]
```

[9]:
```
import re

# Compile patterns for matching 'cat1' to 'cat999'
```

```
cat_pattern = re.compile(r'^cat(\d+)$')
cont_pattern = re.compile(r'^cont(\d+)$')

# Filter and sort categorical columns
cat_col = sorted(
    [col for col in traindf.columns if cat_pattern.match(col)],
    key=lambda s: int(s[3:])
)
cat_col
```

[9]:
```
['cat1',
 'cat2',
 'cat3',
 'cat4',
 'cat5',
 'cat6',
 'cat7',
 'cat8',
 'cat9',
 'cat10',
 'cat11',
 'cat12',
 'cat13',
 'cat14',
 'cat15',
 'cat16',
 'cat17',
 'cat18',
 'cat19',
 'cat20',
 'cat21',
 'cat22',
 'cat23',
 'cat24',
 'cat25',
 'cat26',
 'cat27',
 'cat28',
 'cat29',
 'cat30',
 'cat31',
 'cat32',
 'cat33',
 'cat34',
 'cat35',
 'cat36',
 'cat37',
```

```
'cat38',
'cat39',
'cat40',
'cat41',
'cat42',
'cat43',
'cat44',
'cat45',
'cat46',
'cat47',
'cat48',
'cat49',
'cat50',
'cat51',
'cat52',
'cat53',
'cat54',
'cat55',
'cat56',
'cat57',
'cat58',
'cat59',
'cat60',
'cat61',
'cat62',
'cat63',
'cat64',
'cat65',
'cat66',
'cat67',
'cat68',
'cat69',
'cat70',
'cat71',
'cat72',
'cat73',
'cat74',
'cat75',
'cat76',
'cat77',
'cat78',
'cat79',
'cat80',
'cat81',
'cat82',
'cat83',
'cat84',
```

```
 'cat85',
 'cat86',
 'cat87',
 'cat88',
 'cat89',
 'cat90',
 'cat91',
 'cat92',
 'cat93',
 'cat94',
 'cat95',
 'cat96',
 'cat97',
 'cat98',
 'cat99',
 'cat100',
 'cat101',
 'cat102',
 'cat103',
 'cat104',
 'cat105',
 'cat106',
 'cat107',
 'cat108',
 'cat109',
 'cat110',
 'cat111',
 'cat112',
 'cat113',
 'cat114',
 'cat115',
 'cat116']
```

[11]:
```
cat_index = [i for i in range(0, len(traindf.columns)) if cat_pattern.
 ↪match(traindf.columns[i])]
cat_index
```

[11]:
```
[1,
 2,
 3,
 4,
 5,
 6,
 7,
 8,
 9,
 10,
```

11,
12,
13,
14,
15,
16,
17,
18,
19,
20,
21,
22,
23,
24,
25,
26,
27,
28,
29,
30,
31,
32,
33,
34,
35,
36,
37,
38,
39,
40,
41,
42,
43,
44,
45,
46,
47,
48,
49,
50,
51,
52,
53,
54,
55,
56,
57,

```
58,
59,
60,
61,
62,
63,
64,
65,
66,
67,
68,
69,
70,
71,
72,
73,
74,
75,
76,
77,
78,
79,
80,
81,
82,
83,
84,
85,
86,
87,
88,
89,
90,
91,
92,
93,
94,
95,
96,
97,
98,
99,
100,
101,
102,
103,
104,
```

