최종 발표

American Express - Default Prediction

MST
박규리
박종민
백진선
최가은

A Table of Contents.

1

대회 소개

Part 1,

대회 소개

American Express – Default Prediction

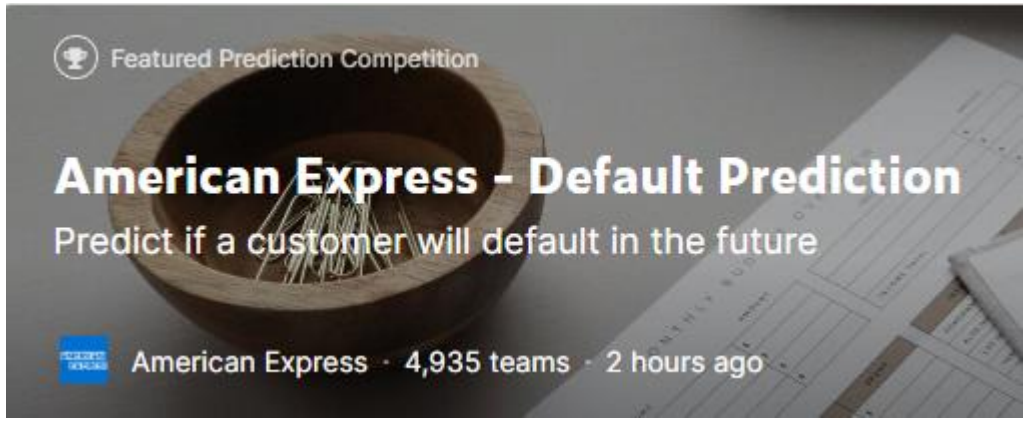
American Express는 글로벌 통합 결제 기업입니다.

세계 최대 결제 카드 발행사인 이 회사는 고객에게 삶을 풍요롭게 하고 비즈니스 성공을 구축하는 제품, 통찰력 및 경험에 대한 액세스를 제공합니다.

- 신용 부도 예측은 소비자 대출 사업에서 위험 관리에 핵심.
- 이 예측은 대출자들이 대출 결정을 최적화할 수 있게 해주며,

이는 더 나은 고객 경험과 건전한 비즈니스 경제로 이어짐.

→ <u>머신러닝 기술을 적용해 신용불량자를 예측</u>



2

데이터 분석

Part 2,

데이터 분석

Data Analysis

Discussion

Kaggle의 discussion을 읽고 정리

EDA

변수들의 특성을 EDA로 정리

데이터 분석

Discussion

How To Reduce Data Size

https://www.kaggle.com/competitions/amex-default-prediction/discussion/328054

1 데이터 유형별 용량 감소

- Column Costomer_ID [64 → 4 byte]
- Column S_2 [10 → 3 byte]
- 11 Categorical Columns [88 → 11 byte]
- 177 Numeric Columns
  [1416 → 353 byte]

2 파일 형식 선택

- csv (행 별 저장) -> parquet (열 별 저장)
- Feather, Parquet, Pickle 형식 데이터 압축
- dtype 기억 여부 업데이트

3 파일 저장 방식

여러 개의 파일

OR

하나의 큰 파일

4 Noise 제거

float32(4byte) → int8(1byte)

데이터 분석

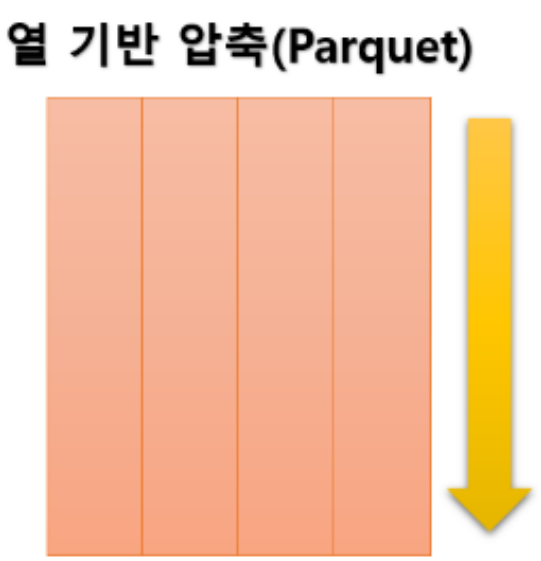
Discussion

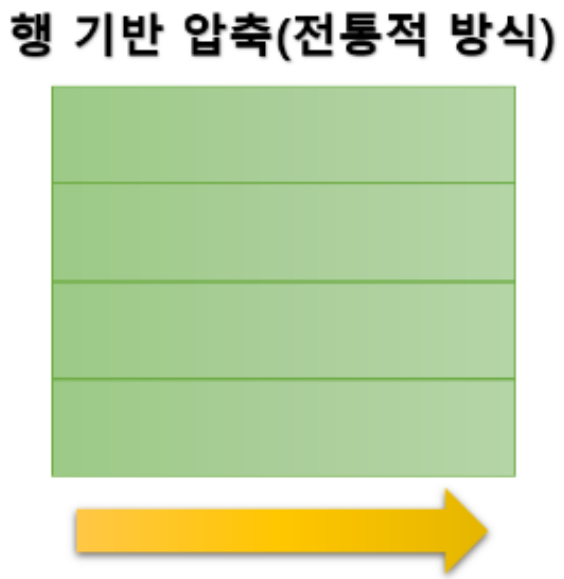
Parquet

- 데이터를 저장하는 방식 중 하나
- 빅데이터 처리는 많은 시간과 비용이 들어가서 **빠르게 읽어야하고, 압축률이 좋아야하고, 특정언어에 종속되지 않아야한다.**
- 데이터를 컬럼 단위로 압축시키고, 필요한 컬럼만 빠르게 읽고, 집계하는데 빠르다.

장점

- 압축률이 좋다. 컬럼 단위로 구성하면 데이터가 균일하게 압축률이 높아 파일의 크기도 작다.
- 컬럼 단위로 데이터가 저장되어 필요한 컬럼만 읽어서 디스크 IO가 적다.
- 컬럼별로 적합한 인코딩을 사용할 수 있다.





데이터 분석

Discussion

Parquet 실제 사용

- train.parquet, test.parquet 를 pd.read_parquet으로 읽어와서 처리
- pyarrow
  - 대용량 파일을 읽을 수 있는 라이브러리
- to_parquet(engine='pyarrow', compression='gzip')

```
train_num_agg = train.groupby('customer_ID')[num_features].agg(['mean', 'std', 'mi
n', 'max', 'last'])
train_num_agg.columns = ['_'.join(x) for x in train_num_agg.columns]

train_cat_agg = train.groupby('customer_ID')[categorical_features].agg(['count', 'la
st', 'nunique'])
train_cat_agg.columns = ['_'.join(x) for x in train_cat_agg.columns]

train_target = (train.groupby('customer_ID').tail(1).set_index('customer_ID', drop=T
rue).sort_index()['target'])

train = pd.concat([train_num_agg, train_cat_agg, train_target], axis=1)

train.to_parquet('./data/test/train_agg.parquet', engine='pyarrow', compression='gzi
p')
```

train agg to parquet

```
test_num_agg = test.groupby('customer_ID')[num_features].agg(['mean', 'std', 'min',
'max', 'last'])
test_num_agg.columns = ['_'.join(x) for x in test_num_agg.columns]

test_cat_agg = test.groupby('customer_ID')[categorical_features].agg(['count', 'las
t', 'nunique'])
test_cat_agg.columns = ['_'.join(x) for x in test_cat_agg.columns]

test = pd.concat([test_num_agg, test_cat_agg], axis=1)

test.to_parquet('./data/test/test_agg.parquet', engine='pyarrow', compression='gzi
p')
```

- train (+target): 911 MB
- test: 1.62 GB

Normalized Gini Coefficient (G). Default Rate (D).

