American Express - Default Prediction

- 대회 내용 : 고객이 미래의 채무 불이행 여부를 예측
- 대회 링크 : https://www.kaggle.com/competitions/amex-default-prediction (https://www.kaggle.com/competitions/amex-default-prediction)
- 코드 참조 링크 : https://www.kaggle.com/code/kagglestart/amex-02-basic-lightgbm-2208 (https://www.kaggle.com/code/kagglestart/amex-02-basic-lightgbm-2208)
- 대회 평가 : M = 0.5 * (G + D)
  - G : Normalized Gini Coefficient
  - D : 4%에서의 기본 비율(default rate)
- 데이터 셋
  - train_data : 16.39 GB, test_data : 33.82 GB

학습 목표

- lightgbm 알고리즘을 활용한 데이터 EDA 부터, 기본 모델을 만들어 제출해봅니다.

목차

01. 라이브러리 불러오기

목차로 이동하기

In [1]:

```
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import gc
import datetime


from sklearn.model_selection import StratifiedKFold
import lightgbm as lgbm
from lightgbm import early_stopping

import seaborn as sns
import matplotlib.pyplot as plt

import warnings
warnings.simplefilter("ignore")

NUM_FOLDS = 5
```

02. 데이터 로드 및 데이터가 차지 RAM Size 줄이기

목차로 이동하기

Discussion : 어떻게 데이터 사이즈를 줄일 것인가?

- https://www.kaggle.com/competitions/amex-default-prediction/discussion/328054 (https://www.kaggle.com/competitions/amex-default-prediction/discussion/328054)
- S_2는 시간이 있는 날짜 열이다. 행당 10바이트를 차지. 이 열을 pd.to_datetime() 로 변환하면 4 바이트가 된다.

In [2]:

```
%%time
df_train = pd.read_parquet("/kaggle/input/amex-data-integer-dtypes-parquet-format/train.parquet")

# S_2를 datatime으로
df_train["S_2"] = pd.to_datetime(df_train["S_2"])
df_train["days"] = (df_train["S_2"] - df_train.groupby(["customer_ID"])["S_2"].transform("min")).dt.

# float32 -> float16
for col in df_train[df_train.columns[df_train.dtypes=="float32"]]:
    df_train[col] = df_train[col].astype("float16")
```

```
CPU times: user 33.1 s, sys: 28.9 s, total: 1min 2s
Wall time: 1min 3s
```

In [3]:

```
gc.collect()
```

Out[3]:

```
68
```

In [4]:

```
print( df_train.shape)
print( df_train['customer_ID'].value_counts().shape )
print( df_train['customer_ID'].value_counts() )
print( )
```

```
(5531451, 191)
(458913,)
0000099d6bd597052cdcda90ffabf56573fe9d7c79be5fbac11a8ed792feb62a    13
a3111280bfa1ed8fafd0b06839eb707f4538497e8087cb62958bb03e1bdde214    13
a31376930229162f886c091e5a56a528f81c10a523285828ed05a6e9ccf56722    13
a312c595dfaee96c8a597107d2754a49b1acfd127400d98991762d87837b1b65    13
a312aff722e7230f9d6a313ff777d6f00166c6bada21a333982426758a2e2a9d    13
                                                                    ..
a84839802f1f37a86a7fe34ddba4791d33d878df3937b509841def0a9e252748     1
01f4f7b14d83b6a8f88e43552792246 15da083b19e3e5f15b98f274ced8cf752     1
eef07ea56302cebcd57374c6565bb3e5c7af856796d9cbc31ed42aa0fc73b7fc     1
d192480082e86e3b4da68f014b284f2a2624b45956eed279416c796de043b7ce     1
d9ea3cffff889b522a69bde89aee382dcff8bffe32c9a38653bdaa2ff4330041     1
Name: customer_ID, Length: 458913, dtype: int64
```

03. customer_ID로 그룹을 만들고, 그룹의 마지막 최신 데이터를 확인

[목차로 이동하기]