```
 105,
 106,
 107,
 108,
 109,
 110,
 111,
 112,
 113,
 114,
 115,
 116]
```

[13]:
```
cont_col = sorted(
    [col for col in traindf.columns if cont_pattern.match(col)],
    key=lambda s: int(s[4:])
)
cont_col
```

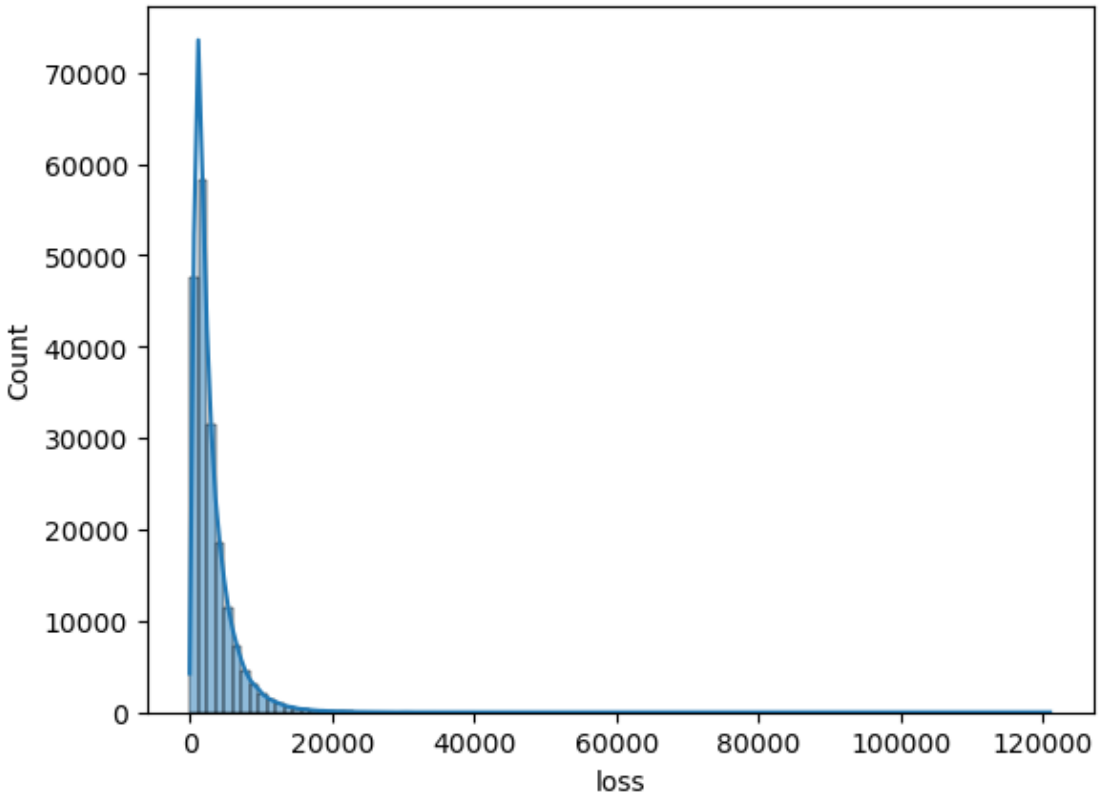
[13]:
```
['cont1',
 'cont2',
 'cont3',
 'cont4',
 'cont5',
 'cont6',
 'cont7',
 'cont8',
 'cont9',
 'cont10',
 'cont11',
 'cont12',
 'cont13',
 'cont14']
```

[15]:
```
cont_index = [i for i in range(0, len(traindf.columns)) if cont_pattern.
 ↪match(traindf.columns[i])]
cont_index
```

[15]:
```
[117, 118, 119, 120, 121, 122, 123, 124, 125, 126, 127, 128, 129, 130]
```

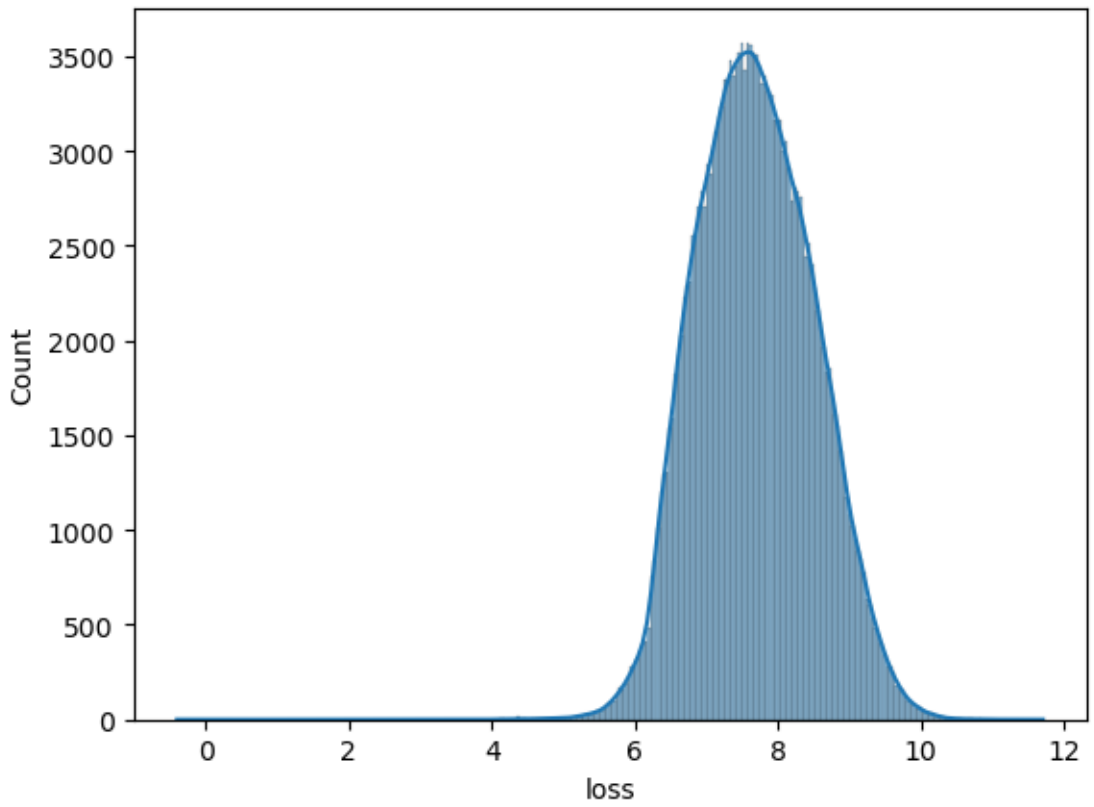
[16]:
```
import seaborn as sns
sns.histplot(df['loss'], bins=100, kde=True),
```

[16]:
```
(<Axes: xlabel='loss', ylabel='Count'>,)
```



AS above, our hist is right skewness. So that is better to transform it to normal. As I mentioned it is better, Not necessary.

[17]:
```
sns.histplot(np.log(df['loss']), kde=True);
```



Now we are going to train and test the model:

[18]:
```
from catboost import CatBoostRegressor
from sklearn.model_selection import train_test_split
```

[19]:
```
cb_reg = CatBoostRegressor(iterations = 120, learning_rate=0.05, depth=6,␣
↪eval_metric='MAE', verbose=10)
```

[21]:
```
x = df.drop(['id', 'loss'], axis=1)
y = np.log(df['loss'])
x_train, x_test, y_train, y_test = train_test_split(x, y, random_state=123,␣
↪test_size=0.25)
```

As we have data with large memory space occupied, we should delete the datas we do not need.

[22]:
```
del x
del y
```

[23]:
```
np.asarray(cat_index) - 1
```

[23]:
```
array([  0,   1,   2,   3,   4,   5,   6,   7,   8,   9,  10,  11,  12,
        13,  14,  15,  16,  17,  18,  19,  20,  21,  22,  23,  24,  25,
        26,  27,  28,  29,  30,  31,  32,  33,  34,  35,  36,  37,  38,
        39,  40,  41,  42,  43,  44,  45,  46,  47,  48,  49,  50,  51,
        52,  53,  54,  55,  56,  57,  58,  59,  60,  61,  62,  63,  64,
        65,  66,  67,  68,  69,  70,  71,  72,  73,  74,  75,  76,  77,
        78,  79,  80,  81,  82,  83,  84,  85,  86,  87,  88,  89,  90,
        91,  92,  93,  94,  95,  96,  97,  98,  99, 100, 101, 102, 103,
       104, 105, 106, 107, 108, 109, 110, 111, 112, 113, 114, 115])
```

As before we removed id column, so the cat_index become -1.

[25]:
```
cb_reg.fit(x_train, y_train, np.asarray(cat_index) -1, eval_set=(x_test, y_test))
```