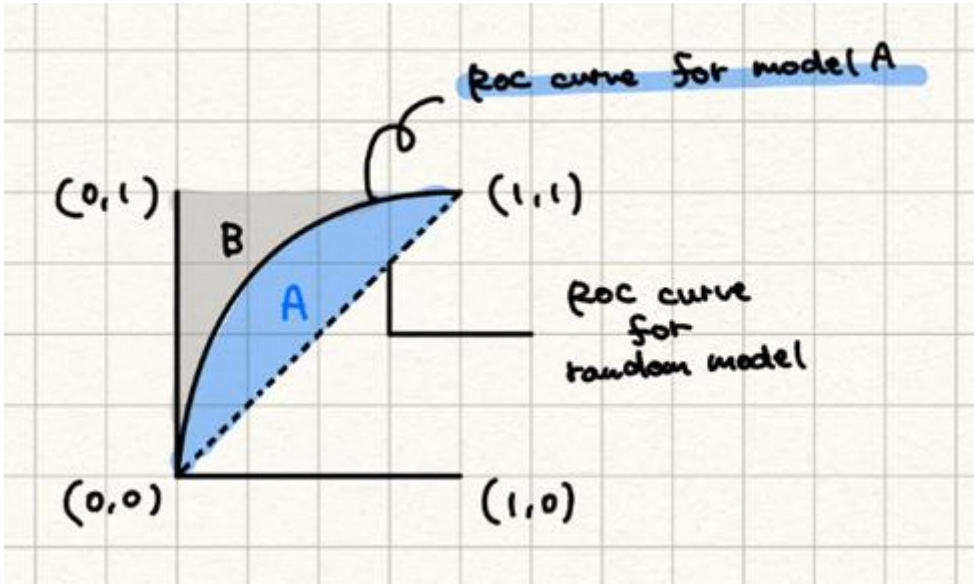
https://www.kaggle.com/competitions/amex-default-prediction/discussion/327116
https://www.kaggle.com/competitions/amex-default-prediction/discussion/333338

$$M = 0.5 \times (G + D)$$

G = Normalized Gini Coefficient
D = Default Rate

$$Gini = A$$

$$Normalized\,Gini = \frac{Gini_{model\,A}}{Gini_{perfect\,model}} = \frac{A}{A+B} = 2A$$



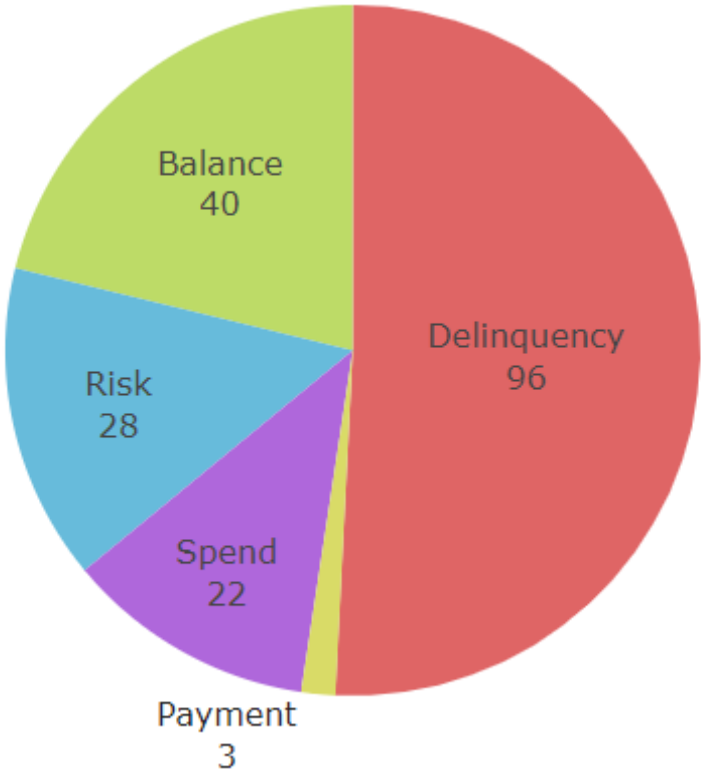
- 지니계수의 범위 : -0.5 ~ 0.5, 정규화된 지니계수의 범위 : -1 ~ 1
- 정규화된 지니계수의 목적 : AUC(랜덤 모델의 경우 = 0, 완전 모델의 경우 1)를 스케일링하는 것

Part 2,

데이터 분석

EDA : D_* = Delinquency variables (연체 변수)

변수 개수



Target 분포

Part 2

데이터 분석

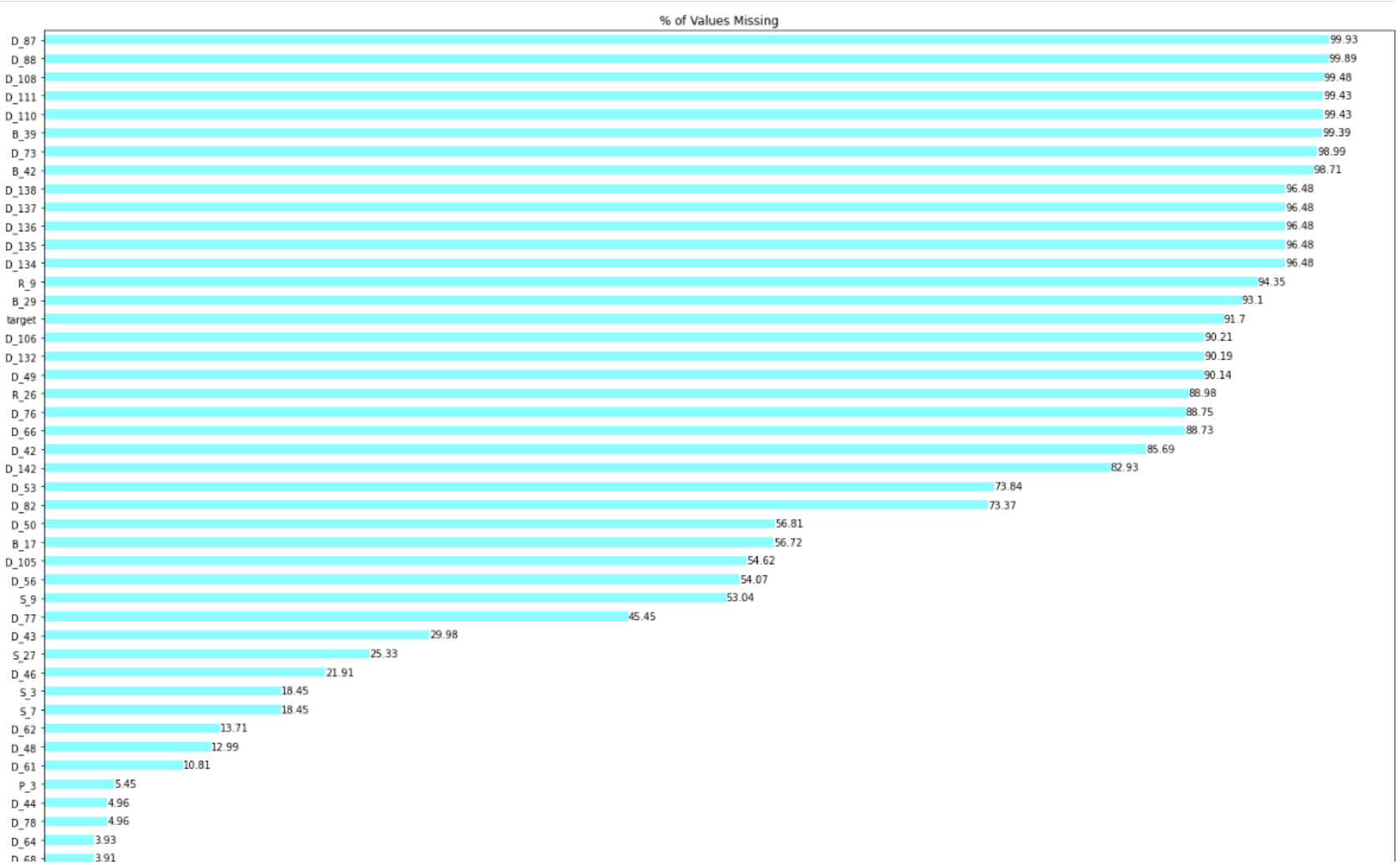
EDA : D_* = Delinquency variables (연체 변수)

