사람은 사람을 봅니다. 이 사람이 좋은 사람인지 아닌지, 유잼인지 노잼인지, 내 마음에 드는지 안 드는지를 판단합니다. 유전자에 내재된 정보와 이번 생의 경험은 우리에게 감정을 제공합니다. 0에서 1 스케일로 나타낸다면, 이 사람은 0.7 정도로 느껴져 좋고 저 사람은 0.2 정도로 느껴져 싫습니다. 좋고 싫음의 기준이 0.5로 정해져 있는 사람도 있고 0.7로 설정된 사람도 있겠습니다. 이 판단이 맞으면 통찰력이 있는 사람이고 틀리면 편견을 가진 사람이 됩니다. 저는 많은 사람을 판단했고 많이 틀렸습니다. 그 사람의 말, 행동, 표정, 태도, 분위기를 보고 좋은 사람인지 아닌지 결정내렸습니다. 그리고 많은 사람을 싫어했습니다. 하지만 그 사람은 누군가에게는 좋은 사람입니다. 이 사람이 좋은 사람이 되고 안 좋은 사람이 되는 기준은 단지 제게 있습니다. 기준을 0으로 만들고 싶었는데.. 실은 0.9나 되는 무척이나 까다롭고 속좁은 사람이었습니다. 제 마음을 넓히면 모든 생명을 사랑할 수 있을까요. 하지만 벌레는 도무지 안 되겠는데요. 가능성은 열어놔야하니까 안 되겠다보다는 어려운데요로 바꾸겠습니다.

오늘은 확률과 기준의 이야기입니다.

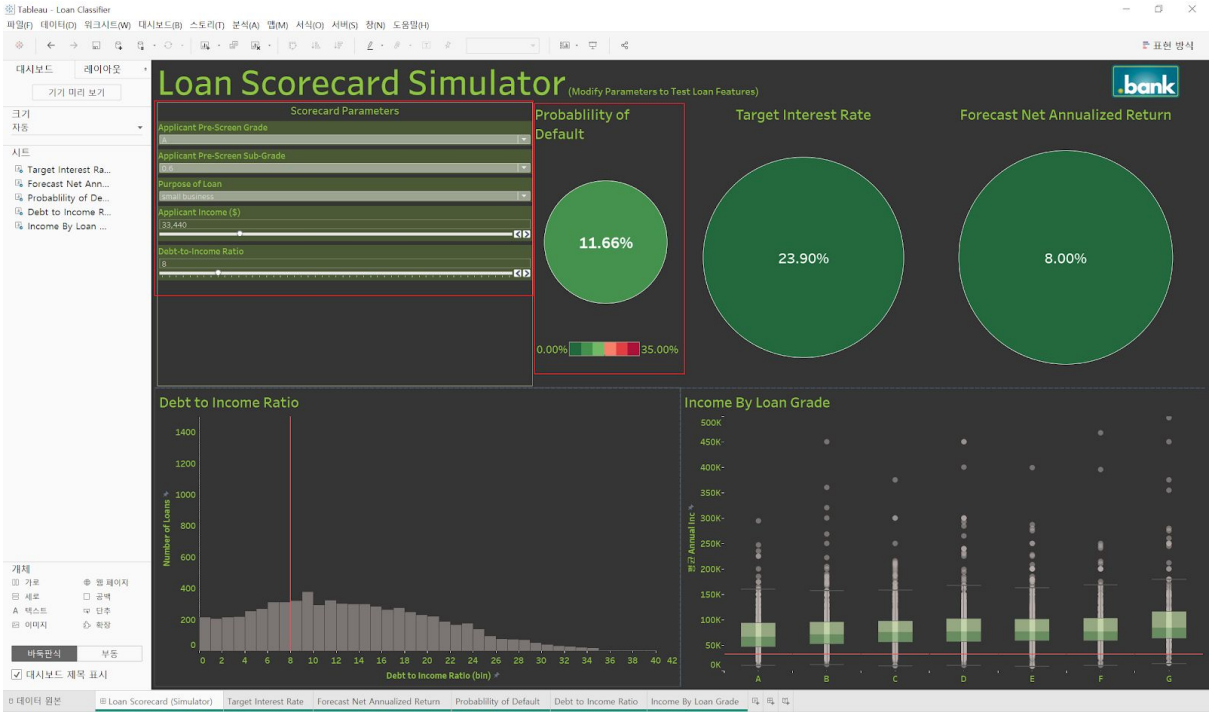
확률을 예측하고 O, X 판단을 해주는 classification model 을 알아보겠습니다.

이론은 몰라도 모델은 돌릴 수 있습니다. 하지만 어떤 data, 어떤 feature가 필요할 지, 어떤 모델, 어떤 옵션을 쓸지 선택할 수가 없겠쥬. 그런데 이론을 먼저 보는 건 아무래도 노잼입니다. 앞으로는 흥미를 끌어올릴 수 있게 모델을 돌려보고 이론을 역으로 보겠습니다.

이번엔 순서를 바꿔서 대시보드부터 보는 게 좋겠습니다.

Collection Recovery Fee	Collections 12 Mths Ex Med	Collections 12 Mths Zero	Debt to Income Ratio	Delinq 2Yrs	Delinq 2Yrs Zero	Failure %	Funded Amnt	Funded Amnt Inv	Inq Last 6Mths	Installment	Int Rate	Last Delinq None	Last Major Derog None	Last Pymnt Amnt	Last Record None
0.000	0	1	17.6300	0	1	0.00000	8,875	8,875	2	316.86	17.1000	1	1	316.86	1
0.000	0	1	5.7800	0	1	0.00000	3,600	3,600	1	128.53	17.1000	0	0	128.53	1
0.000	0	1	13.3400	0	1	0.00000	9,900	9,900	0	349.23	16.2400	0	0	349.23	1
0.000	0	1	25.9700	0	1	0.00000	11,850	11,850	2	418.02	16.2400	0	0	418.02	1
36.000	0	1	7.1100	1	0	1.00000	24,000	24,000	2	730.46	6.0300	0	1	200.00	1
0.000	0	1	11.1100	0	1	1.00000	19,200	19,175	1	584.37	6.0300	1	1	584.37	1
0.000	0	1	16.0000	0	1	1.00000	10,000	10,000	0	304.36	6.0300	1	1	304.36	1
0.000	0	1	9.7400	0	1	0.00000	6,000	6,000	0	182.62	6.0300	1	1	182.62	1
0.000	0	1	18.0900	0	1	0.00000	8,000	8,000	0	243.49	6.0300	1	1	243.49	1
0.000	0	1	22.1000	0	1	0.00000	19,500	19,500	1	593.50	6.0300	1	1	593.50	1
0.000	0	1	21.2900	0	1	0.00000	12,000	12,000	0	365.23	6.0300	1	1	365.23	1
0.000	0	1	23.2200	1	0	0.00000	8,325	8,325	1	253.38	6.0300	0	0	253.38	1
0.000	0	1	8.1300	0	1	0.00000	14,500	14,500	0	441.32	6.0300	1	1	441.32	1
0.000	0	1	7.9400	0	1	0.00000	13,000	13,000	0	395.67	6.0300	1	1	395.67	1
0.000	0	1	15.8400	0	1	0.00000	15,300	15,300	2	465.67	6.0300	1	1	600.00	1
0.000	0	1	12.9600	0	1	0.00000	15,000	15,000	0	456.54	6.0300	1	1	456.54	1
0.000	0	1	8.5700	0	1	0.00000	5,800	5,800	1	176.53	6.0300	1	1	176.53	1
0.000	0	1	20.4300	0	1	0.00000	13,000	13,000	1	395.67	6.0300	1	1	395.67	1
0.000	0	1	20.3700	0	1	0.00000	16,000	16,000	3	486.97	6.0300	1	1	486.97	1
0.000	0	1	29.6100	0	1	0.00000	6,000	6,000	0	182.62	6.0300	1	1	182.62	1
0.000	0	1	4.4600	0	1	0.00000	6,000	6,000	1	182.62	6.0300	1	1	182.62	1
0.000	0	1	18.3900	0	1	0.00000	24,000	24,000	0	730.46	6.0300	0	1	730.46	1
0.000	0	1	12.6600	0	1	0.00000	5,500	5,500	3	167.40	6.0300	1	1	167.40	1

data는 금융 data입니다.
회사가 대출을 해 주고 고객이 잘 갚았는지, 못 갚았는지(default)가 나와있고요.
개인 금융 정보들이 나열돼 있습니다.
연봉, 이자율, DTI, 상환내역 등 numerical column,
돈 빌린 목적(purpose), description 과 같은 text column 등이 있습니다.
전체적인 column은 첨부해드린 대시보드에서 자세히 확인하시기 바랍니다.