```
0:      learn: 0.6477590        test: 0.6494355 best: 0.6494355 (0)     total:
225ms    remaining: 26.7s
10:     learn: 0.5618648        test: 0.5635349 best: 0.5635349 (10)    total:
862ms    remaining: 8.54s
20:     learn: 0.5176842        test: 0.5193726 best: 0.5193726 (20)    total:
1.46s    remaining: 6.89s
30:     learn: 0.4917490        test: 0.4933894 best: 0.4933894 (30)    total:
2.08s    remaining: 5.97s
40:     learn: 0.4757679        test: 0.4777449 best: 0.4777449 (40)    total:
2.67s    remaining: 5.15s
50:     learn: 0.4651767        test: 0.4673669 best: 0.4673669 (50)    total:
3.27s    remaining: 4.43s
60:     learn: 0.4571878        test: 0.4596060 best: 0.4596060 (60)    total:
3.9s     remaining: 3.78s
70:     learn: 0.4513110        test: 0.4538142 best: 0.4538142 (70)    total:
4.51s    remaining: 3.12s
80:     learn: 0.4468587        test: 0.4495596 best: 0.4495596 (80)    total:
5.12s    remaining: 2.47s
90:     learn: 0.4432433        test: 0.4460570 best: 0.4460570 (90)    total:
5.74s    remaining: 1.83s
100:    learn: 0.4404866        test: 0.4433463 best: 0.4433463 (100)   total:
6.37s    remaining: 1.2s
110:    learn: 0.4381826        test: 0.4411855 best: 0.4411855 (110)   total:
6.96s    remaining: 565ms
119:    learn: 0.4366056        test: 0.4396295 best: 0.4396295 (119)   total:
7.52s    remaining: 0us