[4]:

```
"""What are the counts of missing values
def missing(df):
        ncounts = (pd.DataFrame([df.isnul
        ncounts = ncounts.rename(columns=
        margin = 0.60
        #width = (1.-2.*margin)/len(trair
        ax=ncounts.sort_values('train_mis
            kind="barh", figsize=(20, 50)
        )

        ax.bar_label(ax.containers[0])
        plt.tight_layout()
        plt.show()

missing(train_df)
```



- 결측치

결측치가 있는 D 변수가 많음

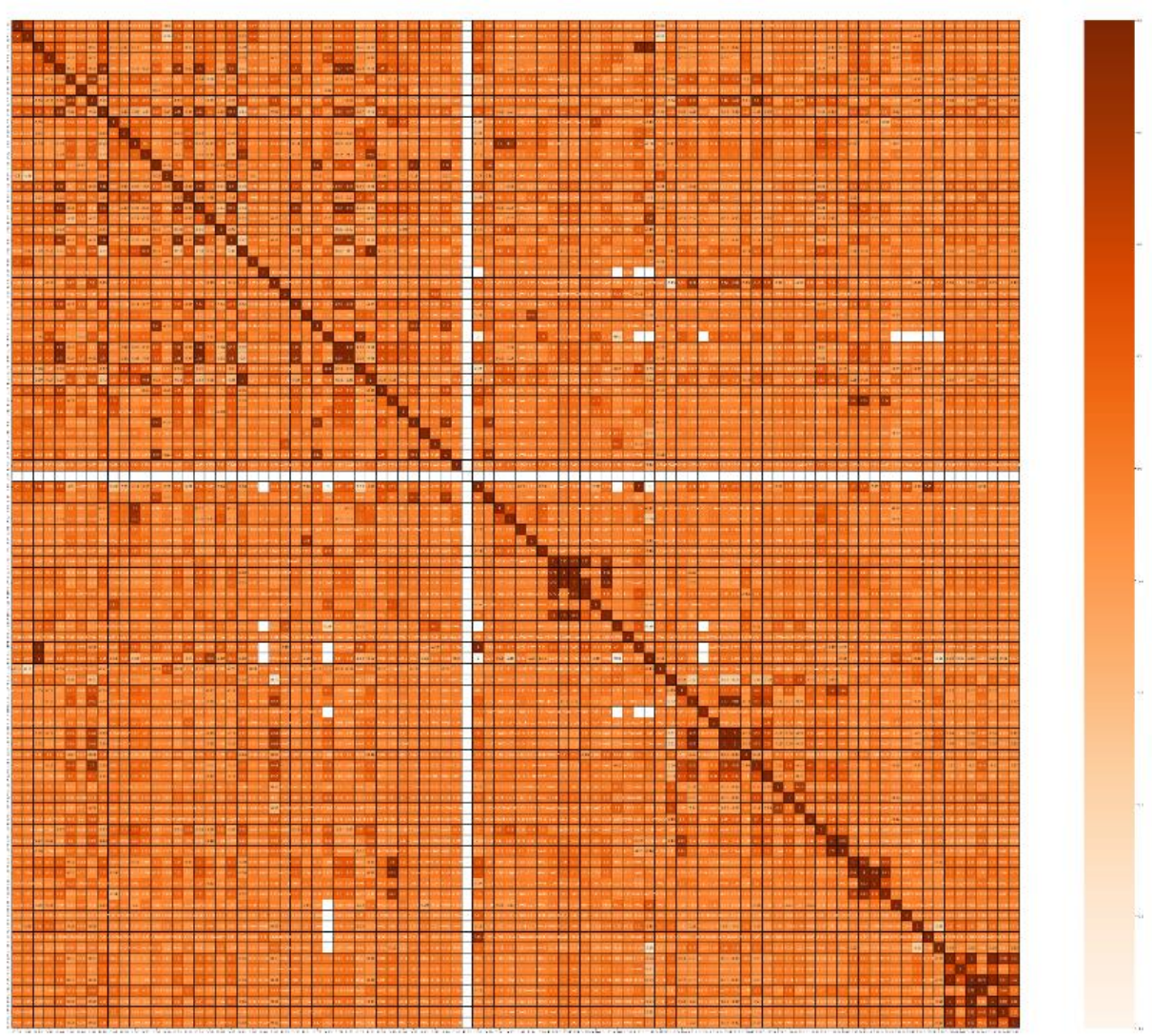
Part 2,

데이터 분석

EDA : D_* = Delinquency variables (연체 변수)

In [36]:

```
#plot correlation between D variables
corr = train_df[D_columns].corr(method='pearson')
plt.figure(figsize=(75,60))
# mask option can be used to not show values <=0.90
#sns.heatmap(corr,vmax=.8,linewidth=.01, square = True, annot = T
90))
sns.heatmap(corr,vmax=.8,linewidth=.01, square = True, annot = T
plt.show()
```



- 상관관계

D_42는 D_110, D_111과 1.0의 상관관계를 갖는다.

D_58은 D_74와 0.92, D_75와 0.93의 상관관계를 갖는다.

D_62는 D_77과 1.0의 상관관계를 갖는다.

D_74는 D_75와 0.99의 상관관계를 갖는다.

D_73은 D_88과 -1.0의 상관관계를 갖는다.

D_139는 D_143 및 D_141과 1.0의 상관관계를 갖는다.

D_141은 D_143과 1.0의 상관관계를 갖는다.