해당 data의 모든 column을 다 사용하진 않습니다.
여기서는 Grade, Sub-Grade, Purpose, Income, DTI 총 5개만 사용했습니다.
왼쪽 red box에서 변수들의 값을 선택할 수 있고요.
선택된 변수에 따라 계산된 확률이 Probability of Default 에 나옵니다.
이 확률에 따라 오른쪽 두 개의 원, 이자율과 연 상환율? 맞나요?을 예측해봤습니다.
태블로 안에서 해당 계산된 필드를 봐주세요.
아래는 두 가지 그래프를 제시해봤지만 단지 주요 정보일 뿐이고 예측모델과는 상관없으니 생략하겠습니다.

저는 처음 궁금했던 게,
학습하는 데 긴 시간이 걸리는데 실시간으로 대시보드에 observation 마다, record마다 확률을 계산해 보여줄 수 있을까 하는 것이였습니다.

이제 코드를 보시면 아실텐데요.
학습시킨 모델을 deploy 시켜서 데이터를 넣었을 때 바로 확률을 볼 수 있습니다.
찬찬히 보겠습니다.

```
import pandas as pd
import numpy as np
from sklearn.neural_network import MLPClassifier
from sklearn import preprocessing
from sklearn.model_selection import train_test_split
from sklearn import metrics
from sklearn.preprocessing import MinMaxScaler

pd.set_option('display.max_columns',50)
pd.set_option('display.width',500)
```

먼저 필요한 모듈을 다 import 했습니다.
필요할 때마다 import 하는 것도 내가 뭘 쓰고 있는지 인식할 수 있어서 좋은 방법입니다.
저처럼 처음부터 import 하지 말고 error 메세지를 보면서 import 하시길 권장합니다.
간략하게 새로운 모듈에 대한 설명을 덧붙이겠습니다.

MLPClassifier: 여러 가지 classifier(분류 모델) 중 신경망을 이용한 모델을 불러옵니다.
hyperparameter가 많아, 그러니까 조정할 옵션이 많아 설명드릴 게 많지만
다음에 logistic regression, SVM, tree 순으로 먼저 설명드리려고 합니다.
신경망은 마지막일텐데, 수미상관 구조를 위해 맛만 보시라고 넣었습니다.

train_test_split: 데이터를 random train, test subset 으로 나눠주는 모듈입니다.
비율을 설정할 수 있고 섞을지 말지 정할 수도 있고요.
각 subset에서 class 비율을 유지할 수도 있습니다.

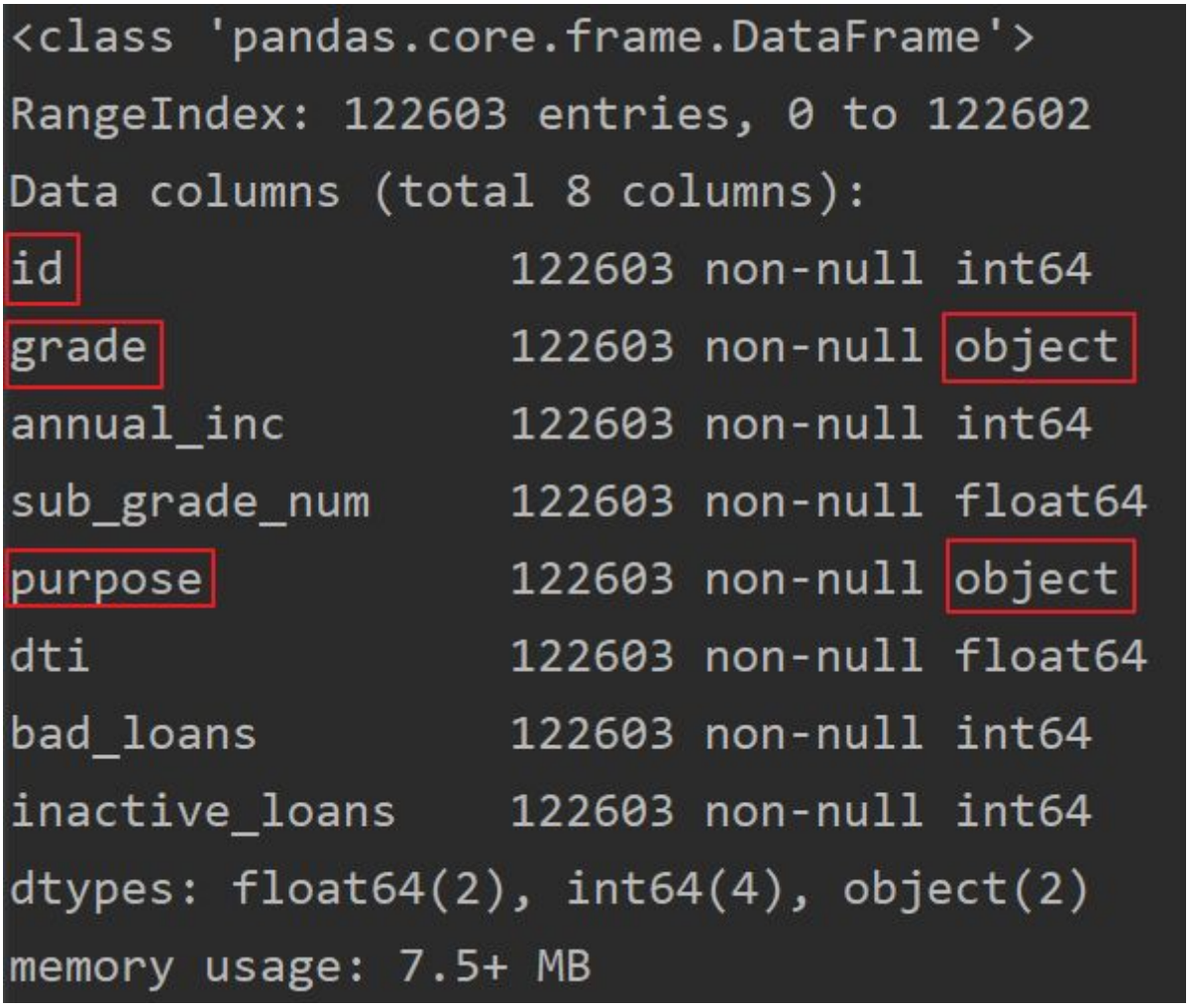
metrics: 결과를 평가할 수 있는 도구들을 제공합니다.
지금은 classification이니 confusion matrix, precision & recall + f1 score를 볼 겁니다.
이게 뭔지는 이따 말씀드릴게유.

```
#데이터 읽기
train = pd.read_csv('c:/Users/JunHyuk/downloads/TabPy materials/LoanStatsFilter.csv')

#데이터 살짝 보기
train.info()
train.head()

train['grade'].unique()
train['purpose'].unique()
train['bad_loans'].unique()
train['inactive_loans'].unique()
# bad_loans는 target
# grade, purpose는 label encoder
# id, inactive_loans 는 버리기
```

먼저 데이터를 불러와서 살펴보겠습니다.
목적을 갖고 보는 게 맞겠지유. 여태 그런 설명을 안 드렸네요. 죄송합니다.