In [5]:

```
df_train.groupby(["customer_ID"]).tail(1)
```

Out[5]:

	customer_ID	S_2	P_2	D_39	B_1
12	0000099d6bd597052cdcda90ffabf56573fe9d7c79be5f...	2018-03-13	0.934570	0	0.009384
25	00000fd6641609c6ece5454664794f0340ad84dddce9a2...	2018-03-25	0.880371	6	0.034698
38	00001b22f846c82c51f6e3958ccd81970162bae8b007e8...	2018-03-12	0.880859	0	0.004284
51	000041bdba6ecadd89a52d11886e8eaaec9325906c9723...	2018-03-29	0.621582	0	0.012566
64	00007889e4fcd2614b6cbe7f8f3d2e5c728eca32d9eb8a...	2018-03-30	0.872070	0	0.007679
...	...	...	...	...	...
5531398	ffff41c8a52833b56430603969b9ca48d208e7c192c6a4...	2018-03-31	0.844238	15	0.028519
5531411	ffff518bb2075e4816ee3fe9f3b152c57fc0e6f01bf7fd...	2018-03-22	0.831055	1	0.292480
5531424	ffff9984b999fccb2b6127635ed0736dda94e544e67e02...	2018-03-07	0.800293	9	0.020569
5531437	ffffa5c46bc8de74f5a4554e74e239c8dee6b9baf38814...	2018-03-23	0.753906	0	0.015839
5531450	fffff1d38b785cef84adeace64f8f83db3a0c31e8d92ea...	2018-03-14	0.981934	0	0.000077

458913 rows × 191 columns

In [6]:

```
df_train = df_train.groupby(["customer_ID"]).tail(1).set_index('customer_ID')
```

04. Target 값

목차로 이동하기

In [7]:

```
%%time
df_train_labels = pd.read_csv("/kaggle/input/amex-default-prediction/train_labels.csv")
df_train_labels["target"] = df_train_labels["target"].astype("int8")
print(df_train_labels.shape)
df_train_labels.head()
```

```
(458913, 2)
CPU times: user 495 ms, sys: 126 ms, total: 620 ms
Wall time: 1.03 s
```

Out[7]:

	customer_ID	target
0	0000099d6bd597052cdcda90ffabf56573fe9d7c79be5f...	0
1	00000fd6641609c6ece5454664794f0340ad84dddce9a2...	0
2	00001b22f846c82c51f6e3958ccd81970162bae8b007e8...	0
3	000041bdba6ecadd89a52d11886e8eaaec9325906c9723...	0
4	00007889e4fcd2614b6cbe7f8f3d2e5c728eca32d9eb8a...	0

In [8]:

```
%%time
df_train = df_train.merge(df_train_labels, on="customer_ID", how='left')
print(df_train.shape)
print(df_train.head())
del df_train_labels
gc.collect()
```

```
(458913, 192)
                                         customer_ID         S_2       P_2  ₩
0  0000099d6bd597052cdcda90ffabf56573fe9d7c79be5f...  2018-03-13  0.934570   
1  00000fd6641609c6ece5454664794f0340ad84dddce9a2...  2018-03-25  0.880371   
2  00001b22f846c82c51f6e3958ccd81970162bae8b007e8...  2018-03-12  0.880859   
3  000041bdba6ecadd89a52d11886e8eaaec9325906c9723...  2018-03-29  0.621582   
4  00007889e4fcd2614b6cbe7f8f3d2e5c728eca32d9eb8a...  2018-03-30  0.872070   

   D_39       B_1       B_2       R_1       S_3  D_41       B_3  ...  D_138  ₩
0     0  0.009384  1.007812  0.006104  0.135010   0.0  0.007175  ...     -1   
1     6  0.034698  1.003906  0.006912  0.165527   0.0  0.005070  ...     -1   
2     0  0.004284  0.812500  0.006451       NaN   0.0  0.007195  ...     -1   
3     0  0.012566  1.005859  0.007828  0.287842   0.0  0.009941  ...     -1   
4     0  0.007679  0.815918  0.001247       NaN   0.0  0.005527  ...     -1   

   D_139  D_140  D_141  D_142  D_143     D_144  D_145  days  target  
0      0      0    0.0    NaN      0  0.002970      0   370       0  
1      0      0    0.0    NaN      0  0.003170      0   390       0  
2      0      0    0.0    NaN      0  0.000834      0   367       0  
3      0      0    0.0    NaN      0  0.005558      0   364       0  
4      0      0    0.0    NaN      0  0.006943      0   366       0  

[5 rows x 192 columns]
CPU times: user 1.01 s, sys: 77 ms, total: 1.09 s
Wall time: 1.09 s
```

Out[8]:

```
0
```

05. 평가지표(Metric)