Part 2,

데이터 분석

EDA : S_* = Spend variables (소비 변수)

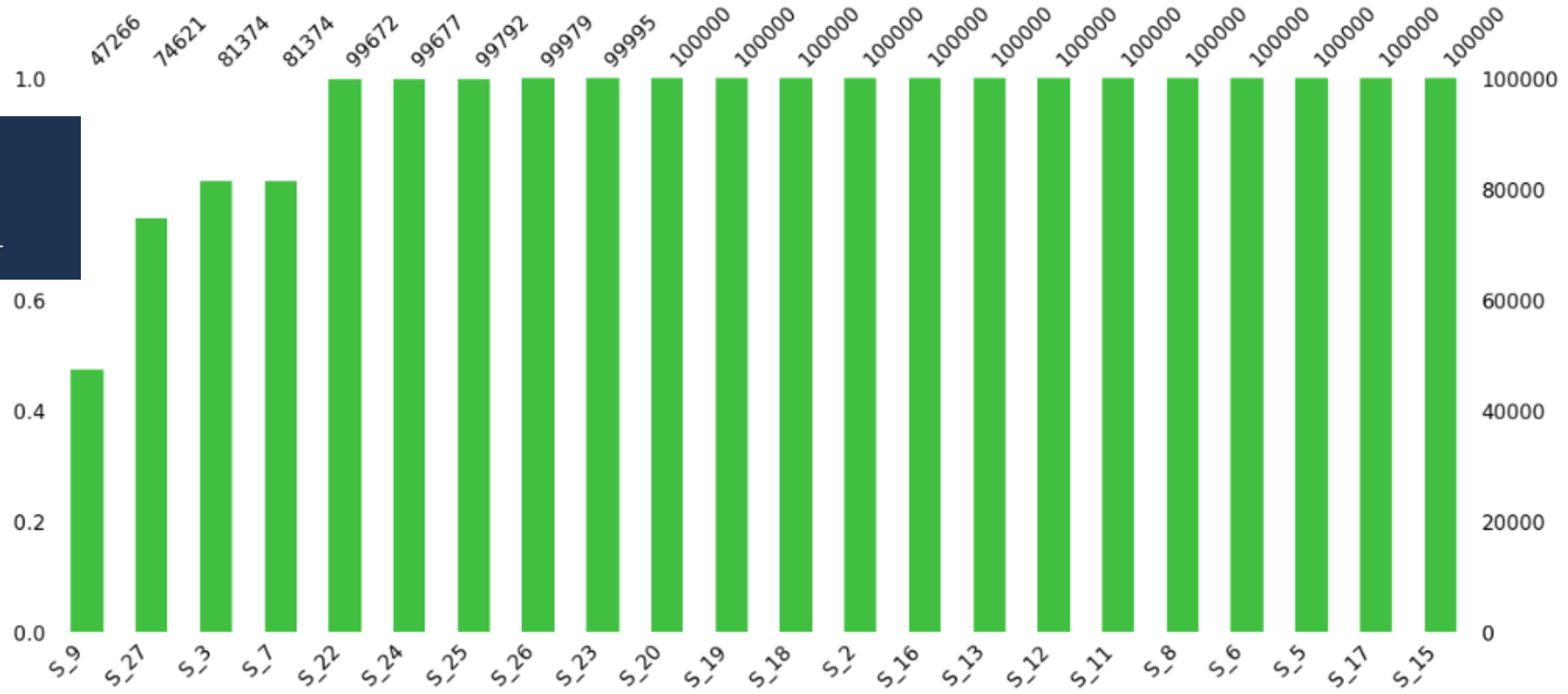
In [41]:

```
#plot of missing values for S variables
msno.bar(train_df[S_columns],figsize=(20,8), sort="ascending", color=(0.25,0.75,0.25))
```

- 결측치

S_9, S_27, S_3, S_7

→ 상대적으로 결측치가 많음



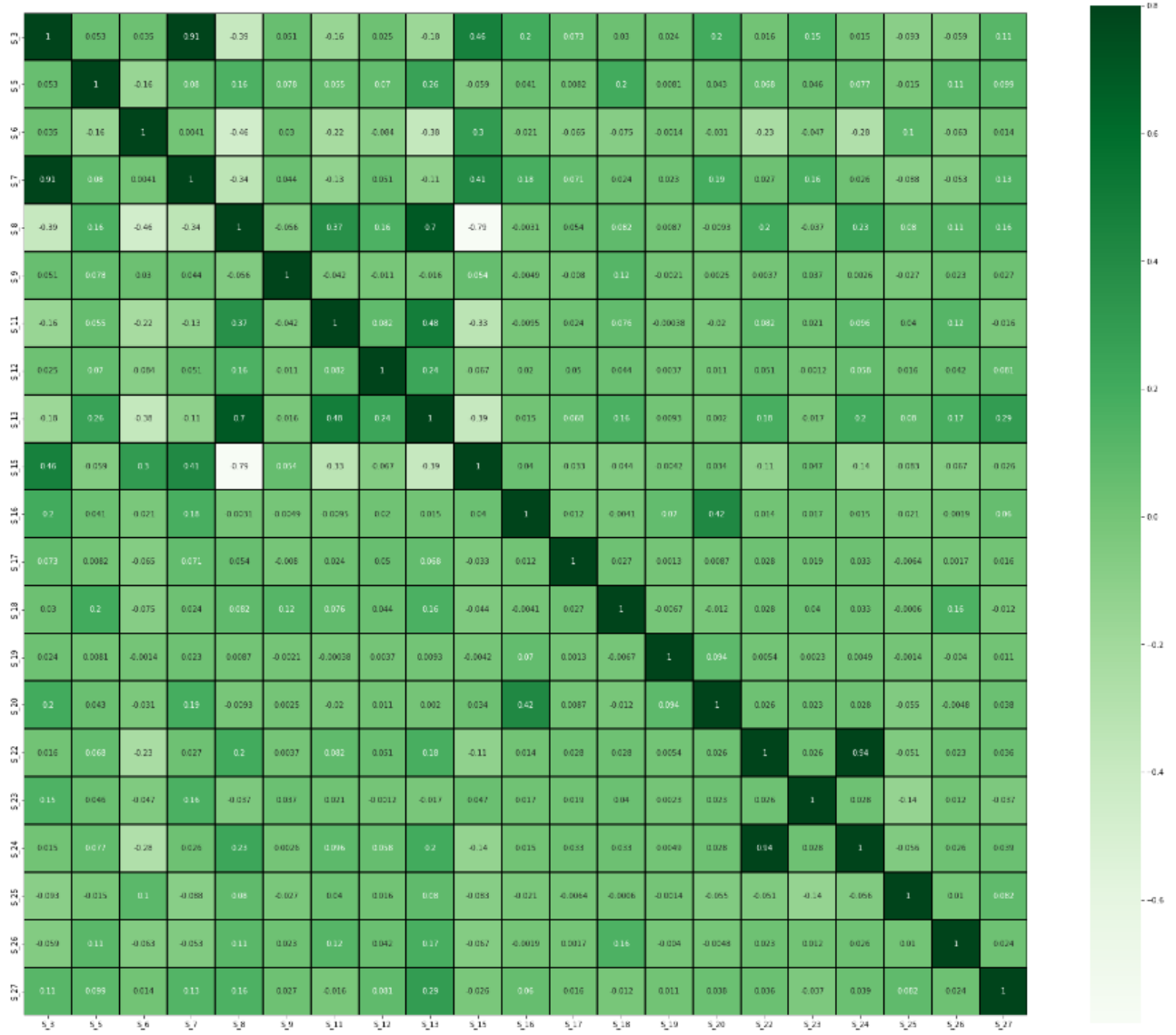
Part 2,

데이터 분석

EDA : S_* = Spend variables (소비 변수)

In [43]:

```
#plot correlation between S variables
corr = train_df[S_columns].corr(method='pearson')
plt.figure(figsize=(30,25))
# mask option can be used to not show values <=0.90
#sns.heatmap(corr,vmax=.8,linewidth=.01, square = True, annot = True,cmap='YlGnBu',linecolor ='black',mask = (np.abs(corr) <= 0
90))
sns.heatmap(corr,vmax=.8,linewidth=.01, square = True, annot = True,cmap='Greens',linecolor ='black')
plt.show()
```