bestTest = 0.439629485
bestIteration = 119
```

[25]:
```
<catboost.core.CatBoostRegressor at 0x2209df80dd0>
```

Then we would change the log(df['loss']) to df['loss'].

[27]:
```
np.exp(0.439629485)
```

[27]:
```
1.5521320237619909
```

Just for notice, all train done on cpu. now we want to do it in gpu.

[28]:
```
cb_reg = CatBoostRegressor(iterations = 200, learning_rate=0.05, depth=6,␣
↪eval_metric='MAE', verbose=10, task_type='GPU', save_snapshot=True,␣
↪snapshot_file='mohammadreza', snapshot_interval=10 )
```

[30]:
```
x = df.drop(['id', 'loss'], axis=1)
y = np.log(df['loss'])
x_train, x_test, y_train, y_test = train_test_split(x, y, random_state=123,␣
↪test_size=0.25)
```

[31]:
```
cb_reg.fit(x_train, y_train, np.asarray(cat_index) -1, eval_set=(x_test, y_test))
```

```
Default metric period is 5 because MAE is/are not implemented for GPU

bestTest = 0.4289730412
bestIteration = 199
```

[31]:
```
<catboost.core.CatBoostRegressor at 0x220c29ab250>
```

[32]:
```
np.exp(0.4289730412)
```

[32]:
```
1.5356796337591856
```

[33]:
```
cb_reg.save_model('catboost_reg.h5')
```

If we want to use the model:

[34]:
```
model = cb_reg.load_model('catboost_reg.h5')
```

[35]:
```
model
```

[35]:
```
<catboost.core.CatBoostRegressor at 0x220c29ab250>
```

[36]:
```
testdf.drop('id', axis=1, inplace=True)
testdf
```

[36]:

	cat1	cat2	cat3	cat4	cat5	cat6	cat7	cat8	cat9	cat10	...	cont5	\
0	A	B	A	A	A	A	A	A	B	A	...	0.281143	
1	A	B	A	B	A	A	A	A	B	A	...	0.836443	
2	A	B	A	B	B	A	B	A	B	B	...	0.718531	
3	A	A	A	A	B	A	A	A	A	A	...	0.397069	
4	B	A	A	A	A	B	A	A	A	A	...	0.302678	
...	...	...	...	...	...	...	...	...	...	...	...	...	
125541	A	A	A	B	A	A	A	A	A	A	...	0.281143	
125542	A	A	A	A	B	B	A	B	A	A	...	0.674529	
125543	B	B	A	A	B	A	A	A	B	B	...	0.794794	