X: 숫자만으로 된 X1, … Xp 을 feature set으로 갖길 바라고요.
Y: class가 0, 1, 2, ... , m 인 column 한 줄을 target으로 갖길 바랍니다.

이후에는 null 과 이상치를 없애거나 처리해주면 좋겠고요.



train.info()의 결과입니다.

우선 target으로 bad_loans 를 쓰면 됩니다. default(연체, 채무불이행?) 여부에 따라 0, 1로 나눠진 column인데 .unique()로 확인해보시기 바랍니다.

그럼 버릴 column을 찾아봅시다.
id는 모델을 만드는 데 도움이 안 됩니다. index로 만들면 나중에 결과 확인시 이용할 수도 있겠지만 지금은 아니네요. 없애기로 합니다.
inactive_loans 는 왜 있는지 모르겠습니다. .unique() 해보면 전부 1인데요. 무슨 뜻인지 모르겠으나 inactive_loans 중에 자료를 가져온 것 같습니다.

남은 column으로 feature set을 구성하면 되겠습니다.
grade, annual_inc, sub_grade_num, purpose, dti 총 5개 column입니다.

그 중 grade, purpose가 object 타입이네요.
unique()로 뭐가 들었는지 확인해 보세요.

```
>>> train['grade'].unique()
array(['G', 'B', 'E', 'A', 'C', 'D', 'F'], dtype=object)
>>> train['purpose'].unique()
array(['debt_consolidation', 'other', 'car', 'moving', 'credit_card',
       'vacation', 'major_purchase', 'small_business', 'house', 'wedding',
       'medical', 'home_improvement'], dtype=object)
```

grade는 A-G로 나눠져있네요.
purpose는 돈을 빌린 목적이 나와있네요. debt_consolidation은 돌려막기인가요? credit_card? 안타깝습니다. wedding도 보이고 medical 도 보입니다. 각자의 사정이 떠올라 영화 한 편 나올 것 같습니다.

만약 description 같은 불특정 text가 대량 나오면 자연어처리에서 text를 단어별로 벡터화 하는 방식으로 연구가 필요할 겁니다. 궁금해하실 분들을 위해 잠깐 언급해봤습니다.

```
>>> len(train.grade.unique())
7
>>> len(train.purpose.unique())
12
```

몇 개나 될까 봤습니다. 각 일곱 개, 열 두 개네요.
label encoder로 숫자로 바꿔보면 각각 0-6, 0-11 로 변환되겠네요.
곧 보겠습니다.

```
train.drop(['id', 'inactive_loans'], axis=1, inplace=True)
# 또는 train = train.iloc[:,1:7]
```

먼저 필요없는 column ‘id’ 와 ‘inactive_loans’ 두 개를 없애줍니다.
default 설정이 axis=0(row) 이기 때문에 axis=1(column)로 지정합니다.
inplace=True를 이용해 train 자체를 바꿔주는 것도 했죠? 처음인가요?
아래 주석에 train.iloc[:, 1:7] 처럼 바꿔줘도 됩니다.
iloc은 positional based indexing이 가능하게 해줍니다.
: 는 전체(모든 row),
1:7은 첫 번째부터 여섯 번째 column까지 선택한다는 말입니다.
loc 도 있는데 나중에 나오면 하시쥬.

```
enc = preprocessing.LabelEncoder()
enc2 = preprocessing.LabelEncoder()
train['grade'] = enc.fit_transform(train['grade'])
train['purpose'] = enc2.fit_transform(train['purpose'])

train.grade.unique()
train.purpose.unique()
```

PCA 당시에도 썼던 label encoder입니다.
이번에는 fitting 할 게 두 가지라 encoder도 두 개 생성하고 적용했습니다.
이전 PCA 할 때는 컬럼 고유값이 2개(cluster 1, cluster2)여서 한 column에 0,1로 표현이 됐습니다.
하지만 이번에는 한 컬럼에 0-6, 0-11 레이블 될 겁니다.

모델에 학습시킬 때는 이렇게 여러 레이블이 있으면 안 됩니다.
왜냐하면, 종류에 따라 1, 4 처럼 숫자를 매기려는 것 뿐인데 모델은 4가 1의 네 배. 이런 식으로 숫자로 인식하기 때문입니다. 만약 키를 0-6등급까지 나타낸다면 4등급 키가 1등급 키의 네 배라고 인식하는 거죠.

그래서 다음 one-hot encoding을 합니다.

```
onehot1 = preprocessing.OneHotEncoder()
onehot2 = preprocessing.OneHotEncoder()

oh_grade = onehot1.fit_transform(np.array(train['grade']).reshape(-1, 1)).toarray()
oh_purpose = onehot2.fit_transform(np.array(train['purpose']).reshape(-1,1)).toarray()

oh_grade_df = pd.DataFrame(oh_grade, columns=['A','B','C','D','E','F','G'])
oh_purpose_df = pd.DataFrame(oh_purpose)

oh_grade_df.head()
oh_purpose_df.head()
```

마찬가지 두 column에 적용해야하니 encoder 두 개 설정합니다.
너무 많은 이름이 필요한데 작명센스가 부족해 자존감에 위기를 겪고 있습니다.
좀 복잡해졌는데 이건 왜 이렇게 해야하지, 이렇게 하면 안 되나 생각이 나실 겁니다.
마음껏 시도해 봅시다.
저는 다만 에러 메세지가 이끄는대로 몸을 맡겼을 뿐입니다.

첫 번째 문단: encoder 지정.
두 번째 문단: 컬럼별 fit_transform 하고 reshape로 column을 하나로 만들고 array로 변환.
세 번째 문단: 위에서 저장해 놓은 array를 dataframe으로 변환.(나중에 train과 합치려고요..)

네 번째 문단: 만든 게 어떻게 생겼나 확인.

```
train = pd.concat([train,oh_grade_df,oh_purpose_df], axis=1)
train

train.drop(['grade','purpose'],axis=1,inplace=True)
train

train.drop(['A',0], axis=1, inplace=True)
train
```

위에서 만든 dataframe을 기존 dataframe인 train 과 합치겠습니다.
옆으로 갖다 붙이는거니 pd.concat을 쓰고, 대상은 list로 묶어주며, axis=1로 합니다.
train을 확인해 보시쥬.

그리고 이제 필요없는 grade와 purpose column을 지웁니다.
또 one hot encoding으로 만든 column 중 하나씩, 총 두 개를 지울 건데요.
설명을 위해 아래 그림을 보겠습니다.

```
>>> np.eye(5)
array([[1., 0., 0., 0., 0.],
       [0., 1., 0., 0., 0.],
       [0., 0., 1., 0., 0.],
       [0., 0., 0., 1., 0.],
       [0., 0., 0., 0., 1.]])
```

```
>>> np.eye(5)[:,1:]
array([[0., 0., 0., 0.],
       [1., 0., 0., 0.],
       [0., 1., 0., 0.],
       [0., 0., 1., 0.],
       [0., 0., 0., 1.]])
```

np.eye는 항등행렬을 나타내는데요. one-hot encoding 한 결과라고 보겠습니다.
두 번째 그림은 항등행렬에서 한 column을 없앤 겁니다.
각 row가 obs.라 하면 두 가지 모두 5가지 obs. 가 따로 구분이 됩니다.
그래서 적은 column으로 같은 효과를 적용할 수 있습니다.
one-hot 을 하면 column이 너무 많아지기 때문에 하나라도 줄이고 싶어 이렇게 합니다.

```
y = np.array(train['bad_loans'])  # index가 없게
X = train.drop('bad_loans', axis=1)

# train, test set 8:2로 나누기.
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
                                                    random_state=None, stratify=y)
```