- 상관관계

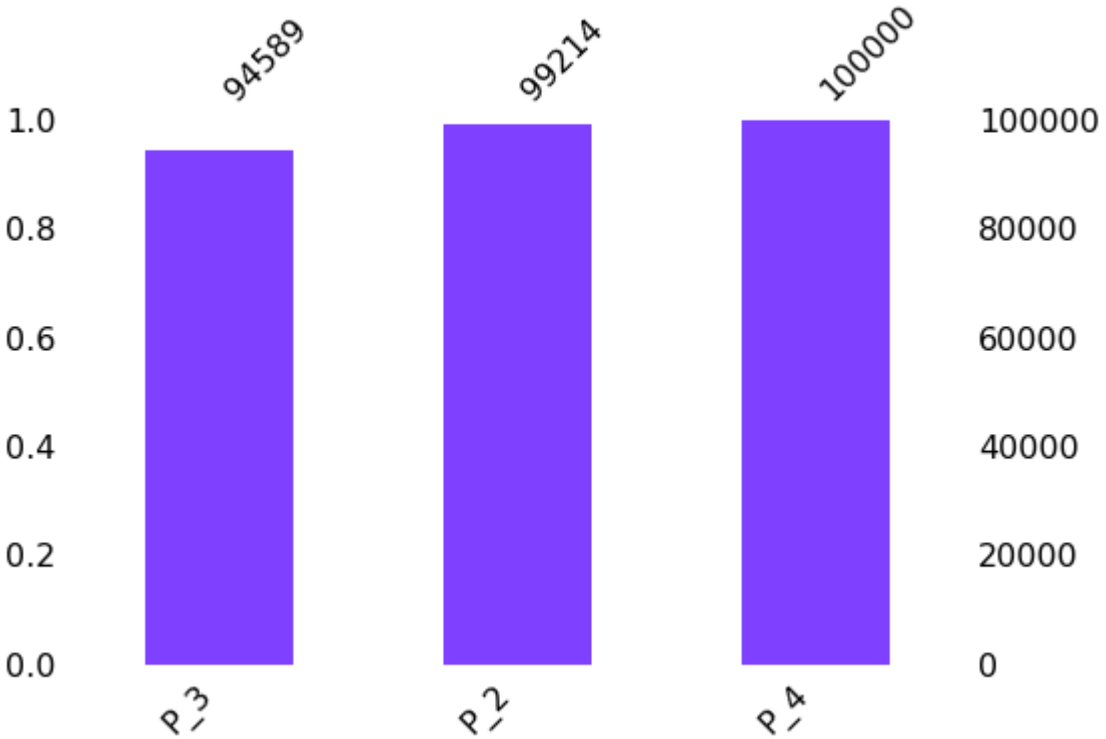
S_3와 S_7의 상관관계 0.91

S_22와 S_24의 상관관계 0.94

Part 2,

데이터 분석

EDA : P_* = Payment variables (지급 변수)