목차로 이동하기

In [9]:

```
# https://www.kaggle.com/code/cdeotte/xgboost-starter-0-793/notebook
# https://www.kaggle.com/kyakovlev
# https://www.kaggle.com/competitions/amex-default-prediction/discussion/327534
def amex_metric_mod(y_true, y_pred):

    labels     = np.transpose(np.array([y_true, y_pred]))
    labels     = labels[labels[:, 1].argsort()[::-1]]
    weights    = np.where(labels[:,0]==0, 20, 1)
    cut_vals   = labels[np.cumsum(weights) <= int(0.04 * np.sum(weights))]
    top_four   = np.sum(cut_vals[:,0]) / np.sum(labels[:,0])

    gini = [0,0]
    for i in [1,0]:
        labels         = np.transpose(np.array([y_true, y_pred]))
        labels         = labels[labels[:, i].argsort()[::-1]]
        weight         = np.where(labels[:,0]==0, 20, 1)
        weight_random  = np.cumsum(weight / np.sum(weight))
        total_pos      = np.sum(labels[:, 0] *  weight)
        cum_pos_found  = np.cumsum(labels[:, 0] * weight)
        lorentz        = cum_pos_found / total_pos
        gini[i]        = np.sum((lorentz - weight_random) * weight)

    return 0.5 * (gini[1]/gini[0] + top_four),_
```

06. 모델 선택 및 학습 - LightGBM

목차로 이동하기

범주형 변수

- https://www.kaggle.com/competitions/amex-default-prediction/discussion/327161 (https://www.kaggle.com/competitions/amex-default-prediction/discussion/327161)
- ['B_30', 'B_38', 'D_114', 'D_116', 'D_117', 'D_120', 'D_126', 'D_63', 'D_64', 'D_66', 'D_68'] 범주형 변수
- D_64, D_66 and D_68 범주형 종류의 개수가 다르다.
- 나머지는 개수가 동일함.

범주형 변수 정보 확인

- cat_col 정보 (LGBM 파라미터로 활용)

In [10]:

```
FEATURES = df_train.columns.drop(["target","customer_ID","S_2"])
categorical_cols = ['B_30', 'B_38', 'D_114', 'D_116', 'D_117', 'D_120', 'D_126', 'D_63', 'D_64', 'D_
cat_col=[]
n=0
for col in df_train[FEATURES]:
    for coll in categorical_cols:
        if col==coll:
            cat_col.append(n)
            break
    n+=1
cat_col
```

Out[10]:

```
[51, 52, 58, 60, 103, 143, 153, 155, 156, 159, 165]
```

In [11]:

```
params = {}
feature_importances = []
scores = []
models = []
pred_val=[]
yval=[]

# 교차 검증 클래스 - 학습용 데이터 셋 인덱스, 검증용 데이터 셋 인덱스
skf = StratifiedKFold(n_splits=NUM_FOLDS, shuffle=True, random_state=2022)
list( enumerate(skf.split(df_train[FEATURES],df_train["target"])) )
```

Out[11]:

```
[(0,
  (array([     0,      1,      2, ..., 458910, 458911, 458912]),
   array([     4,      7,      9, ..., 458889, 458892, 458896]))),
 (1,
  (array([     0,      2,      3, ..., 458910, 458911, 458912]),
   array([     1,      5,     12, ..., 458888, 458906, 458909]))),
 (2,
  (array([     1,      2,      3, ..., 458909, 458911, 458912]),
   array([     0,      6,     25, ..., 458904, 458907, 458910]))),
 (3,
  (array([     0,      1,      3, ..., 458909, 458910, 458911]),
   array([     2,      8,     10, ..., 458905, 458908, 458912]))),
 (4,
  (array([     0,      1,      2, ..., 458909, 458910, 458912]),
   array([     3,     18,     19, ..., 458902, 458903, 458911])))]
```