전처리가 거의 끝나갑니다.
target이 되는 bad_loans 를 y로 지정합니다. 그냥 train['bad_loans']로 해도 되지만 index가 남아있는 게 혹시 문제될가 array로 바꿨습니다.

bad_loans 를 제외한 나머지를 X로 저장합니다.
이제 train, test set을 나눌 차례인데요.
train_test_split 모듈은 데이터를 랜덤으로 골라 나눠주고, 4개 값을 return해 줍니다.
X,y 각각 train, test set을 리턴해주네요. 순서는 달라지면 안 되는데, 당연히 외우지 마시고 문서를 참고하세요.

parameter를 보시면 test_size가 보이시죠? 전체의 20%를 test set으로 삼겠다는 말입니다.

random_state 는 random_seed를 말합니다. 숫자를 지정하면 해당 랜덤 픽을 기억해놨다가 다음에 쓸 수 있습니다. 나중에 필요할 때가 있겠지유.

stratify는 각 label class마다 분포 비율을 고려해 나눠줍니다. 지금은 0,1로 나눠져있으니 두 class의 비율을 유지해 줄 겁니다.

```
scaler = MinMaxScaler(feature_range=(0.0, 1.0))
X_train = scaler.fit_transform(X_train)   # train set에 fitting & transform
X_test = scaler.transform(X_test)         # fitting 된 scaler 로 transform
```

다음은 scaling 할 차례입니다. 이번에는 전체를 0과 1 사이로 좁혀주는 scaler를 써 봅니다.
전에 말씀드렸던 fit과 transform 을 나눠서 할 때가 지금이네요.
X_train 을 fit_transform 하고,
이미 fit 된 scaler를 그대로 써서 X_test는 transform 만 해 줍니다.

이로써 전처리가 다 끝났습니다. 간단하쥬?
이제 모델을 보겠습니다.

```
model = MLPClassifier(hidden_layer_sizes=(500,500,100),
                      activation='relu', solver='adam', alpha=1e-5,
                      batch_size='auto', learning_rate='constant',
                      learning_rate_init=0.0001,
                      power_t=0.5, max_iter=10, shuffle=True, random_state=None,
                      tol=0.00001, verbose=True, warm_start=False, momentum=0.9,
                      nesterovs_momentum=True, early_stopping=False,
                      validation_fraction=0.1,
                      beta_1=0.9, beta_2=0.999, epsilon=1e-08)
```

classification 문제를 해결하기 위해 classifier를 가져와야하는데요.
이번에는 MLPClassifier(Multi-Layer Perceptron) 입니다. hyperparameter가 정말 많습니다.
지금은 모르시지만 하나씩 알아가는 재미를 느낄 게 이렇게나 많구나 하고 넘어가시면 되겠습니다.
다른 classifier는 hyperparameter가 이렇게 많지 않으니 걱정마세유.

```
model.fit(X_train, y_train)


predictions = model.predict(X_test)
predictions[:10]
probs = model.predict_proba(X_test)
probs[:10]
```

MLPClassifier를 model 에 저장했었습니다.
model.fit으로 바로 돌려볼 수 있습니다.
진행과정이 나오죠? 알아서 다 하쥬? 신기합니다.

model.predict 는 0, 1 결과를 예측해줍니다.
model.predict_proba 는 결과를 예측하기 위해 계산된 확률을 보여줍니다.
1이 될 확률이 얼마나 되면 1로 예측하고 얼마면 0으로 판단하는 걸까요?

```
>>> probs[predictions ==1]
array([[0.48699616, 0.51300384],
       [0.49240546, 0.50759454],
       [0.48526392, 0.51473608],
       [0.47687132, 0.52312868],
       [0.49392199, 0.50607801],
       [0.496164  , 0.503836  ],
       [0.43823746, 0.56176254],
       [0.47915968, 0.52084032],
       [0.49005789, 0.50994211],
       [0.48317275, 0.51682725],
       [0.48364086, 0.51635914],
```

prediction 결과가 1인 것만 확률을 봤습니다. 오른쪽이 1이 될 확률입니다.

```
>>> probs[predictions ==1][:,1]
array([0.51300384, 0.50759454, 0.51473608, 0.52312868, 0.50607801,
       0.503836  , 0.56176254, 0.52084032, 0.50994211, 0.51682725,
       0.51635914, 0.51762029, 0.50908796, 0.5138065 , 0.5310087 ,
       0.52502869, 0.520734  , 0.52437692, 0.53071998, 0.50674544,
       0.54763782, 0.51935777, 0.51091165, 0.52532483, 0.50766538,
       0.53286579, 0.52311394, 0.50069442, 0.53090137, 0.51960824,
       0.50420574, 0.55001181, 0.54803206, 0.51444185, 0.50480915,
       0.50262595, 0.505329  , 0.53574723, 0.50456106, 0.50925619,
```

1이 될 확률만 보면 이렇네요.

```
>>> probs[predictions ==1][:,1].min()
0.500159829739133
```

보아하니 0.5 이상이면 1이라고 판단을 하나 봅니다.

이런 판단 기준은 threshold 라 하고, 이 값을 조절하면서 결과를 도출합니다.
어떤 결과를 보고 정해야할까요?
곧 보겠습니다. 그 전에 threshold를 0.2로 하고 prediction을 바꿔봤습니다.

```
threshold = 0.2
threshold_preds = []
for i in range(len(probs)):
    if probs[i][1] >= threshold:
        threshold_preds.append(1)
    else:
        threshold_preds.append(0)
```

for 문 좋아하시는 분들을 위해 준비했습니다.
probs에서 1이 될 확률이
threshold(0.2)를 넘으면 threshold_preds에 1을 추가.
넘지 않으면 threshold_preds에 0을 추가합니다.
list에 원소 추가는 리스트.append(원소) 를 씁니다.

```
a = probs[:, 1]
a[a >= threshold] = 1
a[a < threshold] = 0
a
```

하지만 저는 for문을 싫어합니다. 대수 계산은 벡터 연산을 최대한 활용하도록 합니다.
이렇게 해도 되고요.

```
b = np.where(probs[:, 1] < threshold, 0, 1)
```

이렇게 해도 됩니다.
probs 에서 1이 될 확률이 threshold보다 작으면 0, 아니면 1이 되도록 하는 구문입니다.

```
accuracy = metrics.accuracy_score(y_test, predictions)
accuracy
accuracy = metrics.accuracy_score(y_test, threshold_preds)
accuracy
accuracy = metrics.accuracy_score(y_test, a)
accuracy
accuracy = metrics.accuracy_score(y_test, b)
accuracy
```

0, 1을 맞춘 비율을 accuracy. 정확도라고 할 겁니다.
기존 threshold가 0.5일 때의 predictions 가 갖는 accuracy와
0.2일 때 한 accuracy를 비교해 보시죠.

```
>>> accuracy = metrics.accuracy_score(y_test, predictions)
>>> accuracy
0.8111414705762408
>>> accuracy = metrics.accuracy_score(y_test, threshold_preds)
>>> accuracy
0.6398597120835202
>>> accuracy = metrics.accuracy_score(y_test, a)
>>> accuracy
0.6398597120835202
>>> accuracy = metrics.accuracy_score(y_test, b)
>>> accuracy
0.6398597120835202
```

기존은 0.81, threshold를 0.2로 바꾸니 0.64 정도가 됐습니다.

```
print("The model has {0}% accuracy.".format(accuracy*100))
```

이런 식으로 표현해 줄 수 있고요.

하지만 사실 accuracy는 좋은 지표가 아닙니다.