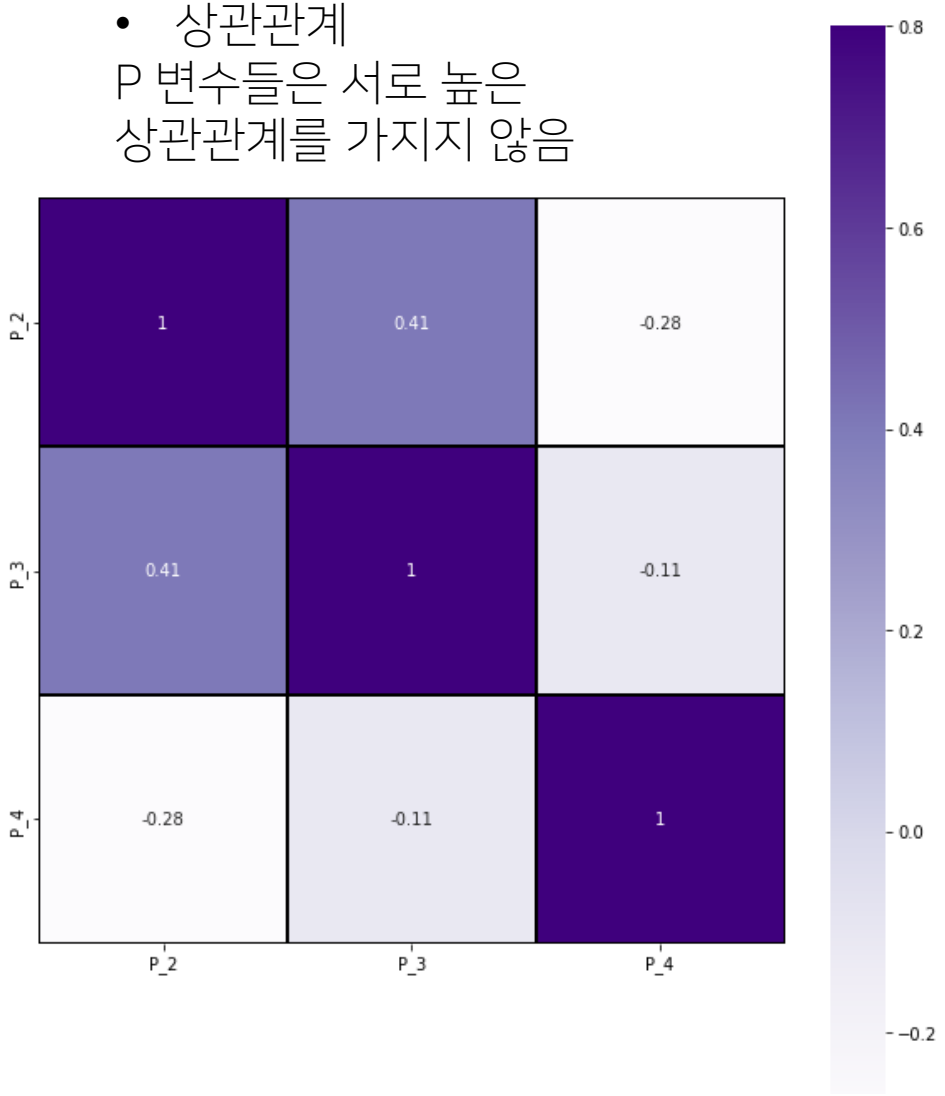


- 결측치

P_3과 P_2에 결측치 있음

- 상관관계

P 변수들은 서로 높은 상관관계를 가지지 않음



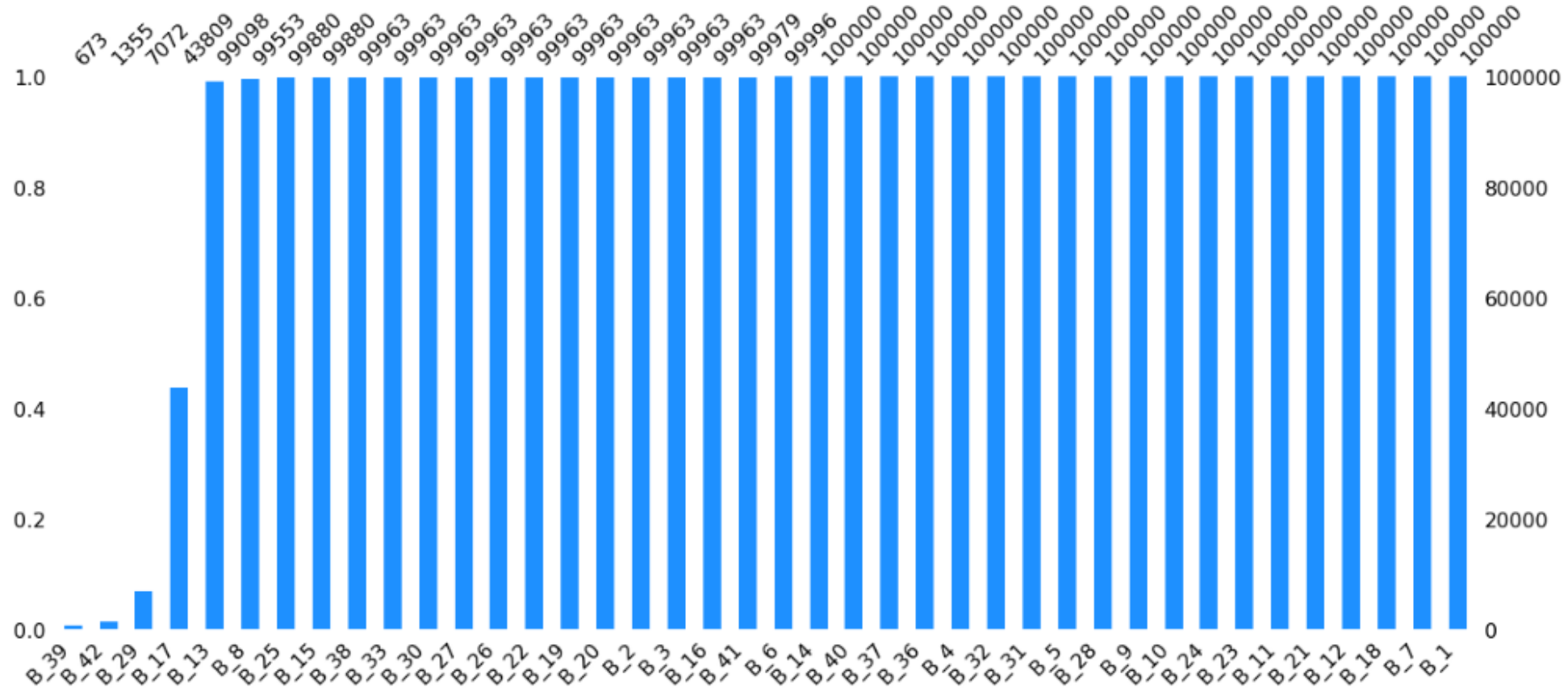
Part 2,

데이터 분석

EDA : B_* = Balance variables (균형 변수)

- 결측치

B_17, B_29, B_42, B_39
→ 많은 결측치를 가짐



Part 2,

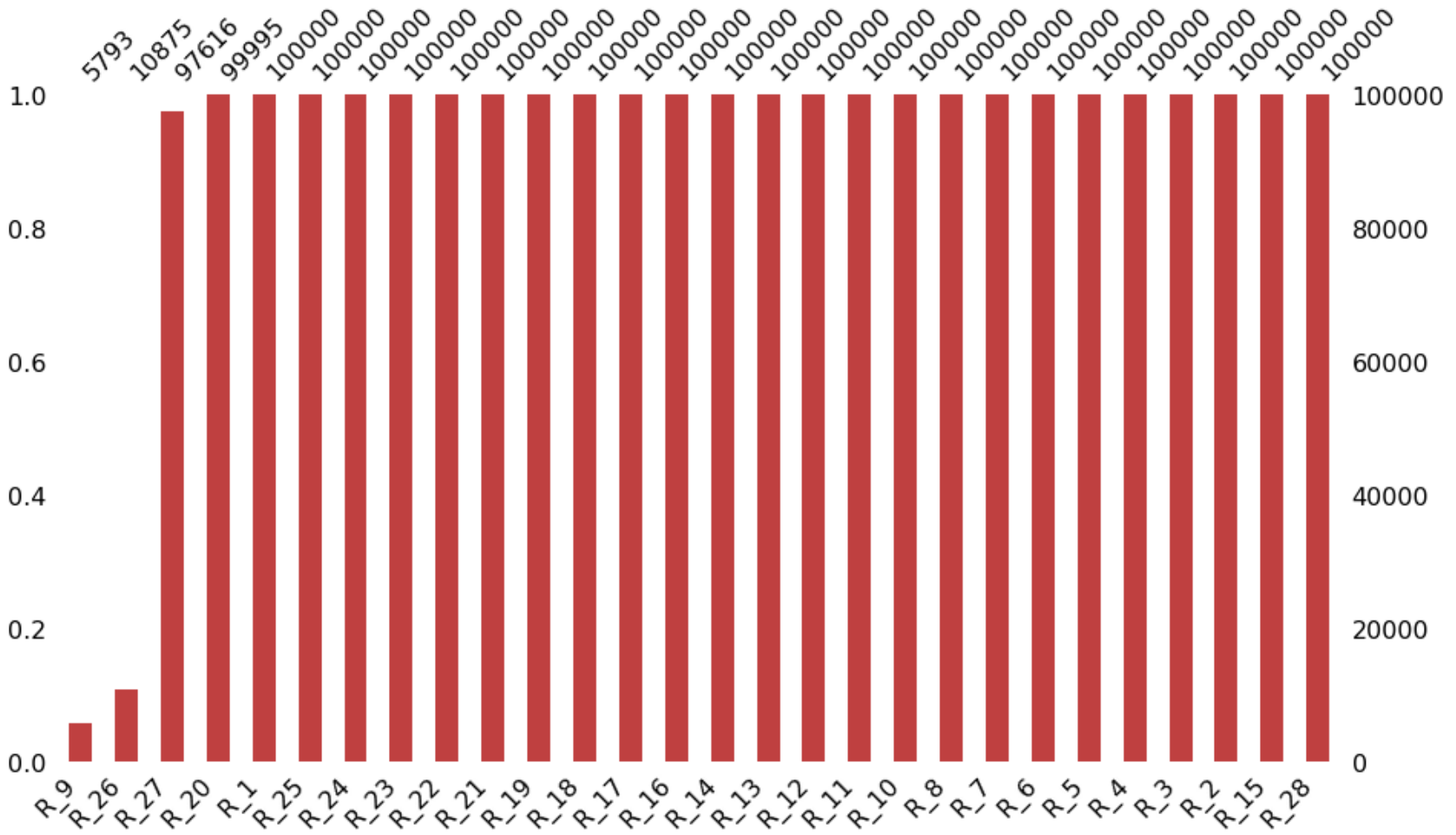
데이터 분석

EDA : R_* = Risk variables (위험 변수)

- 결측치

R_9, R_26

→ 많은 결측치를 가짐



3

모델 및 코드 설명

Part 3,

사용한 모델

Xgboost, Lightgbm, Catboost, ensemble

Xgboost

- 결측치 -127 처리
- 연속형 변수
(mean, std,
min, max, last)
- 범주형 변수
(count, last, nunique)

Score

0.794

Lightgbm

Customer_ID 당
마지막 데이터
포인트만 사용

Score

0.787

Catboost

- 연속형 변수
(mean, std,
min, max, last)
- 범주형 변수
(count, last, nunique)
X_last LabelEncoding