모델 학습 및 모델 학습 후, 정보 저장

In [12]:

```
%%time
params = {}
feature_importances = []  # 특성 중요도
scores = []               # fold 별 점수
models = []               # 모델
pred_val=[]
yval=[]

# 교차 검증 클래스ㅁ
skf = StratifiedKFold(n_splits=NUM_FOLDS, shuffle=True, random_state=2022)

# 폴드별 데이터 나누기
for fold,(train_idx, val_idx) in enumerate(skf.split(df_train[FEATURES],df_train["target"])):

    print('FOLD:',fold)

    # 데이터 나누기
    X_train = df_train.loc[train_idx, FEATURES].values
    y_train = df_train.loc[train_idx, 'target'].values
    X_val = df_train.loc[val_idx, FEATURES].values
    y_val = df_train.loc[val_idx, 'target'].values

    print("y_train t=0 count:", len(y_train[y_train==0]))
    print("y_train t=1 count:", len(y_train[y_train==1]))
    print("y_val t=0 count:", len(y_val[y_val==0]))
    print("y_val t=1 count:", len(y_val[y_val==1]))


    params = {
        "num_iterations":10000,
        'learning_rate': 0.05,
    }

    # LGBM 알고리즘
    model = lgbm.LGBMClassifier(**params).fit(
        X_train,y_train,
        eval_set=[(X_val,y_val),(X_train,y_train)],
        verbose=100,
        callbacks=[early_stopping(100)],
        categorical_feature=cat_col
    )

    # 특성 중요도
    feature_importances.append(model.feature_importances_)
    models.append(model)
    pred_val = np.append(pred_val,model.predict_proba(X_val)[:,1])
    yval = np.append(yval,y_val)

    del X_train,y_train,X_val,y_val,model
    gc.collect()


score = amex_metric_mod(yval, pred_val)[0]
print('score:', score)
f=open("score.txt","a");f.write(str(score));f.close()
```

```
FOLD: 0
```

```
y_train t=0 count: 272068
y_train t=1 count: 95062
y_val t=0 count: 68017
y_val t=1 count: 23766
[LightGBM] [Warning] Met negative value in categorical features, will convert it
to NaN
[LightGBM] [Warning] Met negative value in categorical features, will convert it
to NaN
[LightGBM] [Warning] Met negative value in categorical features, will convert it
to NaN
[LightGBM] [Warning] Met negative value in categorical features, will convert it
to NaN
[LightGBM] [Warning] Met negative value in categorical features, will convert it
to NaN
[LightGBM] [Warning] Met negative value in categorical features, will convert it
to NaN
[LightGBM] [Warning] Met negative value in categorical features, will convert it
to NaN
[LightGBM] [Warning] Met negative value in categorical features, will convert it
```