이제 classification의 결과를 논하는 주요 지표를 볼텐데요.
해당하는 내용이 꽤 기니, 우선 지금까지 전처리 - 모델 빌딩 - 학습 - 결과 구조를 마음 속에 새기시기 바랍니다.

```
metrics.confusion_matrix(y_test, a)
```

```
print(metrics.classification_report(y_test, a))
```

binary classification의 결과를 표시한 confusion matrix와
confusion matrix에서 나오는 지표인 precision - recall, 그리고 f1 score를 보겠습니다.

우선 confusion matrix부터 소개하겠습니다.
매우 헷갈리기 때문에 이름조차 confusion matrix라는 운명을 타고난 아이입니다.
절대 외우지 마시고 이해만 하시기 바랍니다.

		Predicted class		
		0	1	
Actual Class	0	True Negative	False Positive	# y=0
	1	False Negative	True Positive	# y=1
		# y*=0	# y*=1	

binary classification(0, 1 분류)에서는 주로 관심있는 대상을 1로 지정합니다.
1: (+), positive, true, malignant, default
0: (-), negative, false, benign, default 아님
이런 식입니다.

실제 class가 1인데 모델이 1이라고 예측한 게 True Positive 입니다.
실제 class가 0인데 모델이 1이라고 예측하면 False Positive 가 되겠쥬.

실제 class가 1인데 모델이 0이라고 예측한 게 False Negative 입니다.
실제 class가 0인데 모델이 0이라고 예측하면 True Negative 가 되겠쥬.

정확도(accuracy)는 전체 중에 TP, TN 인데요.
하지만 오히려 중요한 건 FP, FN입니다.
왜 accuracy가 중요한 게 아니라 틀린 것에 주목해야하는지 예시를 통해 알아보겠습니다.

ex) Default status		Predicted class		
		0	1	
Actual Class	0	9644	23	9667
	1	252	81	333
		9896	104	10000

적당한 숫자를 넣은 default 예시 자료입니다. 1 = default고요.
FP: default 라 예상했는데 실제로는 잘 갚은 사람입니다. 23명이네요.
FN: 잘 갚으리라 예상했는데 default 난 사람입니다. 252명이 돈을 빌려가놓고 안 갚았네요.

default 난 사람이 10000명 중 333 명이기 때문에
모두 0이라고 예측해도 오차율이 3.33%밖에 안 됩니다.
날씨도 항상 맑다고 해도 정확도가 별로 차이가 안 난답니다.
암도 마찬가지로 음성이 훨씬 많아 비슷한 상황입니다.

하지만 FN은 어떤가요.
333명의 default 인원 중에 252명을 못 맞췄습니다.
대출회사 입장에선 이 숫자가 중요합니다. 빌려가서 안 갚으면 그대로 손해니까요.
252/333 = 75.7%의 오류율을 보이네요. 반대로
81 / 333 = 24.3%의 예측률을 가집니다.

아까 threshold 에 대해 말씀드렸는데요.
이번에 threshold 를 바꿔본 결과를 보겠습니다.
0.5에서 0.2로 바꿨습니다.

ex) Default status		Predicted class		
		0	1	
Actual Class	0	9432	235	9667
	1	138	195	333
		9570	430	10000

threshold가 0.2로 바뀌고 숫자들이 변했는데 감지하셨나요?
이번엔 138/333 = 41.4% 로 오류율이 크게 낮아졌습니다.
예측률은 195/333 = 58.6%로 많이 올라갔네요.
대신 FP는 235명이 되어, 전체 오류율은 (138+235)/10000 으로 이전보다 높아졌습니다.
하지만 이게 회사 입장에서는 더 쓸만하겠쥬?

지금 눈여겨 본 것이 Recall 입니다. FN에 초점을 맞춘 지표입니다.
실제 1 중에 잘 예측 된 비율이지유.
여기서는 195/333 입니다.
반대로 FP에 초점을 맞춘 지표도 있습니다. 바로 Precision 입니다.
1로 예측한 애들 중 진짜 1인 애들인데요.
여기서는 195/430 입니다.

회사는 threshold를 조정해가며 자기 입장에서 더 중요한 지표를 높여 씁니다.

Precision과 Recall 모두 높을 순 없을까요?
그건 모델이 그냥 성능이 좋은 겁니다.
하지만 만든 모델 안에서 threshold를 조정하면
Precision이 높아지면 Recall이 낮아지고
Precision이 낮아지면 Recall이 높아집니다.
둘은 trade off 관계에 있다고 하쥬.

예를 들어 봅시다.

1. threshold를 0.5 -> 0.7 높이면? Higher Precision, Lower Recall.

   이건 확률이 더 높게 예측된 것을 1로 분류하겠다는 얘깁니다.
   원래 0인데 모델이 1이라고 예측했을 때의 위험이 큰 경우일 겁니다.
   뭐가 있을까요. 암이 없는데 있다고 하면 큰일인가요? 잘못 배를 가르면 안 되쥬.
   그래서 매우 확실할 때, 확률이 높을 때 1이라고 분류하겠습니다.
   threshold를 높였으니, 더 높은 확률을 가진 애들만 1로 분류하고 낮은 애들은 다 0으로 분류했습니다.
   그러면 1이라고 예측한 애들이 더 잘 맞겠죠? Precision이 올라갑니다.
   y=1이라고 예측하는 수가 줄어드니 Recall은 줄어들고요.

2. threshold를 0.5 -> 0.3으로 낮추면? Higher Recall, Lower Precision.

   이번에는 False Negative를 줄이자입니다.
   확률이 좀 낮아도 1로 분류해서 실제 1을 놓치는 비율을 낮추는 거죠.
   뭐가 있을까요. 암이 있는데 없다고 하면 큰일이쥬.
   threshold를 낮추면, 1인 사람을 1이라 예측한 수가 늘어나니 TP가 높아져 Recall이 높아집니다.
   반대로, 1이라 예측한 사람 중 틀린 비율이 늘어날테니 FP가 높아져 Precision이 낮아지고요.

헷갈려서 시간이 좀 걸립니다 이건.
그냥 Precision, Recall, 그러니까 FN, FP가 중요하다 정도만 알고 계시다가 실제 사용할 때만 찾아보도록 합니다.

그러면 어떻게 threshold를 정해야할까요?
매 알고리즘마다 threshold를 조정하며 적절한 Precision과 Recall을 찾는 건 매우 힘듭니다.
의사결정에 시간이 많이 걸리기도 하고요.
하나의 metric으로 적절한 threshold를 찾을 수 있으면 좋겠는데요.

	Precision(P)	Recall(R)	Average	F_1 score
Algorithm1	0.5	0.4	0.45	0.444

Algorithm2	0.7	0.1	0.4	0.175
Algorithm3	0.02	1.0	0.51	0.0392

각자 다른 알고리즘 3개, 혹은 같은 알고리즘에서 threshold가 다른 3개를 생각해 볼까요.
알고리즘 1번과 2번을 비교해봅시다.
1번은 0.5 / 0.4인데 2번은 0.7 / 0.1입니다. 만들어놓고 보면 뭐가 더 나을까, 조금 더 조절해봐야할까, 이번 경우에는 P가 더 높아야할까, 그렇다고 R을 무시할 순 없는데, 이런 생각들을 할 겁니다. 답이 없고, 시간도 오래 걸리는 과정이쥬.
하나의 값으로 나타내 비교할 수 있으면 그래도 훨씬 빨라질 겁니다.

자연스럽게 평소에 하던 것처럼 평균을 내볼까요?
Average column이 바로 그겁니다.
하지만 이는 좋지 않은 metric이라는 것이 밝혀졌습니다.

알고리즘 3을 보시면 Recall이 1입니다. 이는 모든 경우를 y=1이라고 예측한 건데요.
Precision은 매우 낮아질 겁니다. 그래서 0.02네요.
반대로 threshold를 높게 잡아 대부분을 y=0이라고 예측하면 1에 가까운 Precision과 0에 가까운 Recall을 얻을 겁니다.