Score

0.794

Lightgbm - feature

변수추가
+ days 특성 추가

Score

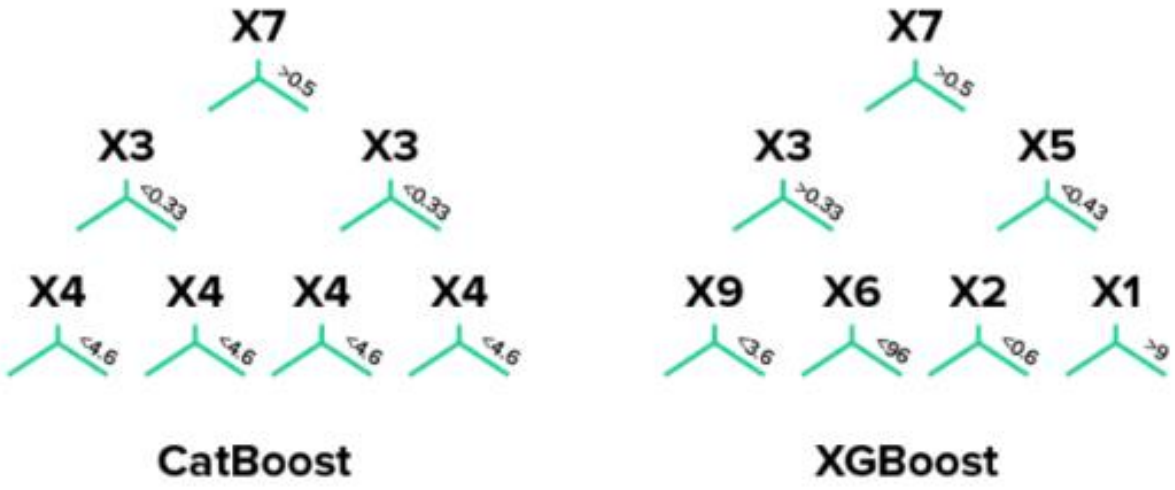
0.78407

Lightgbm , Catboost

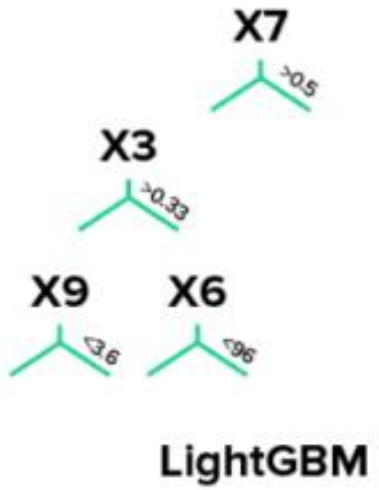
Catboost

Lightgbm,Catboost

Level wise(균형)트리 분할 방식



Leaf wise(리프 중심)트리 분할



CatBoost

- 대칭적인 트리 형성
- 범주형 변수로 이루어진 데이터셋에서 예측 성능이 우수함

소스 코드

Lightgbm

Lightgbm

[3] :

```
def features_process(df, test=False):
    print('days')
    df['S_2'] = pd.to_datetime(df['S_2'])
    df['days'] = (df['S_2'] - df.groupby(['customer_ID'])['S_2'].transform('min')).dt.days.astype('int16') + 1

    print('groupby customer_ID tail(1)')
    df = df.groupby('customer_ID').tail(1).set_index('customer_ID')
    print('shape:', df.shape)

    if not test:
        print('dropna nan >= 80%')
        df = df.dropna(axis=1, thresh=int(0.8 * len(df)))
        print('shape:', df.shape)

    print('feature process')
    df["c_PD_239"] = df["D_39"] / (df["P_2"] * (-1) + 0.0001)
    df["c_PB_29"] = df["P_2"] * (-1) / (df["B_9"] * (1) + 0.0001)
    df["c_PR_21"] = df["P_2"] * (-1) / (df["R_1"] + 0.0001)

    df["c_BBBB"] = (df["B_9"] + 0.001) / (df["B_23"] + df["B_3"] + 0.0001)
    df["c_BBBB1"] = (df["B_33"] * (-1)) + (df["B_18"] * (-1) + df["S_25"] * (1) + 0.0001)
    df["c_BBBB2"] = (df["B_19"] + df["B_20"] + df["B_4"] + 0.0001)

    df["c_RRR0"] = (df["R_3"] + 0.001) / (df["R_2"] + df["R_4"] + 0.0001)
    df["c_RRR1"] = (df["D_62"] + 0.001) / (df["D_112"] + df["R_27"] + 0.0001)

    df["c_PD_348"] = df["D_48"] / (df["P_3"] + 0.0001)
    df["c_PD_355"] = df["D_55"] / (df["P_3"] + 0.0001)

    df["c_PD_439"] = df["D_39"] / (df["P_4"] + 0.0001)
    df["c_PB_49"] = df["B_9"] / (df["P_4"] + 0.0001)
    df["c_PR_41"] = df["R_1"] / (df["P_4"] + 0.0001)
    print('shape:', df.shape)

    return df
```