```
125544    A    A    A    A    A    B    A    B    A     A  ...  0.302678
125545    A    B    A    A    A    A    A    A    B     A  ...  0.413817

           cont6     cont7    cont8    cont9   cont10    cont11    cont12  \
0       0.466591  0.317681  0.61229  0.34365  0.38016  0.377724  0.369858
1       0.482425  0.443760  0.71330  0.51890  0.60401  0.689039  0.675759
2       0.212308  0.325779  0.29758  0.34365  0.30529  0.245410  0.241676
3       0.369930  0.342355  0.40028  0.33237  0.31480  0.348867  0.341872
4       0.398862  0.391833  0.23688  0.43731  0.50556  0.359572  0.352251
...          ...       ...      ...      ...      ...       ...       ...
125541  0.438917  0.815941  0.39455  0.48740  0.40666  0.550529  0.538473
125542  0.346948  0.424968  0.47669  0.25753  0.26894  0.324486  0.352251
125543  0.808958  0.511502  0.72299  0.94438  0.83510  0.933174  0.926619
125544  0.372125  0.388545  0.31796  0.32128  0.36974  0.307628  0.301921
125545  0.221699  0.242044  0.25461  0.31399  0.25183  0.245410  0.241676

          cont13    cont14
0       0.704052  0.392562
1       0.453468  0.208045
2       0.258586  0.297232
3       0.592264  0.555955
4       0.301535  0.825823
...          ...       ...
125541  0.298734  0.345946
125542  0.490001  0.290576
125543  0.848129  0.808125
125544  0.608259  0.361542
125545  0.287682  0.220323

[125546 rows x 130 columns]
```

[37]:
```
model.predict(testdf)
```

[37]:
```
array([7.38559738, 7.6236115 , 9.10951239, ..., 7.84359677, 6.79103233,
       8.21779317])
```

[38]:
```
np.exp(model.predict(testdf))
```

[38]:
```
array([1612.59083566, 2045.93771681, 9040.88536248, ..., 2549.35781777,
        889.83168645, 3706.31413272])
```

[39]:
```
testdf['loss'] = np.exp(model.predict(testdf))
testdf
```

[39]:
```
       cat1 cat2 cat3 cat4 cat5 cat6 cat7 cat8 cat9 cat10  ...     cont6  \
0         A    B    A    A    A    A    A    A    B     A  ...  0.466591
1         A    B    A    B    A    A    A    A    B     A  ...  0.482425
```

```
2         A    B    A    B    B    A    B    A    B     B  ...  0.212308
3         A    A    A    A    B    A    A    A    A     A  ...  0.369930
4         B    A    A    A    A    B    A    A    A     A  ...  0.398862
...     ...  ...  ...  ...  ...  ...  ...  ...  ...   ...  ...       ...
125541    A    A    A    B    A    A    A    A    A     A  ...  0.438917
125542    A    A    A    A    B    B    A    B    A     A  ...  0.346948
125543    B    B    A    A    B    A    A    A    B     B  ...  0.808958
125544    A    A    A    A    A    B    A    B    A     A  ...  0.372125
125545    A    B    A    A    A    A    A    A    B     A  ...  0.221699

           cont7    cont8    cont9   cont10    cont11    cont12    cont13  \
0       0.317681  0.61229  0.34365  0.38016  0.377724  0.369858  0.704052
1       0.443760  0.71330  0.51890  0.60401  0.689039  0.675759  0.453468
2       0.325779  0.29758  0.34365  0.30529  0.245410  0.241676  0.258586
3       0.342355  0.40028  0.33237  0.31480  0.348867  0.341872  0.592264
4       0.391833  0.23688  0.43731  0.50556  0.359572  0.352251  0.301535
...          ...      ...      ...      ...       ...       ...       ...
125541  0.815941  0.39455  0.48740  0.40666  0.550529  0.538473  0.298734
125542  0.424968  0.47669  0.25753  0.26894  0.324486  0.352251  0.490001
125543  0.511502  0.72299  0.94438  0.83510  0.933174  0.926619  0.848129
125544  0.388545  0.31796  0.32128  0.36974  0.307628  0.301921  0.608259
125545  0.242044  0.25461  0.31399  0.25183  0.245410  0.241676  0.287682

          cont14         loss
0       0.392562  1612.590836
1       0.208045  2045.937717
2       0.297232  9040.885362
3       0.555955  4907.367429
4       0.825823   936.389476
...          ...          ...
125541  0.345946  2288.545241
125542  0.290576  1964.778240
125543  0.808125  2549.357818
125544  0.361542   889.831686
125545  0.220323  3706.314133

[125546 rows x 131 columns]
```

[40]:
```
testdf.to_csv('predict.csv')
```

also we can save it in excel mode like testdf.to_excel('predict.xslx')