Average column을 보시면 알고리즘3이 가장 높은 걸 보실 수 있습니다. 좋은 classifier가 아닌데도 말이죠. 분명 1,2번 알고리즘이 더 좋은 classifier일 텐데요.
그럼 Average 말고 어떤 걸 쓸 수 있을까요?

그게 바로 F Score입니다. 식은

$$F_1 \text{ score: } 2\frac{PR}{P+R}$$

입니다. 조화평균이라고 기억나시나요?
P 나 R 이 0에 가까우면 F score도 0에 가깝구유. 둘 다 커야 F score도 커집니다. 1에 가까워지는 거쥬.
주로 이 F score를 씁니다.

위 표에서 F_1 score column을 보시면 적절하게 수치를 매겨주는 거 같죠?
F score에 따르면 Algorithm 1을 선택하겠군요.

이제 파이썬 결과를 다시 살펴보겠습니다.

```
>>> metrics.confusion_matrix(y_test, a)
array([[12243,  7648],
       [ 1636,  2994]], dtype=int64)
```

아주 불친절한 결과표현입니다. 하지만 위의 표와 함께라면 이 역경도 헤쳐나갈 수 있습니다.

		Predicted class		
		0	1	
Actual Class	0	True Negative	False Positive	# y=0
	1	False Negative	True Positive	# y=1
		# y*=0	# y*=1	

위의 불친절한 파이썬 결과와 매치시켰습니다.
사실 Actual / Predicted 위치와 0/1 위치를 바꿔서 여러분을 혼란에 빠뜨릴 수 있었지만 참았습니다.
사실 처음 설명 때 반대로 만들었지만 다시 바꾼 겁니다. 어쨌든 이렇다는군요.

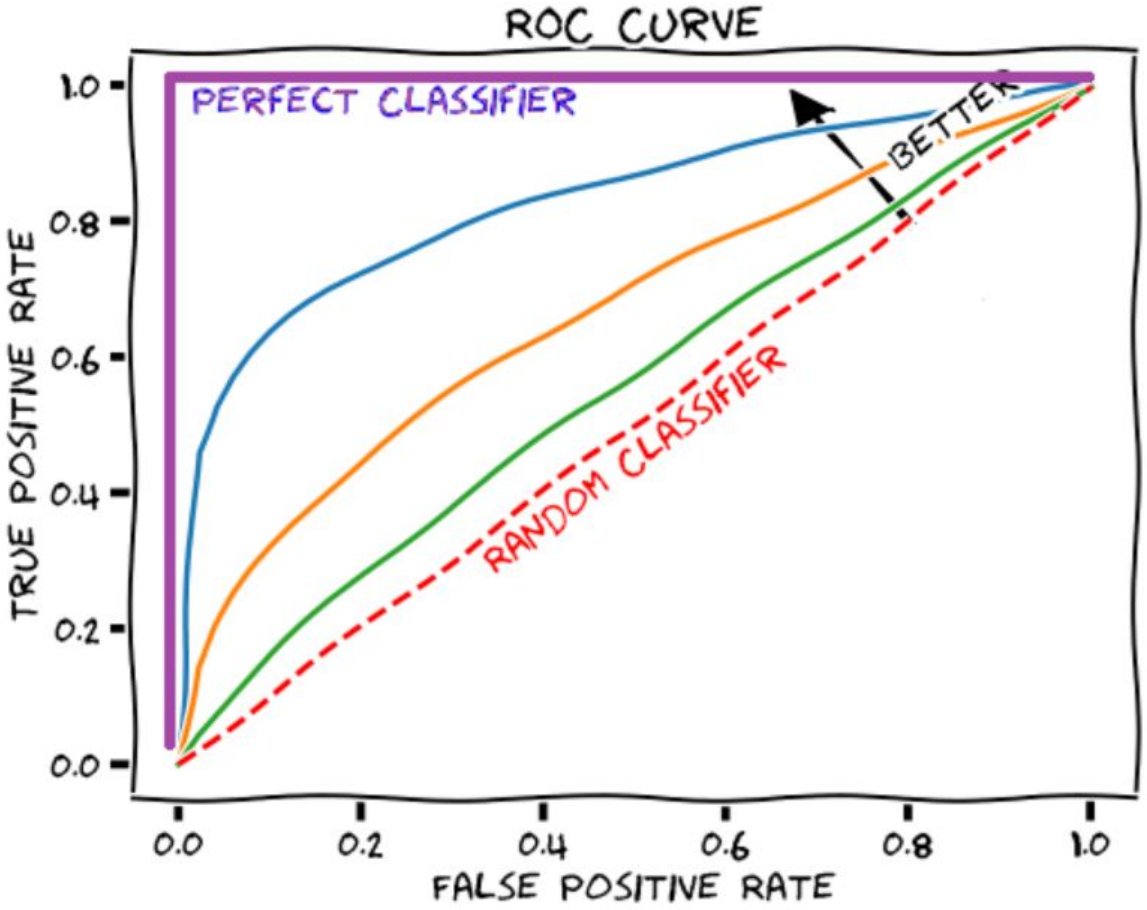
```
>>> print(metrics.classification_report(y_test, a))
              precision    recall  f1-score   support

           0       0.88      0.62      0.73     19891
           1       0.28      0.65      0.39      4630

    accuracy                           0.62     24521
   macro avg       0.58      0.63      0.56     24521
weighted avg       0.77      0.62      0.66     24521
```

Precision, Recall, f1 score까지 계산된 report도 볼 수 있습니다.
지금은 1 기준으로 봐야겠쥬?
이건 threshold가 0.2 기준이었습니다.
기존의 0.5 기준도 한 번 보고 비교해볼까요?

```
>>> print(metrics.classification_report(y_test, predictions))
              precision    recall  f1-score   support

           0       0.81      1.00      0.90     19891
           1       0.51      0.01      0.02      4630

    accuracy                           0.81     24521
   macro avg       0.66      0.50      0.46     24521
weighted avg       0.76      0.81      0.73     24521
```