In [13]:

```
del df_train,train_idx,val_idx,yval,pred_val
gc.collect()
```

Out[13]:

```
21
```

07. 모델 학습 후, 정보 확인

목차로 이동하기

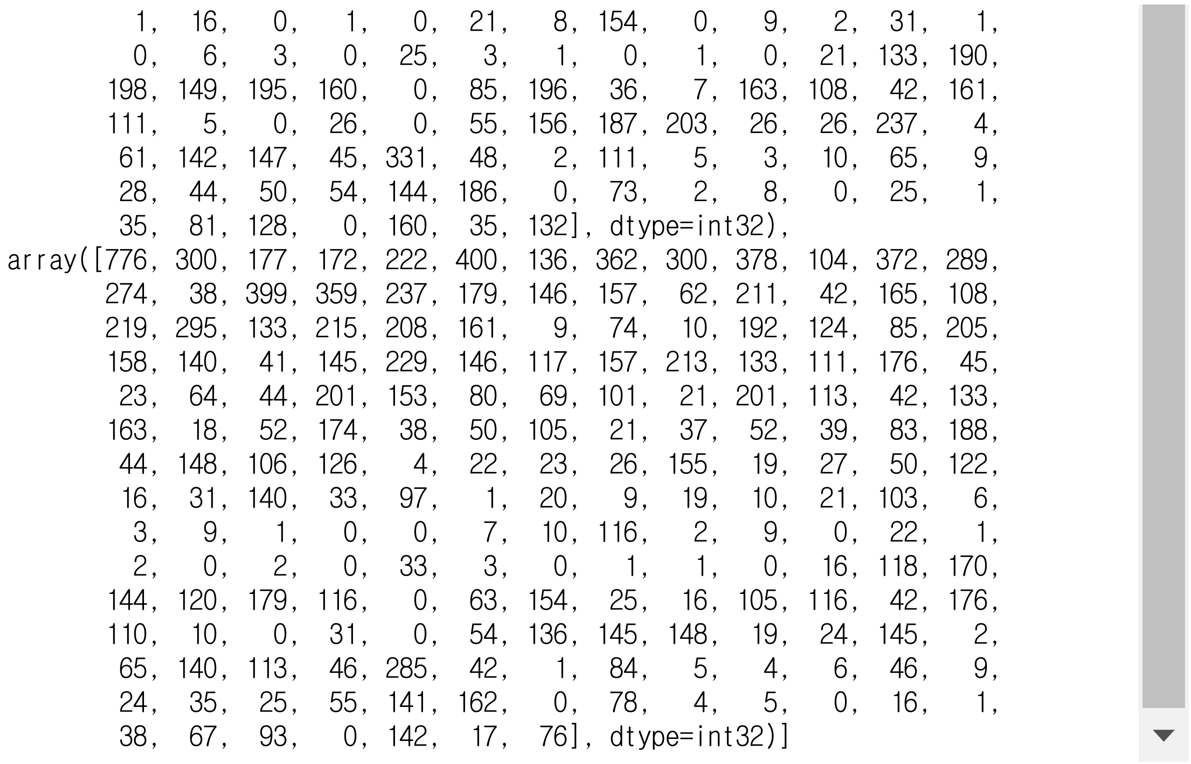
특성 중요도

In [14]:

```
len(feature_importances[0])
feature_importances
```

Out[14]:

```
[array([876, 324, 237, 279, 290, 479, 154, 418, 312, 555, 130, 416, 332,
        350,  36, 542, 418, 341, 211, 208, 197,  86, 294,  42, 262, 116,
        289, 418, 164, 339, 275, 225,  22,  82,   7, 342, 239, 117, 284,
        236, 239,  38, 185, 292, 205, 170, 230, 310, 253, 168, 286,  49,
         39,  70,  80, 348, 203, 127,  69, 112,  16, 290, 205,  54, 232,
        283,  31,  58, 247,  40,  69, 144,  21,  70,  61,  66, 130, 255,
         60, 190, 164, 222,  23,  28,  27,  25, 218,  40,  26,  65, 230,
         21,  25, 213,  36, 148,   0,  10,  10,  21,  13,  31, 132,   9,
          5,  12,   2,   2,   0,  17,   0, 227,   2,   9,   2,  21,   0,
          3,   1,   3,   0,  33,   1,   0,   0,   0,   0,   8, 177, 281,
        208, 254, 277, 208,   0, 124, 257,  53,  19, 217, 148,  49, 191,
        113,  12,   0,  41,   5,  58, 178, 210, 257,  30,  34, 231,   4,
         80, 187, 139,  48, 417,  54,   4, 126,  10,  10,  14,  76,  12,
         24,  35,  52,  71, 163, 275,   0, 102,   2,   9,   0,  26,   0,
         37, 104, 143,   0, 235,  35, 170], dtype=int32),
 array([986, 374, 286, 309, 382, 588, 160, 479, 365, 642, 139, 467, 453,
        415,  30, 591, 504, 410, 210, 233, 243, 129, 384,  49, 309, 131,
        406, 501, 256, 333, 374, 253,  31,  91,  11, 408, 294, 169, 365,
        311, 339,  50, 213, 414, 269, 220, 266, 391, 338, 209, 345,  36,
         46,  60,  85, 404, 212, 142,  60, 105,  20, 362, 306,  64, 330,
        387,  19,  71, 320,  49,  77, 175,  49,  87,  53,  77, 124, 340,
         72, 256, 260, 267,  13,  20,  21,  27, 366,  37,  38,  77, 328,
         20,  60, 334,  44, 253,   3,  21,  30,  21,  11,  43, 129,  12,
          0,  22,   1,   5,   0,  18,  12, 300,   0,  10,   1,  25,   1,
          2,   3,   1,   0,  37,   6,   2,   0,   1,   3,  17, 217, 390,
        289, 280, 295, 289,   0, 131, 299,  35,  21, 257, 166,  56, 283,
        131,  14,   0,  33,   2,  61, 184, 217, 371,  33,  32, 306,   5,
        115, 226, 228,  46, 489,  83,   9, 163,   6,  13,   7,  97,  12,
         21,  49,  86,  84, 204, 359,   0, 129,   6,   4,   0,  20,   0,
         48, 113, 196,   0, 303,  41, 201], dtype=int32),
 array([861, 361, 229, 273, 290, 479, 142, 465, 325, 502, 127, 458, 355,
        374,  38, 463, 414, 344, 203, 208, 205,  74, 329,  41, 268, 129,
        271, 464, 192, 325, 293, 224,  23,  88,  10, 290, 204, 112, 285,
        230, 237,  37, 205, 323, 245, 176, 240, 294, 253, 152, 265,  55,
         42,  74,  73, 325, 251, 116,  64, 106,  14, 306, 205,  50, 274,
        303,  23,  67, 225,  41,  72, 120,  14,  48,  74,  68, 141, 257,
         73, 228, 171, 216,  28,  21,  21,  39, 232,  30,  37,  62, 234,
         28,  33, 244,  31, 175,   4,  28,  29,  23,   8,  44, 126,  11,
          1,  19,   1,   2,   0,  18,   8, 200,   1,  11,   4,  25,   0,
          0,   7,   0,   0,  25,   4,   1,   0,   0,   1,  24, 202, 276,
        203, 218, 252, 211,   3, 132, 266,  37,  16, 232, 159,  46, 234,
        108,  13,   0,  22,   0,  72, 153, 218, 248,  48,  35, 219,   4,
         86, 189, 169,  45, 427,  55,  11, 130,   8,   3,  13,  81,  10,
         27,  48,  70,  64, 163, 230,   0, 102,   1,  16,   0,  23,   0,
         36,  80, 183,   0, 216,  28, 148], dtype=int32),
 array([744, 304, 206, 188, 253, 445, 134, 363, 311, 430, 118, 375, 293,
        312,  37, 415, 369, 270, 184, 166, 153,  72, 267,  31, 205, 109,
        241, 350, 168, 218, 225, 176,   4,  79,   8, 234, 150, 101, 216,
        187, 160,  42, 148, 252, 169, 149, 184, 215, 155, 128, 205,  47,
         33,  58,  53, 214, 149, 103,  60,  97,  12, 234, 190,  38, 185,
        246,  20,  51, 190,  35,  53,  96,  16,  39,  57,  61,  88, 197,
         42, 150, 126, 151,  15,  26,  15,  29, 162,  21,  38,  52, 165,
         23,  40, 159,  30, 155,   0,  21,  24,  22,  10,  28, 111,   7,
```