S_2이용해서 days 만들기

Customer_ID 당 마지막 데이터 포인트 사용

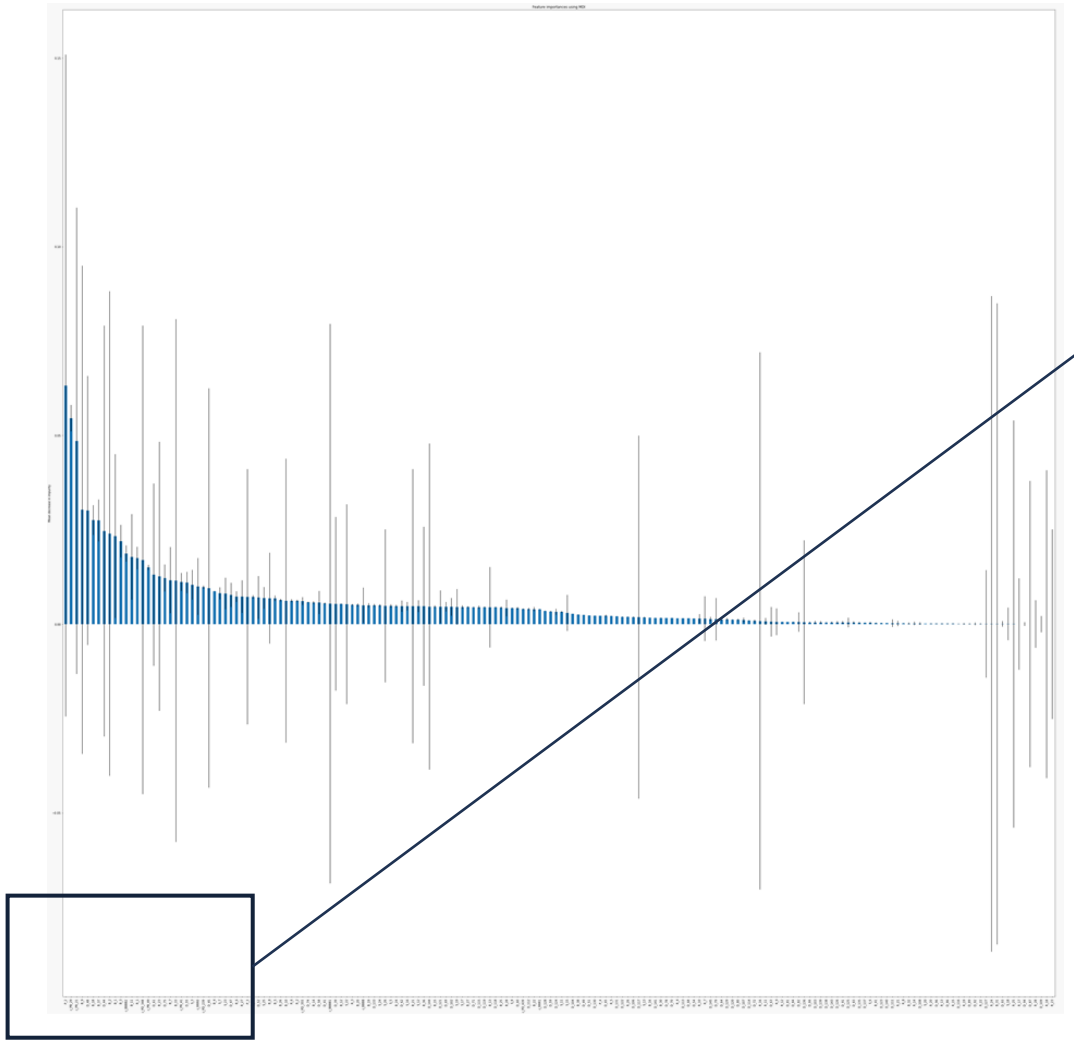
결측치 80% 이상인 특성 제거

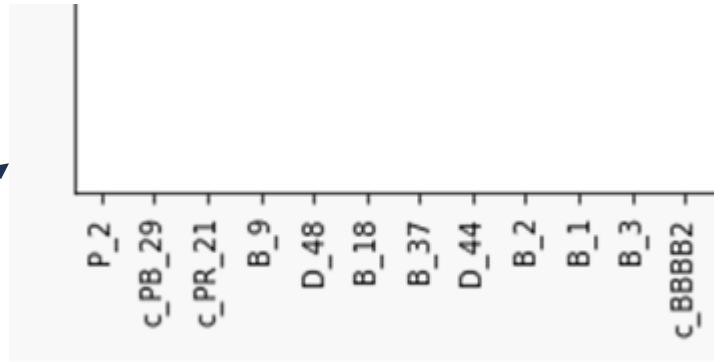
새로운 특성 추가

소스 코드

Lightgbm

Lightgbm 특성 중요도





중요도가 높은 특성

P_2
c_PB_29
c_PR_21
B_9
D_48
B_18
...

Part 3,

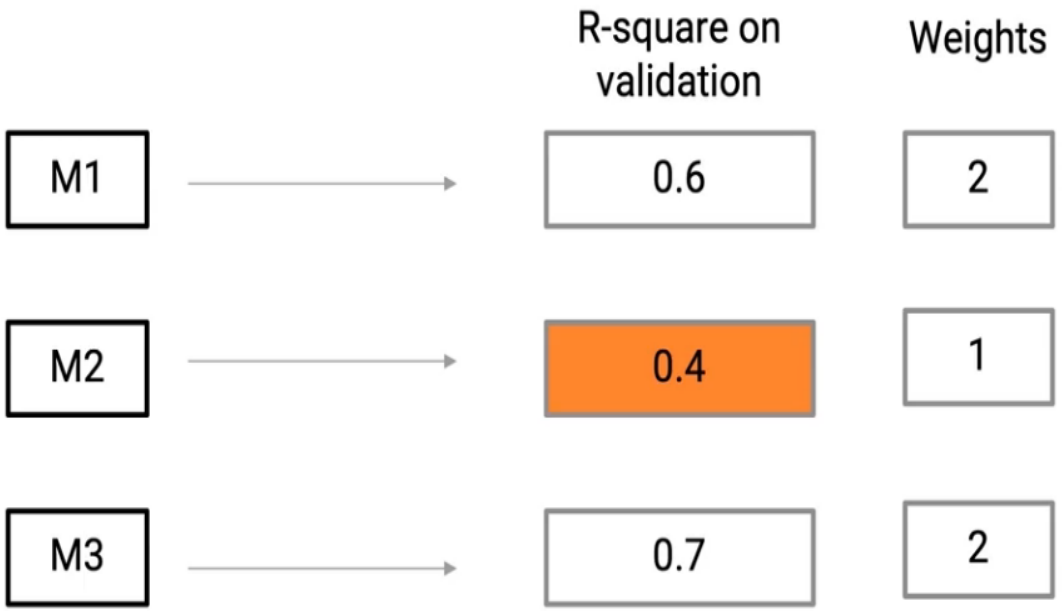
ensemble

Part 3,

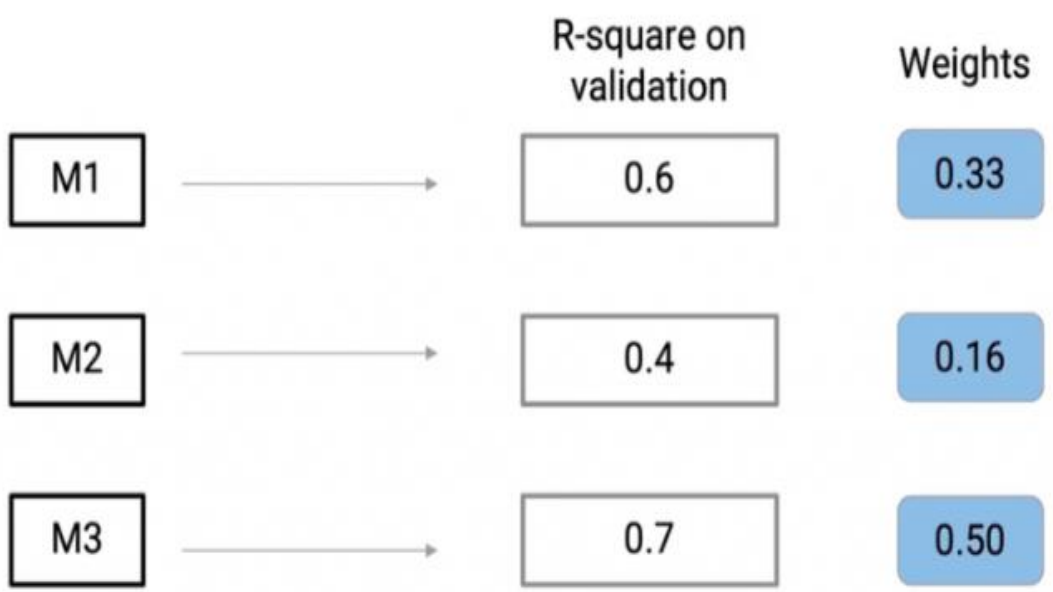
Ensemble

Ensemble Technique : Weighted Averaging, Rank Averaging

Weighted Averaging



Rank Averaging



Ensemble

Weighted Average, Rank Average

Ensemble

모델들의 예측값 불러오기

[11]:

```
pred_paths = sorted([x for x in glob.glob('../input/*/submission_*.csv')])
pred_paths = pred_paths[:-1]
pred_paths
```

모델들의 자체평가 점수 불러오기

```
score_paths = sorted([x for x in glob.glob('../input/*/score_*.txt')])
score_paths = score_paths[:-1]
score_paths
```

```
weights = []
for path in score_paths:
    with open(path, 'r') as f:
        score = float(f.readline())
    weights.append(score)
weights
```

Part 3,

Ensemble

Weighted Average

Ensemble-Weighted Average

```
submit = pd.read_csv('../input/amex-default-prediction/sample_submission.csv')
submit['prediction'] = 0
submit.sort_values(by='customer_ID').reset_index(drop=True)

for df, weight in zip(dfs, weights):
    submit['prediction'] += (df['prediction'] * weight)

submit['prediction'] /= np.sum(weights)

submit.to_csv('submission_ensemble.csv', index=None)
```

Ensemble-Rank Average

```
from scipy.stats import rankdata

submit = pd.read_csv('../input/amex-default-prediction/sample_submission.csv')
submit['prediction'] = 0
submit.sort_values(by='customer_ID').reset_index(drop=True)

ranking_weights = rankdata(weights)
ranking_weights /= np.sum(ranking_weights)

for df, weight in zip(dfs, ranking_weights):
    submit['prediction'] += (df['prediction'] * weight)

submit.to_csv('submission_ranking.csv', index=None)
```

Ensemble

Weighted Average, Rank Average

Weighted Average, Rank Average

Weighted Average

Score

0.79383

Rank Average

Score

0.79455

Q & A