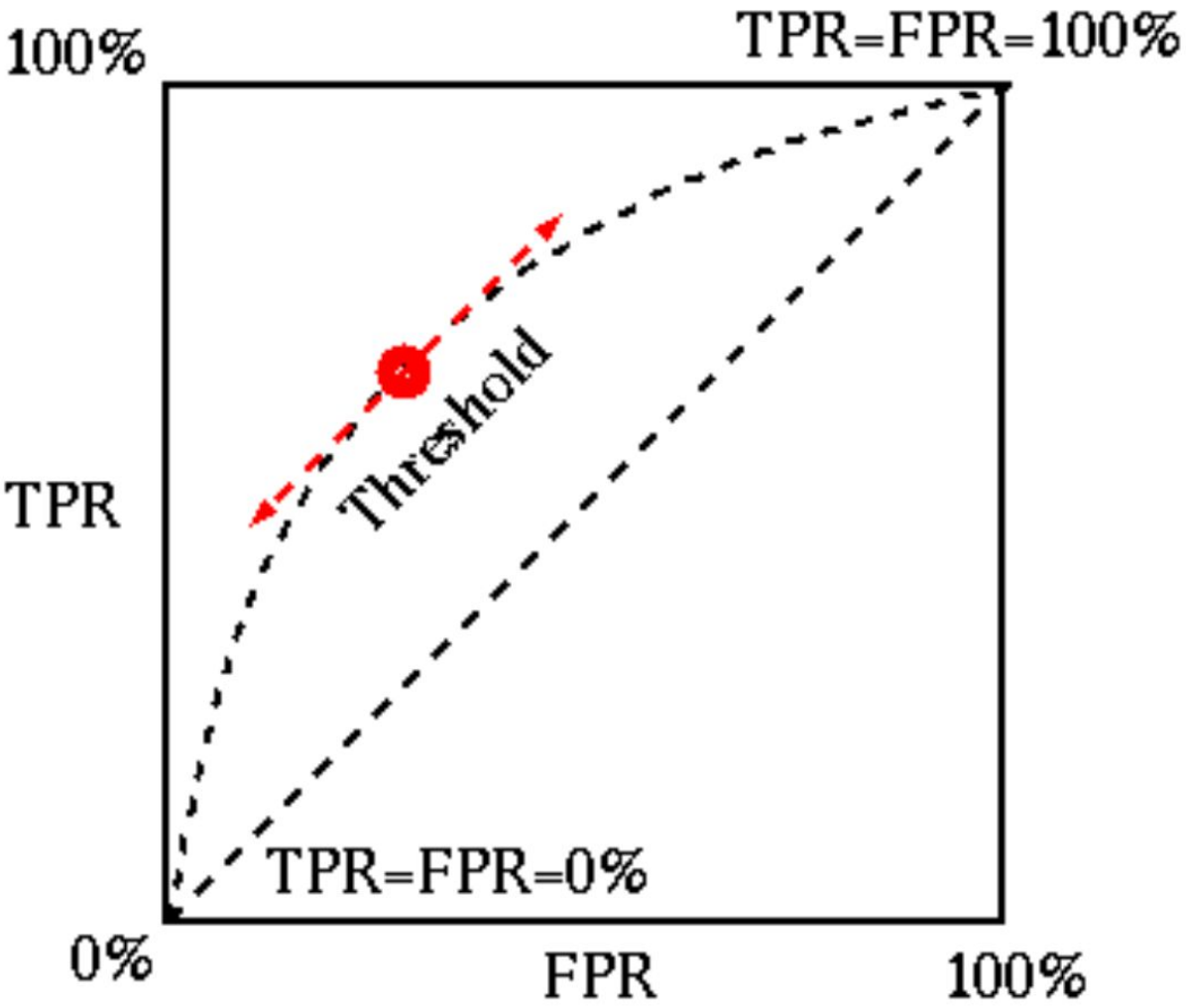
accuracy는 높은데… 나머지가 안 좋았네요. f score도 0.02로 매우 안 좋습니다.
threshold 조절이 확실히 필요하겠네요.

이번에는 ROC curve (Receiver Operating Characteristic curve) 에 대해 알아보겠습니다.
얘가 어떻게 나오는지는 너무 설명이 길어지기도 하고 굳이 알 필요가 없을 것 같기도 해서 일단 생략하고 그림만 간략하게 알려드리겠습니다.

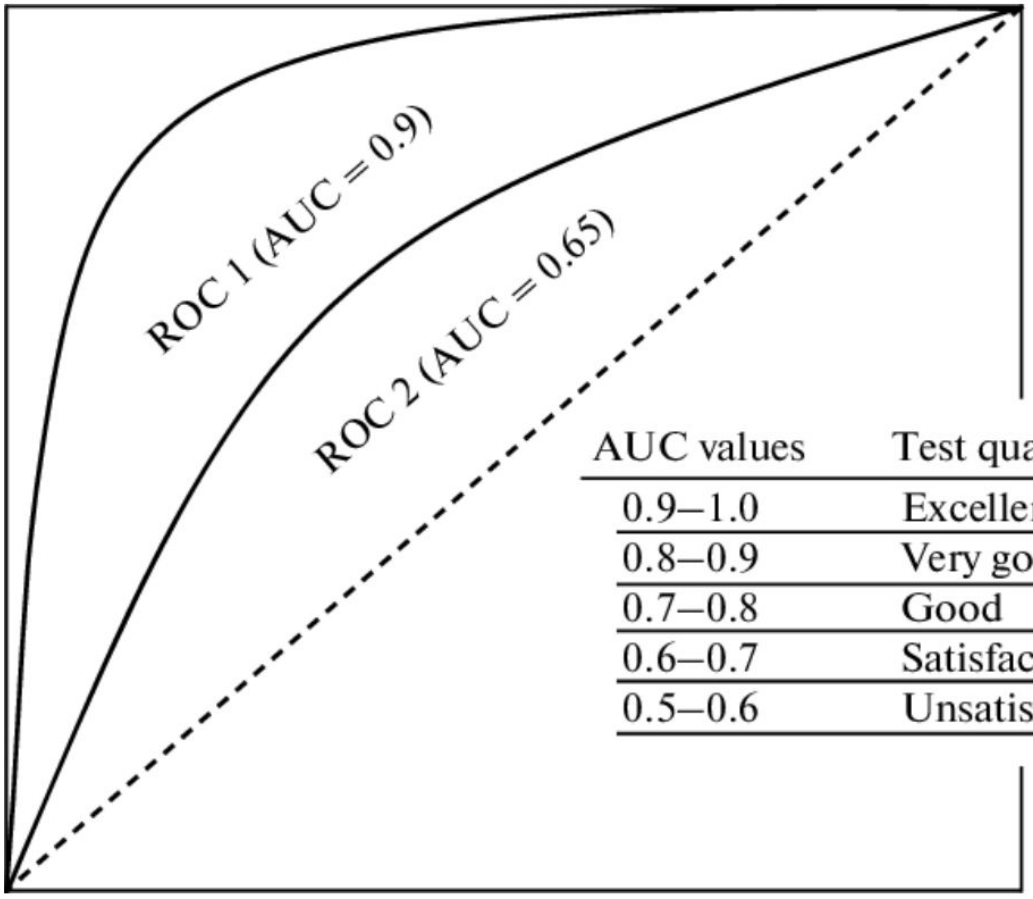


가로축이 False Positive Rate, 세로축이 True Positive Rate 입니다.
뭔지 모르시겠지만 넘어가도록 합니다.

좋은 classifier일수록 곡선이 왼쪽 위로 휘어집니다.



해당 classifier에 대한 곡선이 그려지고요. threshold를 조절하면 그 곡선을 따라 이동합니다.
근데 이건 별로 안 중요합니다.



AUC values	Test quality
0.9–1.0	Excellent
0.8–0.9	Very good
0.7–0.8	Good
0.6–0.7	Satisfactory
0.5–0.6	Unsatisfactory