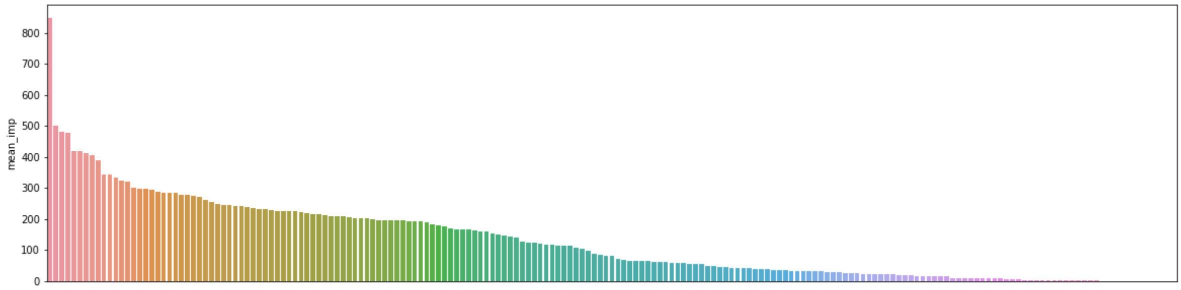


```
          1,  16,   0,   1,   0,  21,   8, 154,   0,   9,   2,  31,   1,
          0,   6,   3,   0,  25,   3,   1,   0,   1,   0,  21, 133, 190,
        198, 149, 195, 160,   0,  85, 196,  36,   7, 163, 108,  42, 161,
        111,   5,   0,  26,   0,  55, 156, 187, 203,  26,  26, 237,   4,
         61, 142, 147,  45, 331,  48,   2, 111,   5,   3,  10,  65,   9,
         28,  44,  50,  54, 144, 186,   0,  73,   2,   8,   0,  25,   1,
         35,  81, 128,   0, 160,  35, 132], dtype=int32),
 array([776, 300, 177, 172, 222, 400, 136, 362, 300, 378, 104, 372, 289,
        274,  38, 399, 359, 237, 179, 146, 157,  62, 211,  42, 165, 108,
        219, 295, 133, 215, 208, 161,   9,  74,  10, 192, 124,  85, 205,
        158, 140,  41, 145, 229, 146, 117, 157, 213, 133, 111, 176,  45,
         23,  64,  44, 201, 153,  80,  69, 101,  21, 201, 113,  42, 133,
        163,  18,  52, 174,  38,  50, 105,  21,  37,  52,  39,  83, 188,
         44, 148, 106, 126,   4,  22,  23,  26, 155,  19,  27,  50, 122,
         16,  31, 140,  33,  97,   1,  20,   9,  19,  10,  21, 103,   6,
          3,   9,   1,   0,   0,   7,  10, 116,   2,   9,   0,  22,   1,
          2,   0,   2,   0,  33,   3,   0,   1,   1,   0,  16, 118, 170,
        144, 120, 179, 116,   0,  63, 154,  25,  16, 105, 116,  42, 176,
        110,  10,   0,  31,   0,  54, 136, 145, 148,  19,  24, 145,   2,
         65, 140, 113,  46, 285,  42,   1,  84,   5,   4,   6,  46,   9,
         24,  35,  25,  55, 141, 162,   0,  78,   4,   5,   0,  16,   1,
         38,  67,  93,   0, 142,  17,  76], dtype=int32)]
```
]