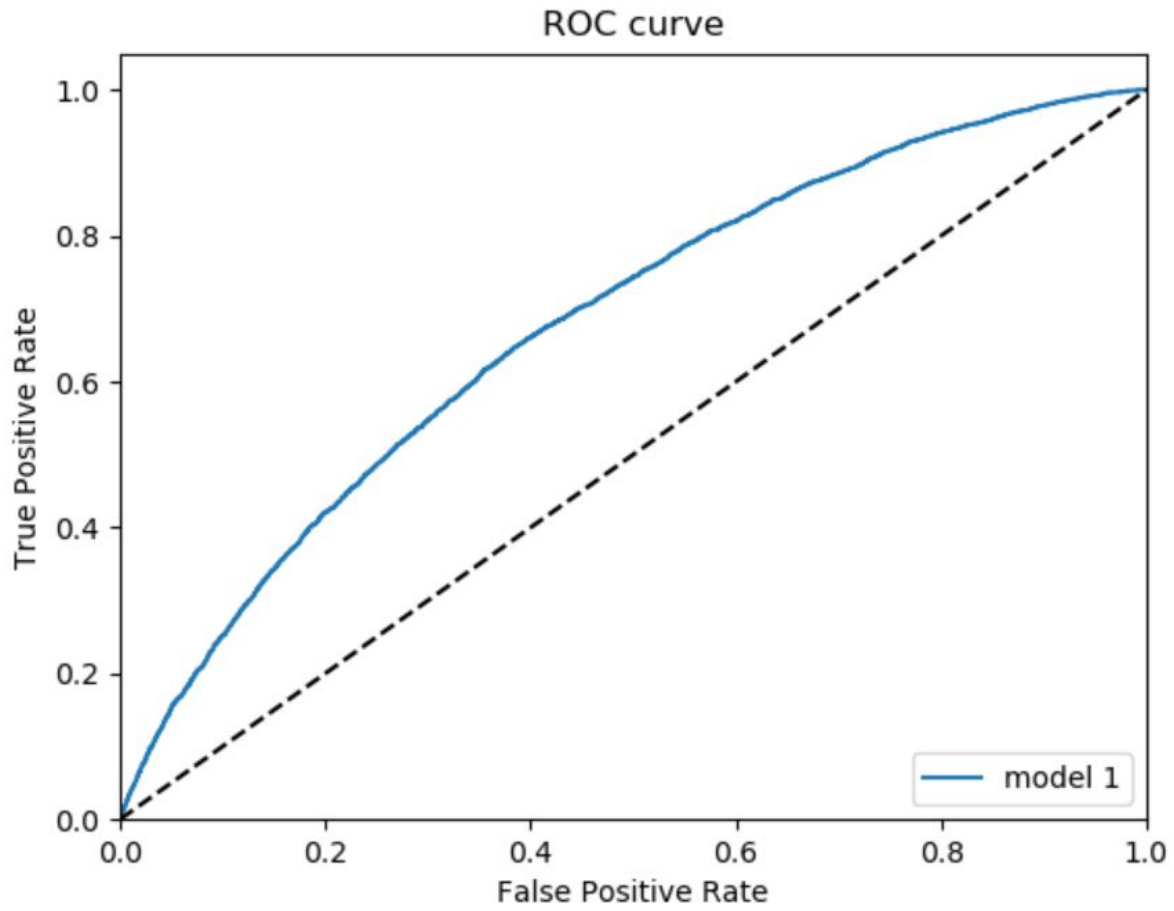
ROC curve 아래 영역의 넓이가 중요합니다. AUC(Area Under Curve) 입니다.
이전에 봤던 f score처럼 하나의 metric으로 쓰이는 AUC 입니다.
여기저기서 그림을 퍼와서 곧 쇠고랑을 차진 않을까 매우 걱정이 됩니다.
면회는.. 오실 거지유?

```
from sklearn.metrics import roc_curve, auc
fpr, tpr, cutoff = roc_curve(y_test,model.predict_proba(X_test)[:, 1])
auc(fpr, tpr)
```

이제 파이썬에서 roc curve 를 그려보고 auc를 구해봅시다.
roc_curve는 x축, y축에 해당하는 fpr, tpr 값들을 구해줍니다.
cutoff는 다른 건데 일단 넘어가시고요.
auc(fpr, tpr)로 auc를 바로 구할 수 있습니다.

```python
import matplotlib.pyplot as plt
plt.figure()      # 그래프 한 장 그릴거다.
plt.plot(fpr, tpr, label = 'model 1')   # roc curve
plt.plot([0,1], [0,1], 'k--')    # 45도 점선
plt.xlim([0.0, 1.0])  # x축 제한
plt.ylim([0.0, 1.05])# y축 제한
plt.xlabel('False Positive Rate')  # x축 이름
plt.ylabel('True Positive Rate')   # y축 이름
plt.title('ROC curve')      # 그래프 제목
plt.legend(loc="lower right")      # 범주 표시. loc: 위치
plt.show()       # 그린 거 보여달라.
```

간단하게 roc curve를 볼 수 있습니다.
사실 plt.plot(fpr, tpr) 이면 그려지긴 하는데.. 나머지는 너무 거추장스럽나요?
그래도 그럴듯해 보이니까 한 줄씩 차근차근 작성해봅시다.



이런 커브가 완성됐나요?

여기까지가 분류모델을 돌리는 전체 구조입니다.
자료 살펴보기 - 모델에 필요한 전처리 - 모델 학습 - 학습된 모델로 예측 - 결과 metric & plot

이제 태블로에 deploy 하는 것만 남았습니다.

```
def loanclassifierfull( arg1,  arg2,  arg3,  arg4,  arg5):
    import pandas as pd

    d = {'1-grade': _arg1, '2-income': _arg2,
         '3-sub_grade_num': _arg3, '4-purpose': _arg4, '5-dti': _arg5}
    df = pd.DataFrame(data=d)

    df['1-grade'] = enc.transform(df['1-grade'])
    df['4-purpose'] = enc2.transform(df['4-purpose'])

    new_grade = onehot1.transform(np.array(df['1-grade']).reshape(-1, 1)).toarray()
    new_purpose = onehot2.transform(np.array(df['4-purpose']).reshape(-1, 1)).toarray()

    new_grade_df = pd.DataFrame(new_grade, columns=['A', 'B', 'C', 'D', 'E', 'F', 'G'])
    new_purpose_df = pd.DataFrame(new_purpose)

    df = pd.concat([df, new_grade_df, new_purpose_df], axis=1)

    df.drop(['1-grade', '4-purpose'], axis=1, inplace=True)
    df.drop(['A', 0], axis=1, inplace=True)

    df = scaler.transform(df)

    probs = model.predict_proba(df)
    return probs[:, 1].tolist()
```

이번에는 5개의 변수만 쓰기로 했었쥬? _arg1 - _arg5 까지 받습니다.
내부에서 dataframe 처리를 위해 pandas를 불러옵니다.
d 에 가져온 column들을 dictionary 형태로 집어넣습니다.
df 에 d를 dataframe으로 변형해서 담고요.

이전에 해 줬던 전처리를 간단하게 해 줍니다.
enc 기억나시나요? enc2는요?
onehot1? onehot2는요?

이번엔 fit 하는 과정이 없이 전부 transform 입니다.
training set에 fit된 parameter들을 그대로 쓰니까요!

new_grade, new_purpose 모두 dataframe으로 만듭니다.

그리고 기존의 df와 다 합쳐줍니다. 옆으로 붙이는 거죠. 그게 pd.concat입니다.
axis=1에 주의하세요. 아니면 밑으로 붙이거든요.

이제 필요없는 column들을 없애주면 기존의 모델에 넣어 결과를 얻을 수 있습니다.
마지막에 scaler도 빠뜨리지 않고 해줍니다.

확률을 저장해서 그 중 1이 되는 확률을 list로 리턴해주면 끝입니다.

```
test = pd.read_csv('c:/Users/JunHyuk/downloads/TabPy materials/LoanStatsFilter.csv')
test = test.iloc[:, 1:6]
func_probs = loanclassifierfull(test.iloc[:, 0], test.iloc[:, 1],
                                test.iloc[:, 2], test.iloc[:, 3], test.iloc[:, 4])
print('Calc Results Come After This')
print(func_probs[:10])
```

deploy용 함수가 잘 돌아가나 테스트 해봅니다.
다시 자료를 test에 저장하고요.
iloc 으로 그 중 우리가 필요한 column만 선택합니다. test.iloc[:, 1:6] 은 dataframe의 1-5번 column까지만 골라줍니다.
참고로 iloc은 이렇게 positional indexing을 해준다고 하고요.
loc은 label based indexing이 됩니다. 언젠가 쓸 날이 오겠쥬. 일단 넘어가시고요.

deploy용 함수에 컬럼을 한 줄씩 변수로 넣어줍니다.
잘 돌아가는 걸 볼 수 있습니다.

```
from tabpy.tabpy_tools.client import Client

client = Client('http://localhost:9004/')
client.deploy('loanclassifierfull', loanclassifierfull,
              'Returns the probablility that a loan will result in '
              'a bad loan based on its Grade, Income, '
              'SubGradeNum, Purpose, and DTI', override=True)
```

드디어 마지막입니다. 함정에 걸리기 전에 어서 터미널에서 tabpy를 켜 줍니다.
이 부분은 세 번째쥬?
deploy(태블로에서 쓸 함수명, 파이썬 코드에서 deploy할 함수명, description, 덮어쓰기)

입니다. 끝이네요.

여기까지 classification + probability 를 예측하는 과정이였습니다.

감사합니다.