In [15]:

```
df_feat_imp = pd.DataFrame(index=FEATURES)
df_feat_imp["imp0"] = feature_importances[0]
df_feat_imp["imp1"] = feature_importances[1]
df_feat_imp["imp2"] = feature_importances[2]
df_feat_imp["imp3"] = feature_importances[3]
df_feat_imp["imp4"] = feature_importances[4]
df_feat_imp["mean_imp"] = df_feat_imp.mean(axis=1).values

df_feat_imp = df_feat_imp.sort_values(by="mean_imp",ascending=False)

df_feat_imp.to_csv("feat_imp.csv")

fig, ax = plt.subplots(figsize=(20,5))
sns.barplot(x=df_feat_imp.index,y=df_feat_imp["mean_imp"])
plt.xticks([])
print(df_feat_imp)

#del df_feat_imp, feature_importances
#gc.collect()
```

```
       imp0  imp1  imp2  imp3  imp4  mean_imp
P_2     876   986   861   744   776     848.6
D_43    555   642   502   430   378     501.4
D_46    542   591   463   415   399     482.0
S_3     479   588   479   445   400     478.2
B_4     416   467   458   375   372     417.6
...     ...   ...   ...   ...   ...       ...
D_109     0     0     0     0     0       0.0
R_18      0     0     0     0     0       0.0
D_137     0     0     0     0     0       0.0
D_143     0     0     0     0     0       0.0
R_23      0     0     0     0     0       0.0

[189 rows x 6 columns]
```



08. 테스트 데이터 셋 확인 및 예측

목차로 이동하기

데이터 로드 및 RAM 사이즈 줄이기

In [16]:

```
df_test = pd.read_parquet("/kaggle/input/amex-data-integer-dtypes-parquet-format/test.parquet")

print("convert float32 columns to float16")
for col in df_test[df_test.columns[df_test.dtypes=="float32"]]:
    df_test[col] = df_test[col].astype("float16")

print("date and time")
df_test["S_2"] = pd.to_datetime(df_test["S_2"])
df_test["days"] = (df_test["S_2"] - df_test.groupby(["customer_ID"])["S_2"].transform("min")).dt.day

print("grouping")
df_test = df_test.groupby(["customer_ID"]).tail(1).set_index('customer_ID')
```

```
convert float32 columns to float16
date and time
grouping
```

5개의 모델로 예측 후, 예측 내용에 대한 평균

In [17]:

```
print("prediction")
pred=[]
for fold in range(5):
    print('FOLD:',fold)

    if len(pred)==0:
        pred = models[fold].predict_proba(df_test.drop(["S_2"],axis=1))[:,1]
    else:
        pred += models[fold].predict_proba(df_test.drop(["S_2"],axis=1))[:,1]


pred = pred/5
```

```
prediction
FOLD: 0
FOLD: 1
FOLD: 2
FOLD: 3
FOLD: 4
```

09. 제출

목차로 이동하기

In [18]:

```
subm = pd.read_csv("/kaggle/input/amex-default-prediction/sample_submission.csv")
subm["prediction"] = pred
subm.to_csv("submission.csv", index=False)
```